EPFL

Thèse n° 9717

Memory of Motion for Initializing Optimization in Robotics

Présentée le 28 juillet 2022

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

pour l'obtention du grade de Docteur ès Sciences

par

Teguh Santoso LEMBONO

Acceptée sur proposition du jury

Prof. C. N. Jones, président du jury
Dr J.-M. Odobez, Dr S. Calinon, directeurs de thèse
Prof. O. Brock, rapporteur
Prof. S. Coros, rapporteur
Prof. A. Ijspeert, rapporteur

École
polytechnique
fédérale
de Lausanne

2022

The fear of the LORD is the beginning of wisdom,
and the knowledge of the Holy One is insight.
— Proverbs 9:10

To my mother,
who raised me and loves me selflessly

Acknowledgements

First and foremost, I would like to thank my LORD Jesus Christ for the completion of this thesis. For many years, I can see His hand guiding me and showering me with blessings upon blessings that I do not deserve. The beauty and the order of His creation continue to inspire me to study the world and find joy in it.

Next, I would like to thank my advisors, Dr. Jean-Marc Odobez and Dr. Sylvain Calinon, for giving me the opportunity to pursue a doctorate jointly with the Idiap Research Institute and EPFL. I am very grateful especially to Sylvain for trusting me to work on the MEMMO project and for all the discussions we had over the last four years.

I would like to thank all my colleagues in the RLI group for our friendships. I really enjoy both the technical and life discussions that we shared over lunch and coffee breaks. Special thanks go to my office mates, Amir and Teng, who made my last year of PhD very fun and enjoyable. I also thank Suhan for a nice collaboration over this last year, which contributed greatly to this thesis.

I would like to extend my gratitude to the members of the MEMMO project. Being involved in this project allows me to learn so much among some of the best researchers in robotics, and I enjoy all the interactions that we had. Special thanks go to Nicolas Mansard and Carlos Mastalli who provided me with so much support and guidance during our collaboration.

I would like to thank all of my teachers, especially Pak Bowo and Pak Yahya, who really inspired me to enjoy maths and physics, both through the books that you lent me and the interesting lessons you taught me.

Finally, I would like to thank all of my family members who continue to support me. To my mother, who used to wake me up lovingly with cold water before going to school. She worked tirelessly to support the education of all her children, and I can not thank her enough for that. To my sisters, who guided me especially in my early childhood when I tend to give up easily, and helped me to walk on the right path. And most importantly, to my wife Ella and my daughter Nina, with whom I share countless joyful moments. Without them, there will be much fewer smiles and laughter during these four years.

Martigny, July 19, 2022 Teguh Santoso Lembono

Abstract

Many robotics problems are formulated as optimization problems. However, most optimization solvers in robotics are locally optimal and the performance depends a lot on the initial guess. For challenging problems, the solver will often get stuck at poor local optima without a good initialization. In this thesis, we consider various techniques to provide a good initial guess to the solver based on previous experience. We use the term *memory of motion* to collectively refer to these techniques. The key idea is to use the existing system models, cost functions, and simulation tools to generate a database of solutions, and then construct a memory of motion model. During online execution, we can then query the initial guess of a given task from the memory of motion. We show that it improves the solver performance in terms of the solution quality, the success rates, and the computation time. We consider two different formulations, i.e., supervised learning and probability density estimation.

In the first part, we formulate a regression problem to find the mapping between the task parameters and the solutions. Such a formulation is convenient, as there are a lot of function approximations available, but using them as a black box tool may result in poor predictions. It is especially the case for multimodal problems where there can be several different solutions for a given task, and standard function approximators will simply average the different modes. We first propose an ensemble of function approximators that can handle multimodal problems to initialize an optimization-based motion planner. We then investigate the problem of initializing an optimal control solver for legged robot locomotion, where we need to also provide the initial guess of the control sequence. We evaluate the effect of different initialization components on the optimal control solver performance.

In the second part, we consider another formulation by first transforming the cost function into an unnormalized Probability Density Function (PDF) and approximating it using various surrogate models. This formulation addresses several shortcomings of the supervised learning approaches by using the cost function itself to train or construct the predictive model. It allows us to generate initial guesses that have high probabilities of having low-cost values instead of simply imitating the dataset. We first show that we can obtain a trajectory distribution of an iLQR problem as a Gaussian distribution, and tracking this distribution results in a cost-efficient and robust controller. We then propose a generative adversarial framework to learn the distribution of robot configurations under constraints. Finally, we use tensor methods to approximate the unnormalized PDF.

Abstract

Since it does not rely on gradient information, the method is quite robust in finding the (possibly multiple) global optima or at least the good local optima of various challenging problems including some benchmark optimization functions, inverse kinematics, and motion planning.

Keywords: Memory of Motion, Function Approximations, Trajectory Optimization, Motion Planning, Optimal Control, Variational Inference, Tensor Methods.

Résumé

De nombreux problèmes de robotique sont formulés comme des problèmes d'optimisation. Cependant, la plupart des solveurs d'optimisation en robotique sont localement optimaux et les performances dépendent beaucoup de l'estimation initiale. Pour les problèmes difficiles, le solveur restera souvent bloqué à de mauvais optima locaux sans une bonne initialisation. Dans cette thèse, nous considérons diverses techniques pour fournir une bonne estimation initiale au solveur basée sur des expériences précédente. Nous utilisons le terme *mémoire de mouvement* pour désigner collectivement ces techniques. L'idée principale est d'utiliser les modèles de système existants, les fonctions de coût et les outils de simulation pour générer une base de données de solutions, puis de construire un modèle de *mémoire de mouvement*. Lors de l'exécution, nous pouvons alors interroger l'estimation initiale d'une tâche, donnée à partir de cette *mémoire de mouvement*. Nous montrons que cette approche améliore les performances du solveur en termes de qualité de la solution, de taux de réussite et de temps de calcul. Nous considérons deux formulations différentes : 1) l'apprentissage supervisé et 2) l'estimation de la densité de probabilité.
Dans la première partie, nous formulons un problème de régression pour trouver la correspondance entre les paramètres de la tâche et les solutions. Une telle formulation est pratique, car il existe de nombreuses approximations de fonctions disponibles, mais les utiliser comme "black box" peut entraîner de mauvaises prédictions. En particulier, dans les cas de problèmes multimodaux où il peut y avoir plusieurs solutions différentes pour une tâche donnée, les approximateurs de fonctions standard produiraient une moyenne des différents modes. Nous proposons d'abord l'utilisation d'un ensemble d'approximations de fonctions qui peuvent gérer des problèmes multimodaux pour initialiser un planificateur de mouvement basé sur l'optimisation. Nous étudions ensuite le problème de l'initialisation d'un solveur de control optimal pour la locomotion d'un robot à jambes, où nous devons également fournir l'estimation initiale de la séquence de contrôle des pas. Nous évaluons ici également l'effet de différents composants d'initialisation sur les performances du solveur.
Dans la deuxième partie, nous considérons une autre formulation en transformant d'abord la fonction de coût en une densité de probabilité non normalisés et en l'approximant à l'aide de divers modèles. Cette formulation comble plusieurs lacunes des méthodes d'apprentissage supervisé en utilisant la fonction de coût elle-même pour former ou construire le modèle prédictif. Cela nous permet de générer des estimations initiales qui ont de fortes probabilités d'avoir des valeurs à faible coût (au lieu de simplement

imiter le dataset). Nous montrons d'abord que nous pouvons obtenir une distribution de trajectoires d'un problème iLQR sous la forme d'une distribution gaussienne, et le suivi de cette distribution aboutit à un contrôleur efficace et robuste. Nous proposons ensuite un modèle de type GAN pour apprendre la distribution des configurations de robots sous contraintes. Enfin, nous utilisons des méthodes tensorielles pour approximer la densità de probabilité non normalisée. Puisque l'approche proposée ne repose pas sur des informations de gradient, cette méthode permet de trouver les optima globaux (éventuellement multiples) ou au moins les bons optima locaux de divers problèmes difficiles, y compris certaines fonctions d'optimisation utilisées comme référence, la cinématique inverse d'un robot, ainsi que la planification de mouvement.

Mots-clés : Mémoire de mouvement, Approximations de fonctions, Optimisation de trajectoires, Planification de mouvement, Contrôle optimal, Inférence variationnelle, Méthodes tensorielles.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

In robotics, an increasing number of problems are formulated as optimization problems. One important reason is that optimization offers a lot of flexibility and ease of defining the problem based on high-level requirements. For example, the problem of finding a robot configuration that corresponds to the desired end effector pose, i.e., inverse kinematics, is commonly cast as a nonlinear optimization problem [2]. In motion planning, optimization-based planners such as CHOMP [3], TrajOpt[4], and STOMP [5] have recently been proposed as alternatives to the popular sampling-based planners. In robot control, optimization-based approaches come in the form of Quadratic Programming-based controller (e.g., Task-Space Inverse Dynamics (TSID) [6]) and optimal control (e.g., Iterative Linear Quadratic Regulator (iLQR) [7]). The optimization framework allows us to easily add new tasks or requirements in the form of cost functions or constraints. With the availability of high-performance off-the-shelf solvers, increasingly complex problems are getting solved with these approaches.

While there are many benefits of such approaches, they unfortunately suffer from the following problems. Firstly, most of the available solvers can be categorized as local optimizers that can get easily stuck in poor local optima. The performance of the solver, especially in nonlinear problems, highly depends on the initial guess provided by the user. Secondly, and not unrelated to the first point, it may take a lot of time to compute the optimal solution, hindering its use for online execution. If the solver is initialized from a good initial guess, it can avoid poor local optima while at the same time reducing the computation time for convergence. In some simple problems such as base motion planning to avoid simple obstacles [8], a heuristics-based method such as initializing using pre-defined waypoints can work well but it does not apply in general cases. This motivates the need for an approach that can automatically generate a good initial guess for a given optimization problem.

Machine learning techniques seem to be very promising for that purpose. We see many applications of various machine learning techniques on robotics problems. One important weakness, though, is reliability. Most of the techniques do not provide us with a guarantee on the performance, and this can be an important deterrence from using it on the real system. Another point is about its accuracy; it is difficult, for example, to ask a machine learning system to produce a precise control sequence associated with legged robot locomotion. Indeed, there are existing works that successfully use machine learning to control real robots in challenging scenarios, but they often require a tremendous amount of data and computation. In control problems an accurate prediction is especially important, as a slight change in the control sequence (especially at the beginning) can result in a large change in the overall trajectory. These shortcomings, however, would not be crucial if the machine learning algorithm is only required to predict the initial guess instead of the final solution. The initial guess can then be further refined by the optimization-based solver, which often provides us with better reliability and accuracy.

Combining machine learning and optimization-based technique in this way offers us the best of both techniques.

This thesis aims at answering the following question: how can we use machine learning techniques to improve the speed and the solution quality of model-based optimization solvers? The main idea is that when the system models are available, we can generate or compute a lot of data offline and store it as *memory of motion*. This may take a lot of different forms, from a raw dataset [9, 10], a mixture of Gaussians [8], neural networks (Generative Adversarial Networks (GAN)) [11], Variational Auto Encoders (VAE) [12]), or Tensor Train model [13]). During online execution, we can then query from this memory of motion the initial guess to be optimized using the model-based optimization solver. In this thesis, we use the term "memory of motion" to refer to both the representation form of the dataset as well as the querying method. The requirement for the memory of motion is that the query has to be fast and the initial guess it produces should be close to the optimal solution.

The success of the memory of motion can provide significant improvements over existing robotics applications. For example, it will enable Model Predictive Control (MPC) with a highly complex model online, allowing us to generate and control increasingly complex behavior. Fast computation means that robots will be more reactive toward user commands. It also allows better adaptation capability towards large disturbances, as the memory of motion can provide us a clue on how to handle such disturbance effectively. How can we build such a memory of motion? Several issues have to be addressed, which will be discussed in the next section.

1.2 Main Challenges

1.2.1 Local Optima

Building a memory of motion with high performance requires a dataset of good quality. Unfortunately, generating the dataset remains a challenging problem. Many robotics problems are highly non-convex with a lot of local optima, and most of the solvers commonly used are local optimizers. While most of the time in robotics we do not need to find the global optima, getting stuck in poor local optima should be avoided. Some techniques allow us to find feasible solutions with probabilistic completeness guarantee, such as sampling-based planner (e.g., Rapidly-exploring Random Tree (RRT) [14], Probabilistic Roadmap (PRM) [15]), but the solutions are not optimal. The optimal versions (e.g., RRT*, PRM* [16]) are available but at much larger computational time.

1.2.2 Multimodal Data

Many robotics problems do not have a one-to-one mapping between the input (i.e., task) and the output (e.g., joint angle trajectory). For example, in Inverse Kinematics (IK) problem, a specific End Effector (EE) pose may correspond to more than one configuration for a 6-Degree of Freedom (DoF) robot and an infinite number of configurations for 7-DoF redundant robot. To build a good memory of motion, ideally, we would like to keep more than one solution to maintain the richness of the solutions. However, learning such multimodal mapping is very difficult for many learning algorithms. Standard learning algorithms such as Gaussian Process Regression (GPR) [17] or Multi-layer Perceptron (MLP) tend to average the different modalities, resulting in a poor prediction. When the modes are easily separable, it is possible to separate the dataset corresponding to different modes and train a different learning algorithm separately for each mode, but most of the time it is difficult to separate the different modes.

Generating multimodal data is also challenging in itself because most robotics solvers only provide us with one single optimal solution. While we can obtain different solutions using multiple random initializations, this remains very heuristic. An algorithm that can reliably generate multimodal outputs by efficiently exploring the whole solution space would be very useful.

1.2.3 Optimality Criteria

When building the memory of motion, ideally we should generate the dataset by using the same solver that we want to initialize. Otherwise, the dataset might be generated with a different notion of optimality from the online solver. Practically, this would mean that the initial guess produced by the memory of motion will not be considered optimal by the solver, and hence requires additional iterations for refinement, which is undesirable. On the other hand, as mentioned previously, the online solver is most of the time a locally optimal solver. This means that it often cannot find an optimal solution even when it exists. The ideal way is to build the dataset using an efficient global planner that has the same optimality criteria as the online solver, but it does not exist in general. Hence, some tradeoff needs to be made when selecting the method for generating the dataset.

1.2.4 Data Representation and Learning Methods

Assuming that we have a good dataset, how should we store it? Storing the dataset as raw data is inefficient both in terms of the memory storage and the query process. We should ideally store only its minimal representation that still allows us to produce a good quality initial guess. Dimensionality reduction techniques such as Principal Component Analysis (PCA) or basis function representation (e.g., Radial Basis Function (RBF)) can potentially be used. This representation also ultimately depends on the

learning algorithms used for the query. For example, when Gaussian Mixture Regression (GMR) is used, the dataset will be represented as a mixture of Gaussians that encode the joint distribution of the input and output. Additionally, this representation should also maintain the multimodality of the datasets, and the learning algorithm should handle such multimodal data properly without simply averaging the different modalities.

1.3 Thesis Organization

The remaining part of the thesis attempt to address the challenges presented above. We begin in Chapter 2 by explaining the relevant background knowledge related to this thesis. Chapter 3 to Chapter 7 are then divided into two parts: supervised learning approaches and probability density estimation approaches. Finally, Chapter 9 concludes the thesis with some remarks about future direction.

Part 1: Supervised Learning Approaches

In the first part, we focus on investigating supervised learning approaches to build the memory of motion. The problem of predicting the initial guess is formulated as a regression problem to learn the mapping $\boldsymbol{f} : \boldsymbol{x} \rightarrow \boldsymbol{y}$ that maps each task $\boldsymbol{x}$ to the initial guess $\boldsymbol{y}$. Datasets are firstly generated, and then techniques such as GPR, Bayesian GMR, and k-nearest neighbors are trained on the datasets. We then applied the approach to motion planning and optimal control.

In Chapter 3, we apply it to motion planning of the PR2 dual-arm robot and the Atlas humanoid robot. We demonstrate that a mixture model such as Bayesian Gaussian Mixture Regression (BGMR) can handle multimodal prediction better than GPR. We also show that the memory can be used as a metric to choose between different goals. Finally, we propose an ensemble technique combining various supervised learning techniques, resulting in a large improvement of the success rate of the motion planning problem.

Chapter 4 investigates the problem of initializing an optimal control solver based on iLQR for generating locomotion movements of a humanoid robot. We simplify the problem by learning one-step motions that can be concatenated to produce multiple step motions. We compare different initialization components and show that initializing iLQR solver with both the state trajectory and the control command sequence results in the best improvement over the cold start initialization. We also show that a dataset that has the same optimality criteria as the online solver results in better initialization performance.

Part 2: Probability Density Estimation Approaches

While supervised learning formulation is convenient due to the availability of off-the-shelf learning algorithms, it has issues in at least two aspects. Firstly, generating good data is often difficult due to the highly nonlinear nature of the problems. Secondly, many supervised learning techniques are not suitable for handling multimodal problems. In the second part, we approach the problem differently by using probabilistic density estimation approaches. We transform the cost functions to (possibly unnormalized) probability density functions and use various techniques (i.e., GAN, Variational Inference, and Tensor-Train decomposition) to approximate the probability distributions. This allows us to obtain much richer solutions compared to standard learning techniques as well as to avoid the issue of multimodal averaging.

In Chapter 5, we show that by transforming the cost of an optimal control problem into a probability density function, we can obtain a Gaussian distribution as the output of a standard iLQR solver. This distribution is then given to a short horizon MPC controller for tracking, and we show that tracking the distribution is more stable and cost-efficient compared to tracking only the optimal trajectory.

In Chapter 6, we use GAN framework to approximate the distribution of complex robot configurations under constraints using both a dataset as well as cost functions corresponding to the tasks and constraints. After training, we can easily generate robot configurations that satisfy the constraints (e.g., stable configurations of a humanoid robot). We can also condition the distributions on the desired end effector poses, obtaining the configurations that reach those poses approximately. By using an ensemble of neural networks as the generator, we can obtain multimodal distributions. We then use it in the context of IK and sampling-based motion planning under constraints.

Finally, in Chapter 7, we approach the problem of approximating the probability distribution using Tensor Train decomposition. By making use of the low-rank nature of the cost functions, we can approximate the probability using low-rank tensors that has a better capacity to approximate complex functions than existing methods, e.g., Gaussian Mixture Model (GMM). The advantage of using this tensor method is that it does not require building explicit datasets, it can approximate multimodal problems easily, it is not easily stuck in local optima, it does not require gradients, and we can condition the distribution on the given tasks. We demonstrate its advantage by applying it to various problems including the common benchmark functions for nonlinear optimization (e.g., Rosenbrock and Himmelblau functions), inverse kinematics under constraints, and motion planning. In all cases, the proposed method can find the multimodal solutions and can be conditioned online based on the tasks to quickly produce approximate solutions during online execution.

1.4 MEMMO Project

Complex movement generation for arbitrary robots with arms and legs interacting in a dynamic environment in real-time remains a challenging task for current robotics tools. The availability of such a technology would certainly revolutionize the motion capabilities of robots and unlock a wide range of very concrete industrial and service applications: robots would be able to react in real-time to any change in the environment or unexpected disturbance during locomotion or manipulation tasks. However, the computation of complex movements for robots with arms and legs in multi-contact scenarios in unstructured environments is not realistically amenable to real-time with current computational capabilities and numerical algorithms.

This thesis is a part of the collaborative H2020 European project MEMMO (Memory of Motion) that aims to solve the above problem by 1) relying on off-line caching of pre-computed optimal motions that are 2) recovered and adapted online to new situations with real-time tractable model predictive control and where 3) all available sensor modalities are exploited for feedback control going beyond the mere state of the robot for more robust behaviors. The objective of MEMMO is to develop a unified yet tractable approach to motion generation for complex robots with arms and legs.

2 Background

2.1 Optimization in Robotics

Optimization has become an indispensable tool in robotics. An increasingly large variety of robotics problems, including calibration, inverse kinematics, motion planning, control, and navigation, are formulated as optimization problems. [1] As more powerful optimization solvers become available and easy to use, the formulation grows in model complexity (e.g., from considering the centroidal dynamics to whole-body dynamics in legged locomotion) as well as the time horizon (from one time step optimization to optimizing over a time horizon).

One of the main benefits of casting a robotics problem as an optimization problem is the ease of translating the task requirement into cost functions or constraints. The formulation also often unifies several different pieces of algorithms (previously crafted manually by the programmers or the roboticists) as a single optimization problem, allowing more complex behaviors to be found. The availability of off-the-shelf optimization solvers certainly helps to push this move towards optimization. Furthermore, convenient tools for calculating the gradient of the dynamics or the cost functions are also easily available, ranging from tools that provide analytical gradient (pinocchio [19]), symbolic gradient (CasADi [20]), to automatic gradient (JAX [21], pytorch [22], tensorflow [23]). These tools alleviate the burden of researchers from crafting their own solvers to focusing more on designing the problem formulation.

However, with the increasing complexity of robotics tasks, the optimization problem becomes very nonlinear and difficult to solve. Most of the optimization solvers used in robotics (e.g., Gauss-Newton, Quasi-Newton, Interior-point Method, and Sequential Quadratic Programming) are gradient-based techniques that can easily get stuck in poor local optima without good initialization. While global optimizers such as Bayesian Optimization [24] and Genetic Algorithm [25] exist, they take a lot of computation time, precluding them from common robotics usage. This implies that for most robotics problems, the performance of the solvers depends on the initial guess provided by the users. In some cases, it is easy to craft some heuristics to generate good initial guesses, but in general, the problem still exists. This motivates the need for a general method to provide such a good initial guess to any kind of optimization problem.

In the following section, we discuss several important robotics problems formulated as optimization problems that are considered in this thesis.

2.1.1 Inverse Kinematics

IK is a problem of determining the robot configuration that corresponds to a given EE pose. For robots that have fewer or equal to 6 DoF, analytical solutions often exist and

[1] We refer to [18] for an interesting discussion regarding the use of optimization formulation in robotics.

can be obtained easily, e.g., when the last three joints intersect at a common point, or using IKFast [26]. However, for higher dimensional robots the number of solutions is infinite, and numerical IK is the standard approach. Given a desired end effector pose $\boldsymbol{p}_{\text{ref}}$ and the initial configuration $\boldsymbol{q}_0$, an iterative optimization is used to find the value of $\boldsymbol{q}$ such that the corresponding EE pose $\boldsymbol{p}(\boldsymbol{q})$ is equal to the desired pose $\boldsymbol{p}_{\text{ref}}$. One common approach is to use Gauss-Newton algorithm, often referred to as the pseudo-inverse method. In the most basic formulation, the IK problem is formulated as minimizing the following quadratic cost,

$$c(\boldsymbol{q}) = \frac{1}{2}\boldsymbol{r}^\top \boldsymbol{r}, \tag{2.1}$$

where $\boldsymbol{r} = \boldsymbol{p}(\boldsymbol{q}) - \boldsymbol{p}_{\text{ref}}$ is the residual vector to be minimized. Starting from the initial guess $\boldsymbol{q}_0$, the next solution is obtained by

$$\boldsymbol{q}_{i+1} = \boldsymbol{q}_i - \alpha \boldsymbol{J}^\dagger \boldsymbol{r}, \tag{2.2}$$

where $\boldsymbol{J} = \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{q}}$ is the Jacobian of the residual function, α is the step length, and the subscript † denotes the (Moore-Penrose) pseudo-inverse operator. The step is iterated until the residual norm is smaller than a specified threshold value.

This formulation can be extended to include more residual functions, each of which corresponds to a particular task or constraint that we want to enforce on the system. For example, in the case of IK for humanoid, we can define the cost function as

$$c(\boldsymbol{q}) = \frac{1}{2}(\boldsymbol{r}_{\text{ee}}^\top \boldsymbol{r}_{\text{ee}} + \boldsymbol{r}_{\text{s}}^\top \boldsymbol{r}_{\text{s}} + \boldsymbol{r}_{\text{l}}^\top \boldsymbol{r}_{\text{l}}),$$

where the subscripts $(\text{ee}, \text{s}, \text{l})$ refer to the residuals corresponding to the constraints on the end effector pose, static stability, and joint limit. Note that Gauss-Newton algorithm can only be applied to least-square problems, hence each of the cost term should be a square function of the residuals.

Furthermore, with the pseudo-inverse method, we can implement some hierarchy in the constraints by using the nullspace projection [27]. For each level of the hierarchy, we can define a cost function as in (2.1). Let $\boldsymbol{J}_1$ and $\boldsymbol{J}_2$ refer to the Jacobian of the residuals of the main and secondary constraints. The nullspace projection operator due to the first Jacobian, $\boldsymbol{N}_1$, can be obtained as $\boldsymbol{N}_1 = \boldsymbol{I} - \boldsymbol{J}_1^\dagger \boldsymbol{J}_1$. We can use this nullspace operator to prevent the secondary constraints from affecting the main constraints. Starting from the initial guess $\boldsymbol{q}_0$, the next solution is obtained by

$$\boldsymbol{q}_{i+1} = \boldsymbol{q}_i - \alpha_1 \boldsymbol{dq}_1 - \alpha_2 \boldsymbol{dq}_2, \tag{2.3}$$

where $\boldsymbol{dq}_1$ and $\boldsymbol{dq}_2$ are the steps corresponding to the main and secondary constraints, defined as

$$\boldsymbol{dq}_1 = \boldsymbol{J}_1^\dagger \boldsymbol{r}_1, \quad \text{and}$$

$$\boldsymbol{dq}_2 = (\boldsymbol{J}_2\boldsymbol{N}_1)^\dagger(\boldsymbol{r}_2 - \alpha_1\boldsymbol{J}_2\boldsymbol{dq}_1),$$

where (α_1, α_2) are the corresponding step lengths.

To determine the step length α_1 and α_2, we need to perform a line search. Armijo condition is often used as the stopping criteria, but other conditions also exist [28]. When there is more than one step length, the first step length (i.e., α_1) should be determined first, and the subsequent step lengths (i.e., α_2, α_3, etc.) are computed such that the tasks with the higher priority are not disturbed.

Finally, to improve the stability of the pseudo-inverse, a damping term $\lambda\boldsymbol{I}$ can be added, defined as

$$\lambda = \mu\boldsymbol{r}^\top\boldsymbol{r} + \bar{\mu}, \tag{2.4}$$

where μ and $\bar{\mu}$ are manually-defined constants. The damping term hence depends on the residual magnitude. As shown in [2], it helps the convergence when starting far from the optimum solution.

The above formulation transforms constraints into cost functions, effectively treating them as soft constraints. When hard constraints are required, the numerical IK problem can alternatively be formulated as Quadratic Programming (QP) [29].

2.1.2 Motion Planning

A typical motion planning algorithm finds a sequence of robot configurations that move from an initial configuration to the desired configuration, or a desired EE pose while avoiding collisions with the environment. Constraints such as staying within joint velocity limits or maintaining a certain orientation can also be included. To avoid obstacles effectively, the planning is usually done in the *configuration space*, i.e., the space of all robot configurations. The configuration space can be divided into the free space, where the configuration is free from collision, and the obstacle space, where the configuration collides with some obstacle(s) in the environment. Motion planning can thus be cast as finding a path through the free space, starting from an initial configuration to a final configuration.

The shape of the free space is often very complex, especially in high dimensions, and it is impossible to find an analytical expression in general. Popular algorithms that can find the path through this space effectively are sampling-based motion planners (e.g., RRT [14], PRM [30]). These planners work by iteratively sampling random configurations from the configuration space and building a graph or a tree connecting all the configurations. They are very effective and probabilistically complete, i.e., given enough samples, they will find a solution when it exists. However, the solution is only a feasible geometric motion that usually needs to be further optimized via a post-processing step. Optimal sampling-based planners such as RRT* and PRM* exist, but they take a significantly

longer computation time.

CHOMP [31] is the first work that proposes to cast the whole motion planning problem as an optimization problem. Instead of finding an initial solution via a sampling-based planner, CHOMP can accept an infeasible solution as initialization and optimizes it directly to obtain the final solution. It is followed by further work such as STOMP [5], ITOMP [32], TrajOpt [4], and GPMP [33]. The optimization formulation allows these methods to incorporate constraints naturally and more easily compared to sampling-based planners. Furthermore, the resulting motion is already (locally) optimal, so no post-processing is required.

In optimization-based planners, the problem of finding a path in the configuration space $\mathcal{R}^m$ from $\boldsymbol{q}_{\text{init}}$ to $\boldsymbol{q}_{\text{goal}}$ is formulated as the following,

$$\boldsymbol{q}^* = \underset{\boldsymbol{q}_0, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_T}{\arg\min} \sum_{t=0}^{T} f(\boldsymbol{q}_t)$$

$$s.t. \quad \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{0},$$

$$\boldsymbol{H}(\boldsymbol{q}) \geq \boldsymbol{0},$$

where $f, \boldsymbol{G}, \text{and} \boldsymbol{H}$ are the cost function, equality, and inequality constraints. While we show the discrete form here, some planners optimize the trajectory in the continuous space. Some planners also formulate the constraints as soft-constraints, i.e., putting the constraints in the form of the cost function.

In CHOMP, the cost function is split into two parts: the prior term that encourages smoothness of the trajectory and the obstacle term that steers the trajectory away from the obstacle. It also uses a covariant gradient descent (from which it derives its name) that ensures each update results in a smooth trajectory. For the obstacle avoidance, it uses *signed distance field*, i.e., a function that outputs the distance from a point $\boldsymbol{p}$ to the boundary of the nearest obstacle. Its values are negative inside the obstacles, positive outside, and zero at the boundary. Since computing such functions for general obstacles can be difficult and time-consuming, CHOMP relies on an offline-precomputed 3-D array that store the value of the signed distance function in a uniform grid. Once this array has been created, computing the signed distance field of a given point can be done online really fast (in the order of microseconds), independent of the complexity of the obstacles.

Other planners differ from CHOMP in the following ways:

- Obstacle avoidance formulation. Instead of precomputing a distance field, TrajOpt uses online convex-convex collision checking using Gilbert-Johnson-Keerthi (GJK) and Expanding Polytope Algorithm (EPA). This allows it to model the robot geometry more accurately using meshes instead of the combination of spheres and

capsules as done in CHOMP. Furthermore, it provides better information (i.e., gradient) on how to take two colliding shapes out of the collision.

- Trajectory representation. In CHOMP, STOMP, and ITOMP, the trajectory needs to be represented by a large number of configurations to ensure that the collision checking against small obstacles still works. TrajOpt requires fewer number of states, due to its continuous collision cost that checks the collision of the swept volumes occupied by the links. GPMP, on the other hand, uses a continuous representation of trajectory via Gaussian Process that inherently ensure the smoothness through its priors.

- Solving strategy. While CHOMP uses covariant gradient descent to ensure smoothness, STOMP relies on stochastic sampling around the current solution to find the step direction, enabling it to optimize cost functions that are non-smooth and non-differentiable. TrajOpt uses Sequential Quadratic Programming that allows it to handle hard constraints easily. Finally, GPMP optimizes over the sparse set of states that define the trajectory while still computing the obstacle cost over a large number of intermediate points, thanks to the Gaussian Process representation.

The success of these optimization-based planners, however, does not mean that they completely replace the sampling-based planners. In fact, they still suffer from a common problem of optimization solver, i.e., getting stuck in poor local optima without good initialization. In motion planning, one important source of bad local optima is the presence of obstacle(s). In a tight environment, it can be very difficult for optimization-based planners to find even a feasible solution. Other constraints such as joint limit and orientation constraint can also cause the solvers to fail at finding a feasible solution, except with a good initialization close to good local optima. Unlike sampling-based planners, these optimization-based planners do not possess a completeness guarantee, and the performance is highly dependent on the initial solution. That is why planners such as TrajOpt suggest the user initialize using different waypoints when it is difficult to obtain a solution.

Even in sampling-based planners, optimization has become a part of its toolbox. As mentioned above, constraints cannot be incorporated naturally into sampling-based planners, but it does not mean that it is impossible. In [34], an extension of a sampling-based planner to handle constraints has been proposed. The main idea is to add a projection step such that all configurations that are added to the graph/tree are first projected to the constraints manifold. These projections steps are performed through gradient-based optimization which is costly (they can take $> 90\%$ of the total planning time [35]). The optimization formulation of this projection step is similar to the one for IK as described in Section 2.1.1, but ignoring the EE reaching cost.

2.1.3 Optimal Control

One of the most popular robot controllers is *computed torque controller*, also known as the *inverse dynamics controller*. Compared to a basic PID controller, a computed torque controller allows us to use a lower gain, as it compensates the robot dynamics using the known dynamics model, effectively performing feedback linearization. This can result in a more compliant behavior, especially important in torque-controlled robots such as Franka Emika or Kuka Light Weight Arm [36]. However, a computed torque controller does not provide us with a convenient way to handle constraints such as joint limit, torque limit, or velocity limit. An extension of inverse dynamics cast the control problem as Quadratic Programming (QP), where objective functions are quadratic and the constraints are formulated as linear constraints. The resulting QP problem can be solved in less than 1ms, allowing it to be part of a control loop (1kHz). It is often referred to as TSID [6].

TSID is an instantaneous controller, i.e., it only considers one time step at a time. While it results in fast computation, it limits the behavior that can be planned and executed. Some complex behavior that requires future anticipation such as angular momentum regulation or momentum building to climb high slopes is difficult to achieve with such a controller [37, 38]. Recently, optimal control is gaining popularity as the alternative formulation to generate and control more complex behavior, especially in legged robots. In legged robots research, optimal control has been used to plan the center-of-mass motion using the simplified Linear Inverted Pendulum model [39, 40]. As research progresses, the model considered is getting more and more complex, moving to a full centroidal model [41, 42, 43] and even recently the whole-body dynamics [44, 45, 46]. It allows more agile behaviors and multi-contact locomotion, at the cost of more computational powers.

Optimal control considers a time horizon and uses the system model to anticipate future events. It cast the control problem as an optimization problem:

$$C(\boldsymbol{x}, \boldsymbol{u}) = \sum_{t=0}^{T-1} c_t(\boldsymbol{x}_t, \boldsymbol{u}_t) + c_T(\boldsymbol{x}_T, \boldsymbol{u}_T), \tag{2.5}$$

subject to the dynamics

$$\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t), \tag{2.6}$$

where $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$ are the state and control command at time t. Optimal Control Problem (OCP) may also have equality and inequality constraints. The objective of solving an OCP is to find the optimal state and control trajectories $(\boldsymbol{x}^*, \boldsymbol{u}^*)$ that minimizes the cost function while respecting the dynamics. In robotics, given the system's high degrees of freedom and complexity, most researchers rely on numerical optimization to solve the problem, often called the direct methods [47].

One of the popular methods for solving an OCP in robotics is iLQR [7] due to its fast computation. With efficient implementation, it is fast enough to control manipulator [48] and even humanoid robot [45] in online MPC fashion. Furthermore, besides finding the optimal control sequence for a given problem, iLQR also provides us with feedback gains that can be used to keep the robot within the planned trajectory, increasing its stability. In the next section, we first look at Linear Quadratic Regulator (LQR) and then discuss iLQR as the extension of Linear Quadratic Regulator (LQR).

Time-Varying Finite Horizon Linear Quadratic Regulator (LQR)

A time-varying finite horizon LQR problem is a subclass of OCP with time-varying linear dynamics

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t,$$

and quadratic costs

$$C(\boldsymbol{x}, \boldsymbol{u}) = \sum_{t=0}^{T-1} (\boldsymbol{x}_t^\top \boldsymbol{Q}_t \boldsymbol{x}_t + \boldsymbol{u}_t^\top \boldsymbol{R}_t \boldsymbol{u}_t) + \boldsymbol{x}_T^\top \boldsymbol{Q}_T \boldsymbol{x}_T,$$

where $\boldsymbol{A}_t$ and $\boldsymbol{B}_t$ are the system matrices, $\boldsymbol{Q}_t$ and $\boldsymbol{R}_t$ are precision matrices for state and control cost. For such class of problems, the solution can be obtained analytically. We focus here on the batch least-squares solution of LQR. Each $\boldsymbol{x}_t$ can be written in terms of $\boldsymbol{x}_0$,

$$\boldsymbol{x}_1 = \boldsymbol{A}_0 \boldsymbol{x}_0 + \boldsymbol{B}_0 \boldsymbol{u}_0,$$

$$\boldsymbol{x}_2 = \boldsymbol{A}_1 \boldsymbol{x}_1 + \boldsymbol{B}_1 \boldsymbol{u}_1 = \boldsymbol{A}_1 \boldsymbol{A}_0 \boldsymbol{x}_0 + \boldsymbol{A}_1 \boldsymbol{B}_0 \boldsymbol{u}_0 + \boldsymbol{B}_1 \boldsymbol{u}_1,$$

and so on until $\boldsymbol{x}_T$. We can then stack all $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$ and get the batch equation

$$\boldsymbol{x} = \boldsymbol{S}_x \boldsymbol{x}_0 + \boldsymbol{S}_u \boldsymbol{u}, \tag{2.7}$$

where $\boldsymbol{x} = (\boldsymbol{x}_0^\top, \boldsymbol{x}_1^\top, \cdots, \boldsymbol{x}_T^\top)^\top$, $\boldsymbol{u} = (\boldsymbol{u}_0^\top, \boldsymbol{u}_1^\top, \cdots, \boldsymbol{u}_{T-1}^\top)^\top$, and

$$\boldsymbol{S_x} = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{A_0} \\ \boldsymbol{A}_1\boldsymbol{A}_0 \\ \vdots \\ \prod_{t=0}^{T-1} \boldsymbol{A}_{T-t} \end{bmatrix}, \boldsymbol{S_u} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{B}_0 & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{A}_1\boldsymbol{B}_0 & \boldsymbol{B}_1 & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \prod_{t=1}^{T-1} \boldsymbol{A}_{T-t}\boldsymbol{B}_0 & \cdots & \cdots & \boldsymbol{B}_{T-1} \end{bmatrix}.$$

We can then write the cost function in batch form as

$$C(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{x}^\top \boldsymbol{Q}_s \boldsymbol{x} + \boldsymbol{u}^\top \boldsymbol{R}_s \boldsymbol{u}, \tag{2.8}$$

where $\boldsymbol{Q}_s = \text{blockdiag}(\boldsymbol{Q}_0, \boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_T)$ and $\boldsymbol{R}_s = \text{blockdiag}(\boldsymbol{R}_0, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_{T-1})$ are

block diagonal matrices. Substituting (2.7) to (2.8), we obtain

$$\begin{aligned} C(\boldsymbol{x}, \boldsymbol{u}) =& \boldsymbol{x}^\top \boldsymbol{Q}_s \boldsymbol{x} + \boldsymbol{u}^\top \boldsymbol{R}_s \boldsymbol{u} \\ =& (\boldsymbol{S}_x \boldsymbol{x}_0 + \boldsymbol{S}_u \boldsymbol{u})^\top \boldsymbol{Q}_s (\boldsymbol{S}_x \boldsymbol{x}_0 + \boldsymbol{S}_u \boldsymbol{u}) + \boldsymbol{u}^\top \boldsymbol{R}_s \boldsymbol{u} \\ =& \boldsymbol{u}^\top (\boldsymbol{S}_u^\top \boldsymbol{Q}_s \boldsymbol{S}_u + \boldsymbol{R}_s) \boldsymbol{u} + 2 \boldsymbol{u}^\top \boldsymbol{S}_u^\top \boldsymbol{Q}_s \boldsymbol{S}_x \boldsymbol{x}_0 + \boldsymbol{x}_0^\top \boldsymbol{S}_x^\top \boldsymbol{Q}_s \boldsymbol{S}_x \boldsymbol{x}_0. \end{aligned}$$

Note that the cost is quadratic in $\boldsymbol{u}$, and it is a standard least-square problem. By computing the gradients and set it to zero, we obtain the solution, i.e.

$$\boldsymbol{u} = -(\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q}_s \boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R}_s)^{-1} \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q}_s \boldsymbol{S}_{\boldsymbol{x}} \boldsymbol{x}_0. \tag{2.9}$$

The optimal state trajectory can then be calculated from (2.7).

Iterative Linear Quadratic Regulator (iLQR)

iLQR can be used to solve more general problems than LQR involving non-quadratic cost functions and nonlinear dynamics. Starting from an initial guess $(\boldsymbol{x}_0, \boldsymbol{u}_0)$, iLQR iteratively refines this guess by making a simpler approximation of the OCP at each step. Let us consider the current guess $(\boldsymbol{x}^k, \boldsymbol{u}^k)$, where k is the iteration index. Given the general cost function in (2.5), we can approximate it as a quadratic function around $(\boldsymbol{x}^k, \boldsymbol{u}^k)$,

$$c_t(\delta \boldsymbol{x}_t, \delta \boldsymbol{u}_t) = \frac{1}{2} \begin{bmatrix} \delta \boldsymbol{x}_t \\ \delta \boldsymbol{u}_t \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{c}_{\boldsymbol{x}\boldsymbol{x},t} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{c}_{\boldsymbol{u}\boldsymbol{u},t} \end{bmatrix} \begin{bmatrix} \delta \boldsymbol{x}_t \\ \delta \boldsymbol{u}_t \end{bmatrix} + \begin{bmatrix} \boldsymbol{c}_{\boldsymbol{x},t} & \boldsymbol{c}_{\boldsymbol{u},t} \end{bmatrix} \begin{bmatrix} \delta \boldsymbol{x}_t \\ \delta \boldsymbol{u}_t \end{bmatrix}, \tag{2.10}$$

where $\boldsymbol{c}_{\boldsymbol{x},t}, \boldsymbol{c}_{\boldsymbol{u},t}, \boldsymbol{c}_{\boldsymbol{x}\boldsymbol{x},t}$, and $\boldsymbol{c}_{\boldsymbol{u}\boldsymbol{u},t}$ are the cost function's first and second order derivatives with respect to $\boldsymbol{x}$ and $\boldsymbol{u}$. We omit here the cross-derivatives $\boldsymbol{c}_{\boldsymbol{x}\boldsymbol{u},t}$ for simplifying the derivations and the notations, but a similar derivation works when $\boldsymbol{c}_{\boldsymbol{x}\boldsymbol{u},t}$ is not zero.

Similarly, we can approximate the dynamics in (2.6) using the linear approximation

$$\delta \boldsymbol{x}_t = \boldsymbol{A}_t \delta \boldsymbol{x}_t + \boldsymbol{B}_t \delta \boldsymbol{u}_t, \tag{2.11}$$

where $\boldsymbol{A}_t$ and $\boldsymbol{B}_t$ are the derivatives of the dynamics $\boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ with respect to $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$, respectively. The derivatives are evaluated at the current guess $(\boldsymbol{x}^k, \boldsymbol{u}^k)$. If the dynamics is approximated as quadratic instead of linear, it is referred to as Differential Dynamic Programming (DDP) [49].

At this stage, we have quadratic costs and linear dynamics as functions of $\delta \boldsymbol{x}$ and $\delta \boldsymbol{u}$. This is therefore a time-varying LQR problem, of which the variables of interest are $\delta \boldsymbol{x}$ and $\delta \boldsymbol{u}$. We can solve this either by the batch least-squares solution or dynamic programming, and obtain the optimal $\delta \boldsymbol{x}^*$ and $\delta \boldsymbol{u}^*$. Although in standard LQR the

two methods give the same outputs, in iLQR they will be different due to the dynamics rollout step, i.e. the forward pass. When solving the LQR subproblem using dynamic programming, at each time step we calculate the resulting $\boldsymbol{u}_t$, and the forward pass is calculated using the actual nonlinear dynamics. In the batch least-squares solution, on the other hand, the forward pass is calculated using the approximated linear dynamics, so the two will give slightly different solutions.

The $\delta\boldsymbol{u}$ calculated by solving the LQR subproblem is a directional step to improve the current guess $(\boldsymbol{x}^k, \boldsymbol{u}^k)$. Typically, a line search is performed to find the optimum step length to move in this direction (see [44]). With the given optimum step length α we calculate the new $\boldsymbol{u}$ as $\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \alpha\delta\boldsymbol{u}^k$. By performing dynamics rollout using this new $\boldsymbol{u}^{k+1}$ we obtain the new state trajectory $\boldsymbol{x}^{k+1}$. The line search guarantees that $(\boldsymbol{x}^{k+1}, \boldsymbol{u}^{k+1})$ has lower cost than the previous guess. We can then make another approximation around the new guess to obtain a new LQR problem and improve the solution. This is iterated until convergence. Besides obtaining the optimal solution $(\boldsymbol{x}^*, \boldsymbol{u}^*)$, we also obtain the time-dependent feedback gain $\boldsymbol{K}_t$ to be used for feedback control in the proximity of $(\boldsymbol{x}^*, \boldsymbol{u}^*)$. At each time step, we obtain the following control law,

$$\boldsymbol{u}_t = \boldsymbol{u}_t^* + \boldsymbol{K}_t(\boldsymbol{x}_t - \boldsymbol{x}_t^*).$$

More details on iLQR can be found in [50, 37, 7].

Extension of iLQR

The standard iLQR formulation does not handle constraints except in the form of soft constraints, i.e., formulating the constraints as cost functions and tuning the weights such that the constraints are satisfied at the convergence. Several extensions to handle hard constraints have been proposed, e.g., using a squashing function [51], QP [52], or a combination of augmented Lagrangian and projection method [53].

iLQR can be classified as a single shooting method since it solves the problem by finding the optimal sequence of control commands [47](the state trajectory is obtained by integrating the dynamics with the optimal control sequence). Initializing such a method is challenging, as small errors in the control sequence (especially at the beginning) can result in a large change in the state trajectory. On the other hand, a multiple shooting method treats both the state and control trajectories as optimization variables, while the dynamics is used as constraints. While we can only initialize the single shooting method with the control sequence, we can use both the state and control trajectories to initialize multiple shooting methods. However, the standard way to solve a multiple shooting problem using non-linear optimization solvers such as interior point method is usually too slow for robotics applications. In [44], an extension of iLQR called Feasibility-prone Differential Dynamic Programming (FDDP) has been proposed. FDDP formulation is

similar to multiple shooting as it optimizes both state and control trajectories jointly while maintaining the computation speed of iLQR. Hence, FDDP accepts initialization by (infeasible) state and control trajectories.

2.2 Learning Methods

How can we use machine learning to predict initial guesses of the optimization problems listed above? In this thesis, we consider two lines of approaches: *supervised learning* and *probability density estimation*.

In supervised learning, the initial guess prediction is treated as a regression problem to learn the mapping $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$, where $\boldsymbol{y}$ is the initial guess and $\boldsymbol{x}$ is the task decription. In Section 2.2.1, we discuss several supervised learning techniques that are considered in this thesis.

The second approach treats the problem quite differently. The cost function is transformed into an unnormalized PDF, and we use probability density estimation techniques to approximate this PDF. We can then sample from this model to obtain samples from the high probability area, i.e., samples that have low cost. In Section 2.2.2 we discuss several probability density estimation techniques that are considered in this thesis.

2.2.1 Supervised Learning

k-Nearest Neighbor (k-NN)

k-NN is a very simple non-parametric method. Given a task $\boldsymbol{x}^*$, the algorithm finds K samples $\{\boldsymbol{x}_k, \boldsymbol{y}_k\}_{k=1}^K$ in the database $\{\boldsymbol{X}, \boldsymbol{Y}\}$ where $\{\boldsymbol{x}_k\}_{k=1}^K$ are the K-nearest to $\boldsymbol{x}^*$ according to a chosen metric (in this chapter the Euclidean metric is used). It then predicts the corresponding robot path $\boldsymbol{y}^*$ by taking the average $\boldsymbol{y}^* = \frac{1}{K}\sum_{k=1}^K \boldsymbol{y}_k$. The method is very simple to implement and it works well if there is a sufficiently dense dataset, but it suffers from the curse of dimensionality; as the dimension of $\boldsymbol{x}$ increases, the number of data that needs to be stored increases exponentially.

Gaussian Process Regressor (GPR)

Like k-NN, GPR [17] is a non-parametric method which improves its accuracy as the number of data increases. While having a higher computational complexity compared to k-NN, GPR tends to interpolate better, resulting in higher approximation accuracy. Given the database $\{\boldsymbol{X}, \boldsymbol{Y}\}$, GPR assigns a Gaussian prior to the joint probability of $\boldsymbol{Y}$, i.e., $p(\boldsymbol{Y}|\boldsymbol{X}) = \mathcal{N}\Big(\boldsymbol{\mu}(\boldsymbol{X}), \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X})\Big)$. $\boldsymbol{\mu}(\boldsymbol{X})$ is the mean function and $\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X})$ is the covariance matrix constructed with elements $\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, where $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the

kernel function that measures the similarity between the inputs $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. In this thesis we use Radial Basis Function (RBF) as the kernel function, and the mean function $\boldsymbol{\mu}(\boldsymbol{X})$ is set to be zero as usually done in GPR.

To predict the output $\boldsymbol{y}^*$ given a new input $\boldsymbol{x}^*$, GPR constructs the joint probability distribution of the training data and the prediction, and then conditions on the training data to obtain the predictive distribution of the output, $p(\boldsymbol{y}^* |\ \boldsymbol{x}^*) \sim \mathcal{N}(\boldsymbol{m}, \boldsymbol{\Sigma})$, where $\boldsymbol{m}$ is the posterior mean computed as

$$\boldsymbol{m} = \boldsymbol{K}(\boldsymbol{x}^*, \boldsymbol{X})\, \boldsymbol{K}^{-1}(\boldsymbol{X}, \boldsymbol{X})\, \boldsymbol{Y}(\boldsymbol{X}), \tag{2.12}$$

and $\boldsymbol{\Sigma}$ is the posterior covariance which provides a measure of uncertainty on the output. In this thesis we simply use the posterior mean $\boldsymbol{m}$ as the output, i.e., $\boldsymbol{y}^* = \boldsymbol{m}$.

While having good approximation accuracy, one major limitation with GPR is that it does not scale well with very large datasets. There are variants of GPR that attempt to overcome this problem, e.g., sparse GPR [54] or using Stochastic Variational Inference (SVI) [55]. More details on GPR can be found in [17] and [1].

Bayesian Gaussian Mixture Regression (BGMR)

When using RBF as the covariance function, GPR assumes that the mapping from $\boldsymbol{x}$ to $\boldsymbol{y}$ is smooth and continuous. When this assumption is met, it performs very well, but it will yield poor results otherwise. For example, when there is a discontinuity in the mapping or there are multimodal outputs, GPR tends to average the solutions from both sides of the discontinuity or both modes. This characteristic is also shared by many other function approximators. To handle discontinuity and multimodality problems, using local models is one of the possible solutions. Each local model can be fit to each side of the discontinuity or to each mode.

Gaussian Mixture Regression (GMR) is an example of such local model approaches [56]. It can be seen as a probabilistic mixture of linear regressions. Given the database $\{\boldsymbol{X}, \boldsymbol{Y}\}$ it can be used to construct the joint probability of $(\boldsymbol{x}, \boldsymbol{y})$ as a mixture of Gaussians

$$p(\boldsymbol{x}, \boldsymbol{y}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{2.13}$$

where π_k, $\boldsymbol{\mu}_k$, and $\boldsymbol{\Sigma}_k$ are the k-th component's mixing coefficient, mean, and covariance, respectively. Given a query $\boldsymbol{x}^*$, the conditional probability of the output $\boldsymbol{y}^*$ is also a mixture of Gaussians. Let $\boldsymbol{\theta} = \{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^{K}$, denoting the GMR parameters to be determined from the data.

We can decompose $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ according to $\boldsymbol{x}$ and $\boldsymbol{y}$ as

$$\boldsymbol{\mu}_k = \begin{pmatrix} \boldsymbol{\mu}_{k,x} \\ \boldsymbol{\mu}_{k,y} \end{pmatrix} \quad \text{and} \quad \boldsymbol{\Sigma}_k = \begin{pmatrix} \boldsymbol{\Sigma}_{k,xx} & \boldsymbol{\Sigma}_{k,xy} \\ \boldsymbol{\Sigma}_{k,yx} & \boldsymbol{\Sigma}_{k,yy} \end{pmatrix}. \tag{2.14}$$

Given a query $\boldsymbol{x}^*$, the predictive distribution of $\boldsymbol{y}$ can then computed by conditioning on $\boldsymbol{x}^*$,

$$p(\boldsymbol{y}|\ \boldsymbol{x}^*, \boldsymbol{\theta}) = \sum_{k=1}^{K} p(k|\ \boldsymbol{x}^*, \boldsymbol{\theta})\, p(\boldsymbol{y}|\ k, \boldsymbol{x}^*, \boldsymbol{\theta}), \tag{2.15}$$

where $p(k|\ \boldsymbol{x}^*, \boldsymbol{\theta})$ is the probability of $\boldsymbol{x}^*$ belonging to the k-th component,

$$p(k|\ \boldsymbol{x}^*, \boldsymbol{\theta}) = \frac{\pi_k\, \mathcal{N}(\boldsymbol{x}^*|\ \boldsymbol{\mu}_{k,x}, \boldsymbol{\Sigma}_{k,xx})}{\sum_{i=1}^{K} \pi_i\, \mathcal{N}(\boldsymbol{x}^*|\ \boldsymbol{\mu}_{i,x}, \boldsymbol{\Sigma}_{i,xx})}, \tag{2.16}$$

and $p(\boldsymbol{y}|\ k, \boldsymbol{x}^*, \boldsymbol{\theta})$ is the predictive distribution of $\boldsymbol{y}$ according to the k-th component,

$$p(\boldsymbol{y}|\ k, \boldsymbol{x}^*, \boldsymbol{\theta}) = \mathcal{N}\Big(\boldsymbol{\mu}_{k,y} + \boldsymbol{\Sigma}_{k,yx}\boldsymbol{\Sigma}_{k,xx}^{-1}(\boldsymbol{x}^* - \boldsymbol{\mu}_{k,x}), \boldsymbol{\Sigma}_{k,yy} - \boldsymbol{\Sigma}_{k,yx}\boldsymbol{\Sigma}_{k,xx}^{-1}\boldsymbol{\Sigma}_{k,xy}\Big), \tag{2.17}$$

which is a Gaussian distribution with the mean being linear in $\boldsymbol{x}^*$. The resulting predictive distribution (2.15) is then a mixture of Gaussians. The point prediction $\boldsymbol{y}^*$ can be obtained from this distribution by applying moment matching to the distribution in (2.15) to approximate it by a single Gaussian, and use the mean of the Gaussian as the desired output $\boldsymbol{y}^*$, see [57, 58] for details.

In GMR, the parameters π_k, $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ are determined from the data by Expectation Maximization method, while the number of Gaussians K is usually determined by the user. Bayesian GMR (BGMR) [59] is a Bayesian extension of GMR that allows us to estimate the posterior distribution of the mixture parameters (instead of relying on a single point estimate as in GMR). The number of components K can also be automatically determined from the data. As a Bayesian model, BGMR gives priors to the parameters π_k, $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$, and computes the posterior distribution of those parameters given the data. In high dimensional problems, the prior reduces the overfitting that commonly occurs with GMR. The prediction $\boldsymbol{y}^*$, given the input $\boldsymbol{x}^*$, is then computed by marginalizing over the posterior distribution and conditioning on $\boldsymbol{x}^*$. The resulting predictive distribution of $\boldsymbol{y}$ is a mixture of t-distributions,

$$p(\boldsymbol{y}|\boldsymbol{x}^*, \boldsymbol{X}, \boldsymbol{Y}) = \sum_{k=1}^{K} p(k|\boldsymbol{x}^*, \boldsymbol{X}, \boldsymbol{Y})\, p(\boldsymbol{y}|k, \boldsymbol{x}^*, \boldsymbol{X}, \boldsymbol{Y}), \tag{2.18}$$

where $p(k|\boldsymbol{x}^*, \boldsymbol{X}, \boldsymbol{Y})$ is the probability of $\boldsymbol{x}^*$ belonging to the k-th component of the mixture, and $p(\boldsymbol{y}^*|k, \boldsymbol{x}^*, \boldsymbol{X}, \boldsymbol{Y})$ is a multivariate t-distribution, the mean of which is linear in $\boldsymbol{x}^*$. We can interpret (2.18) as K probabilistic linear regression models, each of which has the probability of $p(k|\boldsymbol{x}^*, \boldsymbol{X}, \boldsymbol{Y})$. More details about BGMR can be found

in [59].

To obtain a point-prediction $\boldsymbol{y}^*$ from (2.18), there are several approaches. One of the most used is to take the mean of the predictive distribution in (2.18) using moment matching. While this approach can provide smooth estimates (as required in many applications), the same problems as in GPR will appear in the case of discontinuity and multimodality; taking the average in those cases will give us poor results. Instead, as the point prediction, we use the mean[2] of the component in (2.18) having the highest probability, which approximately corresponds to the mode of the multimodal distribution. Alternatively, we can also use the mean of each t-distributions as separate predictions, which gives us several possible solutions. In some cases (e.g., when we would like to retrieve all possible solutions) this approach can be very useful.

Artificial Neural Networks

Neural networks are probably the most popular learning methods nowadays, not the least due to their ease of use, especially with the availability of libraries such as tensorflow [23] and pytorch [22] that makes modeling and training neural networks very easy. While convolutional layers are widely used with applications involving images, in many robotics applications MLP (i.e., neural networks consisting of fully-connected layers with nonlinear activation units) is often sufficient, even with as few as two to three hidden layers [60].

Due to its ease of use, it is easy to treat a neural network as a black box learning method for performing regression. Such treatment, however, often fails when the problem at hand defies the implicit assumptions of the standard MLP model. For example, when learning a function with multimodal output, a standard MLP will average the multiple modalities, resulting in a poor prediction (the same with a standard GPR). A neural network model called Mixture Density Networks (MDN) [61] handles this issue by predicting a mixture of Gaussian distributions instead of a single prediction. Each Gaussian can correspond to a different mode if trained properly. The resulting conditional probability distribution helps to model multimodal functions more precisely.

2.2.2 Probability Density Estimation

In probabilistic machine learning, it is customary to view a cost function probabilistically by transforming it to an unnormalized probability distribution, e.g., via the following equation [62]:

$$\tilde{p}(\boldsymbol{x}) = \exp(-c(\boldsymbol{x})), \tag{2.19}$$

where $c(\boldsymbol{x}), \tilde{p}(\boldsymbol{x})$ are the cost function and the unnormalized probability density function, respectively. The two are connected as follows: the point $\boldsymbol{x}$ that has low cost is viewed

[2] As in Gaussian distribution, the mean of a multivariate t-distribution is also its mode.

as a random variable with high probability, and the point $\boldsymbol{x}$ with high cost as a random variable with low probability. We can then view the problem of minimizing the cost function as finding the random variable $\boldsymbol{x}$ that maximizes the probability $p(\boldsymbol{x})$ (i.e., finding the mode of the distribution). This transformation allows us to use various techniques from the probability density estimation community. For example, instead of minimizing the cost function, we can fit a parametric probability density function such as GMM and then perform sampling. When the density estimation is good, the samples will come from around the region with the low cost surrounding the (multiple) optima. This results in a richer set of solutions compared to the optimization approach.

To understand (2.19), we first look at a simple example, i.e., a quadratic cost function, which can be related to a Gaussian distribution. We then discuss more general ways to solve the density estimation problem, i.e., using Variational Inference, GAN, and Tensor Train (TT).

Quadratic Cost and Product of Gaussians

A quadratic cost can be viewed probabilistically as corresponding to a Gaussian distribution. Given the quadratic cost

$$c(\boldsymbol{x}) = (\boldsymbol{x} - \bar{\boldsymbol{x}})^\top \boldsymbol{W} (\boldsymbol{x} - \bar{\boldsymbol{x}}), \tag{2.20}$$

the optimal solution $\boldsymbol{x}^* = \bar{\boldsymbol{x}}$ does not contain much information about the cost function itself. Instead, we can view $\boldsymbol{x}$ as a random variable with a Gaussian probability, i.e.,

$$p(\boldsymbol{x}) = \mathcal{N}(\bar{\boldsymbol{x}}, \boldsymbol{W}^{-1}) = \frac{1}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \bar{\boldsymbol{x}})^\top \boldsymbol{W} (\boldsymbol{x} - \bar{\boldsymbol{x}})\right), \tag{2.21}$$

where $\bar{\boldsymbol{x}}$ and $\boldsymbol{W}^{-1}$ are the mean and the covariance of the Gaussian, respectively. The negative log-likelihood of this Gaussian distribution is equivalent to (2.20) up to a constant factor. Note that if we transform (2.20) by the exponential transformation (2.19), we obtain (2.21) except for the constant factor. According to $p(\boldsymbol{x})$, $\boldsymbol{x}$ has the highest probability at $\bar{\boldsymbol{x}}$, and $\boldsymbol{W}^{-1}$ gives the directional information on how this probability changes as we move away from $\bar{\boldsymbol{x}}$. The point having the lowest cost in (2.20) is therefore associated with the point having the highest probability in (2.21).

Similarly, an objective function composed of several quadratic terms

$$\hat{\boldsymbol{\mu}} = \arg\min_{\boldsymbol{x}} \sum_{k=1}^{K} (\boldsymbol{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{W}_k (\boldsymbol{x} - \boldsymbol{\mu}_k) \tag{2.22}$$

can be seen as a product of Gaussians $\prod_{k=1}^{K} \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{W}_k^{-1})$, with centers $\boldsymbol{\mu}_k$ and covariance

matrices $\boldsymbol{W}_k^{-1}$. The Gaussian $\mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{W}}^{-1})$ resulting from this product has parameters

$$\hat{\boldsymbol{\mu}} = \left(\sum_{k=1}^{K} \boldsymbol{W}_k\right)^{-1} \left(\sum_{k=1}^{K} \boldsymbol{W}_k \boldsymbol{\mu}_k\right), \quad \hat{\boldsymbol{W}} = \sum_{k=1}^{K} \boldsymbol{W}_k.$$

$\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{W}}$ are the same as the solution of (2.22) and its Hessian, respectively. Viewing the quadratic cost probabilistically allows us to capture more information about the cost function in the form of the covariance matrix $\hat{\boldsymbol{W}}^{-1}$.

Variational Inference

For a general cost function, we cannot obtain its corresponding probability density function analytically as for the quadratic function above. One popular method to approximate a probability density function is Variational Inference (VI) [63]. It recasts the approximation problem as an optimization. VI approximates the *target density* $p(\boldsymbol{x})$ with a *tractable density* $q(\boldsymbol{x}; \boldsymbol{\lambda})$, where $\boldsymbol{\lambda}$ are the *variational parameters*. Tractable density means that drawing samples from $q(\boldsymbol{x}; \boldsymbol{\lambda})$ should be easy and $q(\boldsymbol{x}; \boldsymbol{\lambda})$ should be properly normalized.

$q(\boldsymbol{x}; \boldsymbol{\lambda})$ is a parameterized distribution (e.g., GMM [64] or neural network [65]), and to approximate the target density, we minimize the Kullback-Leibler (KL) divergence between the two distributions. There are two different KL divergence measures, i.e., the forward KL,

$$D_{\mathrm{KL}}(p||q) = \int_{\boldsymbol{x}} p(\boldsymbol{x}; \boldsymbol{\lambda}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x}; \boldsymbol{\lambda})} d\boldsymbol{x},$$

and the reverse KL,

$$D_{\mathrm{KL}}(q||p) = \int_{\boldsymbol{x}} q(\boldsymbol{x}; \boldsymbol{\lambda}) \log \frac{q(\boldsymbol{x}; \boldsymbol{\lambda})}{p(\boldsymbol{x})} d\boldsymbol{x}.$$

Forward KL can be seen as the expectation of the difference between the two distributions under $p(\boldsymbol{x})$, i.e., $D_{\mathrm{KL}}(p||q) = \mathbb{E}_p[\log p(\boldsymbol{x} - \log q(\boldsymbol{x})]$. For general distributions, analytical expression of $D_{\mathrm{KL}}(p||q)$ is not tractable, and its evaluation requires samples taken from the distribution $p(\boldsymbol{x})$. However, we often do not have access to $p(\boldsymbol{x})$, only to the unnormalized density function $\tilde{p}(\boldsymbol{x})$ where $p(\boldsymbol{x}) = \frac{\tilde{p}(\boldsymbol{x})}{Z}$ and Z is the normalization constant which is difficult to find. On the other hand, Reverse KL only requires us to sample from the tractable density $q(\boldsymbol{x})$ from which we can sample easily by design.

Furthermore, Forward KL is often said to be *zero avoiding*, i.e., it ensures that $q(\boldsymbol{x})$ covers the region where $p(\boldsymbol{x}) \neq 0$, while reverse KL is *zero forcing*, i.e., it may set $q(\boldsymbol{x}) = 0$ even in area where $p(\boldsymbol{x}) \neq 0$ to minimize its cost function. This is especially important when $p(\boldsymbol{x})$ is multimodal distribution, as shown in Fig. 2.1. Here, $p(\boldsymbol{x})$ is a mixture of Gaussians (shown in blue), while $q(\boldsymbol{x})$ is a Gaussian distribution whose parameters (i.e.,

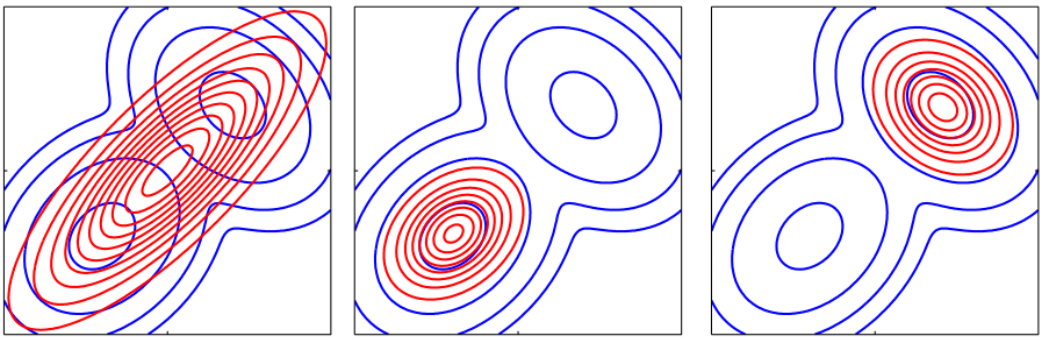


Figure 2.1 – The difference between Forward KL (left) and Reverse KL(middle and right). The blue and red contours refer to the target and the tractacble density function, respectively. The figure is taken from [1].

mean and covariance) need to be optimized. Optimizing with Forward KL results in the left figure where the Gaussian tries to cover both modes (its mean will be between the two target means), while optimizing with Reverse KL result in a smaller Gaussian that fits only one of the modes (middle or right, depending on the initialization).

In practice, both measures can be used depending on the applications. Approximation using reverse KL is easier since the sampling is straightforward, but it may miss some of the modes. While sampling in forward KL is more difficult, there are ways to overcome it, e.g., using importance sampling [66], but a good proposal distribution would be required.

Generative Adversarial Networks (GAN)

GAN [67] is an unsupervised learning model to learn the probability distribution associated to a dataset. It consists of two main components: a generator $G(\boldsymbol{z};\boldsymbol{\theta_G})$ and a discriminator $D(\boldsymbol{q};\boldsymbol{\theta_D})$. The generator is trained to transform the input noise $\{\boldsymbol{z}\}$ drawn from $p_z(\boldsymbol{z})$ (typically a unit Gaussian) into samples $\{\boldsymbol{q}\}$ that look similar to the data distribution. To do this, a discriminator is trained in parallel to output the probability $p(\boldsymbol{q})$ that tells whether $\boldsymbol{q}$ comes from the dataset or the generator. The training of GAN is therefore like a game between the generator and the discriminator where one tries to beat the other. The generator and discriminator are neural networks with parameters $\boldsymbol{\theta_G}$ and $\boldsymbol{\theta_D}$.

Note that GAN requires a dataset that contains samples from the target distribution. When only the unnormalized density function $\tilde{p}(\boldsymbol{x})$ is available, we need to first generate samples from that distribution, which is often challenging. For robotics problems, one simple method is to start from uniform samples and optimize them according to the cost function, so that the optimized samples are located in the region with high probability. Sampling techniques such as Monte Carlo sampling can also be used, but it does not scale well to high dimensional problems that are common in robotics.

Tensor Train (TT) Decomposition

A tensor is a multidimensional array. The number of modes (or dimensions) of the multidimensional array is commonly called the order of the tensor. For example, a vector can be considered as a first-order tensor and a matrix as a second-order tensor. We can use a tensor to approximate a probability function by first discretizing the function and storing the discrete value in the tensor. For example, a 2D function can be stored in a 2D array. However, this naive approach does not scale up to higher dimensions, as the number of entries in the tensor will increase exponentially with the number of dimensions. Tensor Train decomposition [68] offers a nice solution by decomposing the tensor into a set of third-order tensors called *cores* [3]. This impacts both the construction and the storage of the tensor; we do not need to evaluate every single entry to construct the tensor, and we only store the tensor cores instead of the whole tensor. TT decomposition does this by exploiting the correlation between the variables of the function, which results in low-rank tensors. Once we obtain the TT model for approximating the probability density function, we can easily generate samples from the model.

Tensor methods have not been widely used in robotics. In Chapter 7, we will see that they provide a lot of functionalities that are suitable for many optimization problems in robotics, including handling multimodal problems naturally and obtaining global optima in challenging optimization problems.

2.3 Memory of Motion

The idea of building a memory of motion that learns from the previous experience has previously been explored in the context of optimal control and motion planning. In [9] a trajectory library is constructed to learn a control policy. A set of trajectories are planned offline using A^* algorithm and stored as a library, then k-NN is used online to determine the action to perform at each state. In [69], they use a similar approach to predict an initial guess for balancing a two-link robot, which is then optimized by Differential Dynamic Programming. An iterative method to build a memory of motion to initialize an optimal control solver is proposed in [70]. They use neural networks to approximate the mapping from the task descriptors (initial and goal states) to the state and control trajectories. Another neural network is trained to approximate the value function, which is then used as a metric to determine how close two states are dynamically. In [71] GPR is used to predict new trajectories based on the library of demonstrated movements encoded as Dynamic Movement Primitives (DMP) [72]. GPR is used to map the task descriptors to the DMP parameters.

In sensor-based control, Pastor et al. [73, 74] use the previously recorded sensor data to

[3] Tensor decomposition techniques can be viewed as higher-order extensions of Singular Value Decomposition (SVD) in matrices.

construct a predictive model for subsequent task executions. The robot trajectories are encoded as movement primitives and augmented with the sensor data. In [75] they extend the work to also predict when to switch the behavior from one movement primitive to another using the reference sensor signal.

In sampling-based motion planning, Probabilistic Roadmap (PRM) [15] can be seen as the construction of a memory of motion by precomputing the graph connecting robot configurations. In [76], it is extended to adapt to dynamic environments by first building a graph that only considers an obstacle-free workspace and encoding the mapping from the workspace cells to the nodes and edges of the graph. Given the obstacles, the graph is modified accordingly by removing the nodes and the edges that collide with the obstacles. Instead of using the encoding map, an optimized collision detection circuit is constructed to perform parallel collision checks for the edges in [77]. In [78], an elastic roadmap is proposed to capture the connectivity of the workspace. Unlike PRM, the elastic roadmap contains a set of paths in the task space belonging to different homotopy classes. Given a specific goal, the roadmap provides global guidance to the controller such that it can achieve the task without getting stuck at local minima, similar to the role of the memory of motion in this thesis. It extends the work on elastic strip [79]. Unlike the elastic strip that considers only a single homotopy class, an elastic roadmap is able to re-plan globally and hence adapts better to changing environments with the help of on-board sensor [80].

Some works exploit Rapidly-exploring Random Trees (RRT) [14], another popular sampling-based method. For example, in [81] an offline computation is used to speed up the online planning in the form of an additional bias in sampling the configurations. In [82], an *Experience Graph* is built from previously planned solutions. During the online planning, the search is biased towards this graph. The *Lightning* framework is proposed in [83] to plan paths in high-dimensional spaces by learning from experience. The path library is constructed incrementally. Given the current path library and a task to be executed, the algorithm runs two versions of the planner online, one that plans from scratch and the other one initialized by the library.

Memory of motion has also been used for initializing optimization-based motion planner. In [84], a high-dimensional task descriptor is constructed, and the metric between the task descriptors is refined to minimize the necessary refinement of the initial trajectory using L_1 norm, resulting in a sparse metric and hence sparse descriptors. In [10], a sub-indexing is used to reduce the amount of memory storage and to use the sub-trajectories of the original solutions. In robot locomotion [85], a mapping from the task space to the optimal trajectory for cyclic walking is learned using various machine learning algorithms, but the prediction is not re-optimized online. In [86], the initial trajectories for real-time catching are predicted using k-NN, Support Vector Regression, and GPR.

Despite the existing work of using a memory of motion, the following aspects are still missing, especially in the context of warm starting a trajectory optimization solver:

- None of the previous methods explicitly attempt to handle multimodal problems.
- They do not address the challenge of building the dataset as mentioned in Section 1.2.
- The existing techniques only provide a single prediction for a given problem.

2.4 Conclusion

In this chapter, we have first discussed various optimization problems in robotics that are considered in this thesis. They share some common features: all of them are nonconvex optimization problems that can get stuck in local optima, depending on the initialization. We then discuss various learning techniques that can be used to provide good initializations to the optimization solvers. Finally, we mention existing works that attempt to produce good initialization, i.e., warm start, for some optimization problems, and discuss briefly their limitations. In the upcoming chapters, we propose novel ways to use these learning methods to speed up the various optimization problems while overcoming the limitations of existing approaches.

Supervised Learning Approaches Part I

3 Memory of Motion for Warm Starting Trajectory Optimization

Trajectory optimization for motion planning requires good initial guesses to obtain good performance. In this chapter, we build a memory of motion based on a database of precomputed robot paths to provide good initial guesses. The memory of motion relies on function approximators and dimensionality reduction techniques to learn the mapping between the tasks and the robot paths. Three function approximators are compared: k-Nearest Neighbor, Gaussian Process Regression, and Bayesian Gaussian Mixture Regression. In addition, we show that the memory can also be used as a metric to choose between several possible goals. Finally, using an ensemble method to combine different function approximators results in a significantly improved warm-starting performance. We demonstrate the proposed approach with motion planning examples on the PR2 dual-arm robot and the Atlas humanoid robot.

This chapter is a result of the joint work with A. Paolillo and E. Pignat. A. Paolillo helped in formulating the experiments and E. Pignat helped in formulating and providing the codes of the Bayesian GMR.

Publication Note

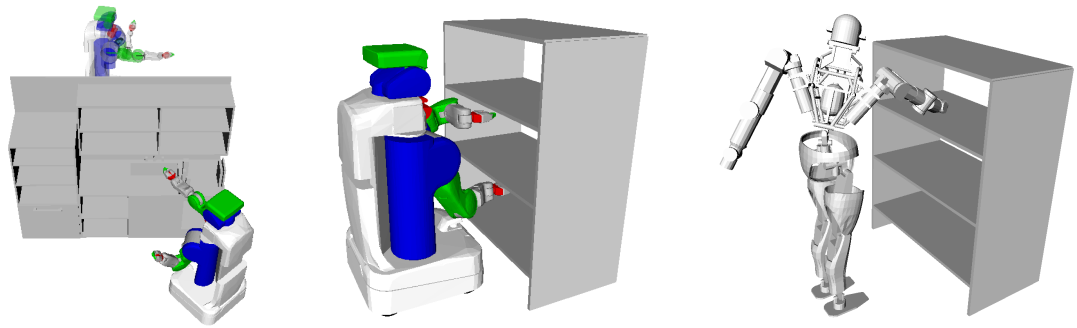
The material presented in this chapter is adapted from the following publication:

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

Supplementary Material and Source Codes

Video related to this chapter is available at:
https://youtu.be/b49xwN9mon0

Source codes related to this chapter are available at:
https://github.com/teguhSL/memmo_for_trajopt_codes



(a) (b) (c)

Figure 3.1 – Examples of motion planning problems: (a) moving the PR2 base to the goal while avoiding an obstacle, (b) dual arm motion of PR2 to pick items from a shelf to another, and (c) whole-body motion of Atlas.

3.1 Introduction

Motion planning for high-dimensional robots presents many challenges, especially in the presence of constraints such as obstacle avoidance, joint limits, etc. To handle the high-dimensionality and the various constraints, many approaches [4, 31, 5] focus on *trajectory optimization* methods that attempt to find a locally optimal solution. Trajectory optimization formulates the motion planning problem as an optimization problem (see Section 2.1.2 for more details) that is non-convex in general, which makes finding the global optimum difficult. Trajectory optimization methods such as TrajOpt [4], CHOMP [31], or STOMP [5] solve the non-convex problem by iteratively optimizing around the current solution. While such approach is very popular and yields good practical results, the convergence and the quality of the solution are very sensitive to the choice of the initial guess. If it is far from the optimal solution, the method can get stuck at poor local optima.

To overcome this problem, our approach builds a *memory of motion* that learns how to provide good initializations (i.e., a *warm start*) to the solver based on previously solved problems. Functionally, the memory of motion is expected to learn the mapping $\boldsymbol{f} : \boldsymbol{x} \rightarrow \boldsymbol{y}$ that maps each task $\boldsymbol{x}$ to the robot path $\boldsymbol{y}$. Such mapping can be highly nonlinear and *multimodal* (i.e., one task $\boldsymbol{x}$ can be associated to several robot paths $\boldsymbol{y}$), and the dimension of $\boldsymbol{y}$ is typically very high. Our proposed method relies on machine learning techniques such as function approximation and dimensionality reduction to learn this mapping effectively. We use the term *memory of motion* to include both the database of motions and the algorithms to query the warm starts from the database.

We point out that while other techniques such as sampling-based motion planners can also be used to warm start the solver (e.g. in [10]), such methods typically require a considerable computation time (i.e. in the order of seconds) that is comparable to the

solver's convergence time itself, given the very high dimensional problems considered here. In contrast, querying the memory of motion can be done very fast, in the order of milliseconds. Additionally, our proposed method produces initial guesses that are close to the optimal solutions, reducing the convergence time.

The contribution of this chapter is the following. First, we propose the use of function approximation methods to learn the mapping $\boldsymbol{f}(\boldsymbol{x})$. We consider three methods: k-Nearest Neighbors (k-NN), Gaussian Process Regression (GPR), and Bayesian Gaussian Mixture Regression (BGMR), and discuss their different characteristics on various planning problems. We show in particular that BGMR handles multimodal output very well. Furthermore, we show that the memory of motion can be also be used as a *metric* for choosing optimally between several possible goals. Finally, we demonstrate that using an ensemble of function approximators to provide warm starts boosts the success rate significantly.

The chapter is organized as follows. Section 3.2 explains the methods for constructing and using the memory of motion. The experimental results are presented and discussed in Section 3.3 and 3.4. Finally, Section 3.5 concludes the chapter.

3.2 Method

Section 3.2.1 discusses the main idea of building the memory of motion using function approximation and dimensionality reduction techniques to learn the mapping between the task and the associated robot path. Section 3.2.2 then explains how the memory of motion can be used as a metric for choosing between different goals. Finally, Section 3.2.3 describes how the warm starting performance can be improved significantly using an ensemble method.

3.2.1 Building a Memory of Motion

To learn the mapping $\boldsymbol{f} : \boldsymbol{x} \rightarrow \boldsymbol{y}$, we firstly generate a set of tasks $\{\boldsymbol{x}\}$ and the corresponding robot paths $\{\boldsymbol{y}\}$. This is done by sampling $\boldsymbol{x}$ from a uniform distribution covering the space of possible tasks and run the trajectory optimizer to obtain the robot paths $\boldsymbol{y}$ until we obtain N samples ($\boldsymbol{x}$,$\boldsymbol{y}$). Let $\boldsymbol{X} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{N-1})$ and $\boldsymbol{Y} = (\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{N-1})$. The mapping $\boldsymbol{f}$ can be learned by training function approximators using the database $\{\boldsymbol{X}, \boldsymbol{Y}\}$. In this chapter we consider three function approximators: k-NN, GPR, and BGMR (see Section 2.2.1 for the description of each method).

Algorithm 1 Building a Memory of Motion

INPUT: number of samples N
OUTPUT: the database $\{\boldsymbol{X}, \boldsymbol{Y}\}$ and the function approximator $\boldsymbol{f}$

$\boldsymbol{X} = [\,]$, $\boldsymbol{Y} = [\,]$
for $i = 1, 2, \ldots, N$ **do**
  sample a random task $\boldsymbol{x}_i$
  compute the initial guess $\tilde{\boldsymbol{y}}_i$ to achieve $\boldsymbol{x}_i$ by straight-line motion
  solve $\boldsymbol{x}_i$ using TrajOpt warm started by $\tilde{\boldsymbol{y}}_i$, to obtain the path $\boldsymbol{y}_i$
  if $\boldsymbol{y}_i$ is valid **then**
    add $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ to $\{\boldsymbol{X}, \boldsymbol{Y}\}$
  end if
end for
apply PCA to $\boldsymbol{Y}$ to obtain $\hat{\boldsymbol{Y}}$ (*Optional*)
train the function approximator $\boldsymbol{f}$ on $\{\boldsymbol{X}, \boldsymbol{Y}\}$ (*or on* $\{\boldsymbol{X}, \hat{\boldsymbol{Y}}\}$ *if PCA is used*)

Algorithm 2 Using the Memory as a Metric

INPUT: A list of goals $\{\boldsymbol{x}_j\}_{j=1}^{M}$, a function approximator $\boldsymbol{f}$
OUTPUT: The optimal goal $\boldsymbol{x}^*$ and the corresponding path $\boldsymbol{y}^*$

for $j = 1, 2, \ldots, M$ **do**
  compute the initial guess $\tilde{\boldsymbol{y}}_j = \boldsymbol{f}(\boldsymbol{x}_j)$
  compute the cost $\ell(\tilde{\boldsymbol{y}}_j)$
end for
$j^* \leftarrow \arg\min_j \ell(\tilde{\boldsymbol{y}}_j)$
$\boldsymbol{x}^* \leftarrow \boldsymbol{x}_{j^*}$
$\tilde{\boldsymbol{y}}^* \leftarrow \tilde{\boldsymbol{y}}_{j^*}$
solve $\boldsymbol{x}^*$ using TrajOpt warm started by $\tilde{\boldsymbol{y}}^*$, to obtain the path $\boldsymbol{y}^*$

Dimensionality reduction

In our problem, the path $\boldsymbol{y} \in \mathbb{R}^{DT}$ is a vector consisting of the sequence of configurations with dimension D during T time steps, which can be very high. This motivates us to use dimensionality reduction techniques to reduce the dimension of $\boldsymbol{y}$. For example, when T is large and the time interval is small, RBF can be used to represent the evolution of each variable as weights of the basis functions. Techniques such as Principal Component Analysis (PCA), Independent Component Analysis, Factor Analysis, and Variational Autoencoder [1, 87] can also be used. The mapping to be learned then becomes the mapping from $\boldsymbol{x}$ to $\hat{\boldsymbol{y}}$, where $\hat{\boldsymbol{y}}$ is the projection of $\boldsymbol{y}$ to the lower dimensional subspace. The advantage is that the memory required to store the data is reduced significantly, while the approximation performance is maintained or even improved because the important correlations between the variables are preserved. In this work, since the number of time steps is not large, we use PCA to reduce the dimension of $\boldsymbol{y}$.

Algorithm 3 Ensemble Method

INPUT: Task $\boldsymbol{x}^*$, a list of function approximators $\{\boldsymbol{f}_j\}_{j=1}^{M}$
OUTPUT: The path $\boldsymbol{y}^*$ that accomplishes the task $\boldsymbol{x}^*$

for all $j = 1, 2, \ldots, M$ **do in parallel**
  compute the initial guess $\tilde{\boldsymbol{y}}_j = \boldsymbol{f}_j(\boldsymbol{x}^*)$
  solve $\boldsymbol{x}^*$ using TrajOpt warm started by $\tilde{\boldsymbol{y}}_j$, to obtain the path $\boldsymbol{y}_j$
  if $\boldsymbol{y}_j$ is valid **then**
    $\boldsymbol{y}^* = \boldsymbol{y}_j$
    Terminate the parallel execution
  end if
end for

3.2.2 Using the Memory as a Metric

In some planning problems, there can be several alternative goals to be achieved. For example, in robot drilling task [88], the orientation around the drilling axis is free (the number of possible goals is infinite). A naive way is to choose one of the goals randomly, plan the motion, and if it fails then select another goal. While this is simple to implement, it reduces the solution space significantly and may result in failure even when a solution exists, especially if the chosen goal turns out to be difficult or even impossible to reach. Another method is to plan the paths to each goal and select the one having the smallest cost, but this is computationally expensive. Therefore, it will be useful to have a *metric*, i.e., a distance function, that measures the cost to a given goal. Our idea is to use the memory of motion as the metric.

In Section 3.2.1, function approximators were trained to predict an initial guess to achieve a given task $\boldsymbol{x}$. The possible goals can then be formulated as multiple tasks $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{M-1}\}$. For each task $\boldsymbol{x}_i$, the function approximator predicts the initial guess $\tilde{\boldsymbol{y}}_i$ corresponding to the task, and the corresponding cost $\ell(\tilde{\boldsymbol{y}}_i)$ can be computed. The initial guess $\tilde{\boldsymbol{y}}_i^*$ and the corresponding task $\boldsymbol{x}_i^*$ with the lowest cost is then taken as the chosen goal to be given to the trajectory optimizer. Since the cost computation can be done quickly relative to optimization time, this approach can yield significant improvements to the trajectory optimizer performance.

3.2.3 Using Ensemble Method to Provide Warm Start

In machine learning, methods such as AdaBoost [89] and Random Forests [90] have shown that using an ensemble of methods often yields improved performance compared to choosing a single method. We propose to use an ensemble method where we run multiple trajectory optimizations in parallel, each one warm started by one of the function approximators in Section 3.2.1, and once one of them finds a successful path the others are terminated. Since each function approximator has different learning characteristics, combining them in this way can significantly improve the motion planning performance.

Table 3.1 – Base motion planning with one waypoint

Method	Success (%)	Conv. time (s)	Cost (rad/s)
STD	80.0	0.55±0.29	1.37±0.37
k-NN	93.0	0.35±0.20	1.45±0.47
GPR	96.0	0.37±0.15	**1.32±0.36**
BGMR	**97.0**	**0.32±0.14**	1.34±0.35

The method in Section 3.2.2 can also be used as one of the ensemble's component.

3.3 Experiments

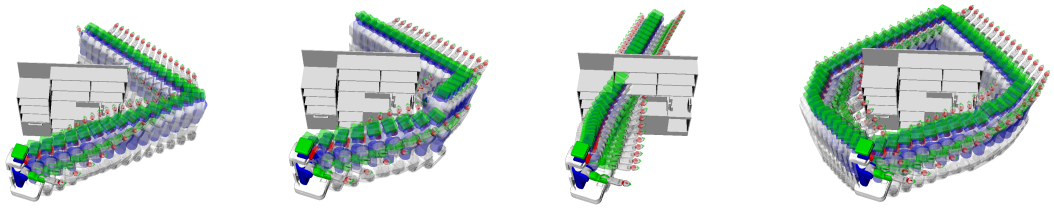
To evaluate the proposed method, we consider several examples of motion planning for PR2 and Atlas robots. TrajOpt [4] is used as the trajectory optimizer to be warm started. The output is the robot path that accomplishes the given task. In this chapter we only work with robot path as the output, but the method can also be applied to robot trajectory.

We consider 5 motion planning cases presented in ascending order of complexity. Each case is chosen to demonstrate certain characteristics of the proposed method. For each case, we follow the following procedures. First we generate the dataset by randomly sampling N_{train} tasks from a uniform distribution in the whole task space and run TrajOpt to find the paths achieving the tasks. The number of time steps T is set to 30, except for Atlas ($T = 15$). In all cases, the cost is defined as the discrete velocity of the states. The number of N_{train} is different for each case, depending on the complexity of the task. The function approximators are then trained with or without PCA using the dataset. We heuristically set 50 components for the PCA; for the k-NN, we use $K = 1$.

To validate the performance, we sample N_{test} random tasks and use the various methods to warm start TrajOpt. The solutions are compared in terms of *convergence time*, *success rate* and *cost*. The planning is considered successful if the solution is feasible. The comparison results are presented in the Tables 3.1-3.6. The values are averaged over N_{test} tasks, and the standard deviation is also given for the convergence time and the cost. In the presented results, we use the label 'STD' to refer to the solution obtained by warm starting the solver with a straight-line path (via waypoint, if any), and the names of the function approximators for the rest. The subscript 'PCA' is added when PCA is used. The query time for predicting the warm starts by each method is negligible w.r.t. the convergence time, i.e. less than 5 ms for most methods, except for BGMR without PCA (around 20ms), so they are not included in the comparison.

Table 3.2 – Base motion planning with two waypoints

Method	Success (%)	Conv. time (s)	Cost (rad/s)
STD	79.0	0.53±0.23	1.43±0.37
k-NN	**95.0**	0.32±0.16	1.53±0.62
GPR	0.0	-	-
BGMR	94.0	**0.31±0.15**	**1.33±0.40**



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

Figure 3.2 – Motion planning for the PR2 mobile base. Warm start produced by (a) straight-line with waypoint, (b) k-NN, (c) GPR and (d) BGMR.

3.3.1 Base motion planning

The task is to plan the motion for the PR2 mobile base from a random pose in front of the kitchen to another random pose behind the kitchen (Fig. 3.1a). In this case, the state $\boldsymbol{q}$ is the 3-DoF planar pose of the base. The task descriptor is then $\boldsymbol{x} = (\boldsymbol{q}_{\text{init}}^\top, \boldsymbol{q}_{\text{goal}}^\top)^\top$. The database is constructed with $N_{\text{train}} = 200$ samples and the evaluation is performed with $N_{\text{test}} = 100$. Although this is an easy problem, TrajOpt actually finds it difficult to solve when given a poor initialization. For example, initializing TrajOpt with a straight-line interpolation from $\boldsymbol{q}_{\text{init}}$ to $\boldsymbol{q}_{\text{goal}}$ never manages to find a feasible solution because it results in a path that moves the robot through the kitchen while colliding, and the solver get stuck in poor local optima due to the conflicting gradients. To obtain better initialization for building the database, we initialize TrajOpt with two manually chosen waypoints on the left and on the right of the kitchen ($\boldsymbol{q}_{\text{left}}$ and $\boldsymbol{q}_{\text{right}}$, respectively).

We consider two cases of building the database: in the first one, we only use $\boldsymbol{q}_{\text{right}}$ as waypoint, while in the second we use both $\boldsymbol{q}_{\text{left}}$ and $\boldsymbol{q}_{\text{right}}$. We initialize TrajOpt with the straight-line motion from $\boldsymbol{q}_{\text{init}}$ to the waypoint and from the waypoint to $\boldsymbol{q}_{\text{goal}}$. With this setting we build the database, train the function approximators, and obtain the results as shown in Table 3.1 and 3.2.

In the first case, the mapping from $\boldsymbol{x}$ to $\boldsymbol{y}$ is unimodal because all movements go through the right. Table 3.1 shows that the performance of k-NN, GPR and BGMR are quite similar. In the second case, however, the output is multimodal because the database

Table 3.3 – Planning from fixed $\boldsymbol{q}_{\text{init}}$ to random $\boldsymbol{q}_{\text{goal}}$.

Method	Success (%)	Conv. time (s)	Cost (rad/s)
STD	80.0	0.77±0.37	1.83±0.61
k-NN	91.2	**0.58±0.29**	1.93±0.69
GPR	92.4	0.65±0.25	1.84±0.57
GPR_{PCA}	**92.8**	0.66±0.26	**1.83±0.57**
BGMR	88.8	0.64±0.26	1.85±0.56
BGMR_{PCA}	92.0	0.67±0.26	1.84±0.58

Table 3.4 – Planning from random $\boldsymbol{q}_{\text{init}}$ to $\boldsymbol{q}_{\text{goal}}$.

Method	Success (%)	Conv. time (s)	Cost (rad/s)
STD	75.2	0.82±0.43	**1.31±0.74**
k-NN	65.6	1.16±0.58	1.55±0.88
GPR	85.6	0.85±0.39	1.32±0.74
GPR_{PCA}	88.0	**0.81±0.36**	1.33±0.73
BGMR	84.0	0.81±0.40	1.34±0.76
BGMR_{PCA}	78.3	0.88±0.42	1.39±0.78
Waypoints	94.0	1.52±0.67	1.83±1.34
Ensemble	**97.2**	1.06±0.41	1.42±0.82

contains two possible ways (modes) to accomplish the same task. This affects GPR significantly (see Table 3.2), as GPR averages both modes and outputs a path that goes through the kitchen, while k-NN and BGMR are not affected. k-NN does not average the modes because we use $K = 1$, while BGMR overcomes the multimodality by constructing local models for each mode automatically.

Fig. 3.2 shows the examples of warm starts produced by each method in the second case. As expected, GPR provides a warm start that goes through the kitchen (hence 0 success rate). With BGMR, if we retrieve the components with the two highest probability, both possible solutions are obtained.

3.3.2 Planning from a fixed initial configuration to a random goal configuration

Here $\boldsymbol{q}$ consists of 14 joint angles of the two 7 DoF arms of PR2. The task $\boldsymbol{x}$ is to move from a fixed $\boldsymbol{q}_{\text{init}}$ to a random goal configuration $\boldsymbol{q}_{\text{goal}}$ (i.e. $\boldsymbol{x} = \boldsymbol{q}_{\text{goal}}$). The database is constructed with $N_{\text{train}} = 500$, and the evaluation results with $N_{\text{test}} = 250$ are presented in Table 3.3.

Since each PR2 arm is redundant, the path from $\boldsymbol{q}_{\text{init}}$ to $\boldsymbol{q}_{\text{goal}}$ can be multimodal, which may pose a problem for GPR. However, Table 3.3 shows that GPR and BGMR perform similarly. This is due to the fact that although redundant robots can achieve a goal configuration in many different ways, planning using optimization here results in similar motions for similar goal configurations. The use of PCA does not improve the performance significantly, but it still helps to reduce the size of the data. In this case, for each path it reduces the number of variables from 30×14 ($D \times T$) to 50 (number of PCA components), more than 8 times reduction while maintaining the performance.

3.3.3 Planning from a random initial configuration to a random goal configuration

To proceed with a more complex case, the task here is to plan a path from a random initial configuration $\boldsymbol{q}_{\text{init}}$ to a random target $\boldsymbol{q}_{\text{goal}}$. The task $\boldsymbol{x}$ consists of the initial and goal configurations, $\boldsymbol{x} = (\boldsymbol{q}_{\text{init}}^\top, \boldsymbol{q}_{\text{goal}}^\top)^\top$. The database is constructed with $N_{\text{train}} = 500$ and evaluated with $N_{\text{test}} = 250$. The result is presented in Table 3.4.

k-NN performs poorly here, similarly to STD, due to the dimension of the input space $\boldsymbol{x}$ that is much larger compared to Section 3.3.2. To achieve good performance, k-NN requires a much denser dataset. GPR outperforms BGMR by a wide margin.

The last row of Table 3.4 shows the result of the ensemble method described in Section 3.2.3. Given an input $\boldsymbol{x}^*$, the method uses all function approximators to provide different warm starts, each of which is used to initialize an instance of TrajOpt in parallel. Once a valid solution is obtained, the other instances of TrajOpt are terminated. This method results in a huge boost of the success rate, with comparable convergence time and cost to the other methods. As comparison, we also include here the standard multiple initializations suggested by TrajOpt (labeled as ‘waypoints‘). Each initialization is created by interpolating through a waypoint that is manually defined. While the success rate is high, the convergence time and the cost increase significantly. On the contrary, each initialization in the ensemble method has a good probability of being close to the optimal solution, resulting in lower cost and convergence time. If having a lower cost is more important than the computational time, the parallel execution can also be programmed such that we wait until all instances of TrajOpt finish the optimization, then the cost of the resulting paths are compared and the one with the lowest cost is chosen. This alternative implementation can reduce the cost, but at the expense of higher computational time.

3.3.4 Planning to Cartesian goals from a fixed initial configuration

In Section 3.3.2 and 3.3.3, we use TrajOpt to plan to goals in configuration space. In practical situations, however, the task is often to reach a certain Cartesian pose using

Table 3.5 – Planning from fixed $\boldsymbol{q}_{\text{init}}$ to random Cartesian goal.

Method	Success (%)	Conv. time (s)	Cost (rad/s)
STD	65.2	1.10±0.62	1.86±0.86
k-NN	73.6	1.28±0.96	1.84±0.81
GPR	66.4	1.81±0.96	1.87±0.87
GPR_{PCA}	66.8	1.68±0.98	1.78±0.83
BGMR	74.4	1.37±0.82	1.82±0.86
BGMR_{PCA}	77.2	1.33±0.75	1.84±0.80
METRIC GPR_{PCA}	86.8	**0.70±0.30**	**1.49±0.56**
Ensemble	**98.0**	1.50±0.60	1.60±0.68

Table 3.6 – Planning the motion of Atlas from fixed $\boldsymbol{q}_{\text{init}}$ to random Cartesian goal.

Method	Success (%)	Conv. time (s)	Cost (rad/s)
STD	50.8	6.31±3.90	0.12±0.07
k-NN	58.8	1.48±1.39	0.11±0.06
GPR	54.4	**1.29±1.09**	**0.10±0.05**
GPR_{PCA}	60.0	1.54±1.46	0.11±0.05
BGMR	56.4	1.32±1.57	**0.10±0.05**
BGMR_{PCA}	58.0	1.36±1.16	0.11±0.06
Ensemble	**71.2**	1.46±1.40	0.12±0.06

the end effector (e.g., to pick an object on the shelf), instead of planning to a specific joint configuration. One way to solve this problem is to first compute a configuration that achieves the Cartesian pose using an inverse kinematic solver and plan to this configuration, but it does not make use of the flexibility inherent in the task. TrajOpt has an option to plan directly to a Cartesian goal, but it typically requires longer convergence time and lower success rate than planning to a joint configuration goal.

We present two approaches to use the memory of motion in this problem. In the first approach, we rely on the similar procedure as in previous cases: we formulate the task as $\boldsymbol{x} = (\boldsymbol{p}_{\text{left}}^\top, \boldsymbol{p}_{\text{right}}^\top)^\top$ where $\boldsymbol{p}_{\text{left}}$ and $\boldsymbol{p}_{\text{right}}$ are the Cartesian positions of the right and left hand of PR2. The database is then constructed with $N_{\text{train}} = 1000$ and the function approximators are trained. In this approach, TrajOpt plans to a Cartesian goal directly. The second approach relies on the fact that a Cartesian goal corresponds to multiple goals in configuration space. In Section 3.3.2 we have already constructed several function approximators that can predict an initial guess $\tilde{\boldsymbol{y}} = \tilde{\boldsymbol{q}}_{0:T}$, given a goal $\boldsymbol{q}_{\text{goal}}$ in configuration space. The second approach uses one of them as a metric (Section 3.2.2) to choose between the different goals in configuration space. First, given a Cartesian goal $\boldsymbol{x}$, we run an inverse kinematic solver to find $M = 5$ joint configurations that satisfy this pose. For each joint configuration, we use the function approximator to predict the initial guess of the robot path to reach that configuration, and we compute the cost of that

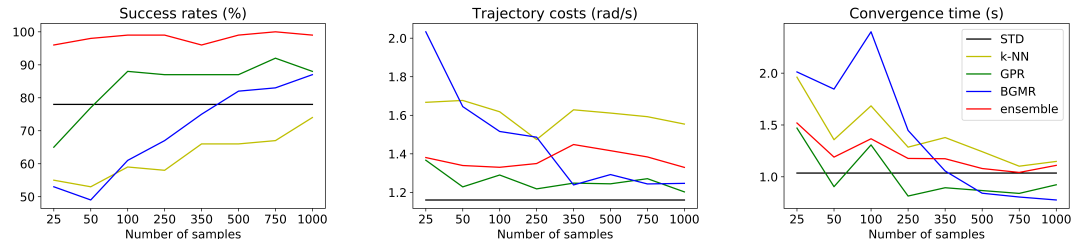
path. Finally, the path with the lowest cost and its corresponding goal configuration are chosen, and TrajOpt is run to reach this goal configuration with the given path as the warm start. Note that in this second approach, TrajOpt plans to a joint configuration instead of a Cartesian goal. For this approach we choose the method GPR_{PCA} from the Section 3.3.2, and use the term 'METRIC GPR_{PCA}' to differentiate from the first approach (denoted in standard notation).

We present the results in Table 3.5 with $N_{\text{test}} = 250$. Among the methods using the first approach, we note that BGMR yields better result than GPR because the mapping from the Cartesian goal $\boldsymbol{x}$ to the robot path $\boldsymbol{y}$ here is multimodal, as planning to a Cartesian pose has more redundancy compared to planning to a joint configuration. This again demonstrates that BGMR handles multimodal output better than GPR. However, the second approach METRIC GPR_{PCA} outperforms even BGMR. The improvement over the first approach is very significant in all three criteria. This demonstrates that using the memory as a metric to choose the optimal goal results in large improvements. We point out that the additional computational time required to find $M = 5$ IK solutions and the corresponding warm starts is only around 0.1 s, which is negligible compared to the convergence time. Finally, we use the ensemble method that uses all function approximators in parallel, including METRIC GPR_{PCA}. This boosts the success rate to 98%.

3.3.5 Planning whole-body motion for an Atlas robot

Finally, we also applied our method for planning the motion of the 34-DoF Atlas robot (28-DoF joints and 6-DoF root pose). We consider the same task as in Section 3.3.4, i.e. planning from a fixed initial configuration to a random Cartesian pose, in this case chosen to be the location of Atlas' right hand. The task $\boldsymbol{x} = (p_x, p_y, p_z)^\top$ corresponds to the target position of Atlas' right hand, while the orientation is not constrained. The feet location are fixed, while the Zero Moment Point (ZMP) is constrained to be between the two feet location. We use here the first approach as explained in Section 3.3.4, i.e. treating it as a regression problem where the input $\boldsymbol{x}$ is the Cartesian goal and the output $\boldsymbol{y}$ is the trajectory, and use the various function approximators to predict the initial guesses. The database is constructed with $N_{\text{train}} = 1000$ and the evaluation is performed with $N_{\text{test}} = 250$. The results are presented in Table 3.6.

k-NN performs quite well, as the input size of $\boldsymbol{x}$ is small (the position of the hand is constrained to be inside the shelf). Unlike in Section 3.3.4, the performance of GPR and BGMR are quite similar, although the goals are also in the Cartesian space. This is due to the difference in the implementation; in Section 3.3.4, given a Cartesian goal, we use an inverse kinematic solver to calculate the joint configuration that satisfies this goal, and calculate the initial guess as straight-line interpolation from the fixed initial configuration to the goal configuration. This initial guess is used when building the



(a) (b) (c)

Figure 3.3 – Performance comparison against the number of training samples N_{train}

database. Due to the redundancy of the PR2 dual arm, similar Cartesian goals can correspond to very different joint configurations, resulting in the multimodality of the solutions in the database. In this Atlas experiment, however, we do not provide initial guesses to TrajOpt when building the database, so TrajOpt always tries to solve the problem with zero initialization. This results in more uniform solutions, and hence GPR can still perform quite well. Finally, using the ensemble method again shows superior results, giving us an increase of the success rate by more than 10%.

Planning for such a high-dimensional problem with many constraints (feet location, ZMP constraint, kinematic constraint) requires quite a lot of computational time ($\sim$ 6.3 s in average without a warm start). Using the memory of motion in this complex task further exemplifies the benefit of the approach, as our method speeds up the computational time significantly by more than four times faster. We note that the tasks are sampled randomly, and there is no guarantee that the task is indeed feasible. This explains why even the best method (i.e. the ensemble method) only achieves $\sim$ 70% success rates.

3.4 Discussions

3.4.1 Choice of Function Approximators

In Section 3.3, we have compared the performance of k-NN, GPR and BGMR over different tasks, and shown that they have different characteristics. When the dataset is quite dense or the input space is small, k-NN usually manages to obtain good performance (as shown in Section 3.3.1, 3.3.2, and 3.3.4), while for larger input space (Section 3.3.4) it does not yield good results. GPR performs the best when the output is unimodal (Section 3.3.2 and 3.3.3), while for multimodal output BGMR has a better performance than GPR (Section 3.3.4). This comparison can guide us to select the best method for each task. However, it may not be obvious whether a given task (and its solution) is unimodal or multimodal (e.g. compare Section 3.3.4 and 3.3.5). A better way is to combine the different methods via an ensemble method, as we have shown in this chapter.

3.4.2 Data Requirement

In Fig. 3.3, we plot the performance of various methods against the number of training samples, with STD given as the baseline. We choose the task in Section 3.3.3, since it has the largest input space among the other tasks. It is interesting that when the training size is small, GPR performs quite well, while k-NN and BGMR are even worse than STD. As training size increases, k-NN and BGMR start to approach the performance of GPR. On the contrary, the performance of the ensemble method is quite stable even when the training size is small. As the training size grows, its convergence time decreases, while the success rate is already high even when the training size is small.

3.4.3 Ensemble Method

Using an ensemble method for motion planning has been explored in [91], which uses an ensemble of motion planners. While such approach also manages to boost the performance successfully, it is not easy to design and set up several motion planners for a given task. On the contrary, many function approximators are available and can be used easily, since our problem is formulated as a standard regression problem. We only need to configure one motion planner (in this work, TrajOpt, but other optimization frameworks can also be used) for a given task, unlike in [91]. Another benefit of our ensemble method is that each of the ensemble's component starts from an initial guess that has good probability of being close to the optimal solution. This reduces the average computational time, as we have shown by comparing it against the multiple waypoints initialization in Table 3.4.

3.4.4 Dynamic Environment

In this work we assume that the environment is static, so that the trajectories previously planned remain valid. When the environment changes, a new memory of motion has to be built. For the simple example in Section 3.3.1, building the memory takes only $\sim$3 minutes of computational time, but complex example such as Section 3.3.5 takes $\sim$3 hours. While paralellization can be used to speed up the building process, more effective strategies would be interesting to explore. In [92], an efficient way of updating a dynamic roadmap when the environment changes is presented. Such method can possibly be used to modify the existing memory of motion, so that we do not have re-build from scratch but only modify those affected. Alternatively, when the environment largely remain the same but a few obstacles are moving (as in many real tasks), we can include these obstacles' locations as additional inputs to the regression problem, at the expense of larger input size.

3.5 Conclusion

We have presented an approach to build a memory of motion to warm start trajectory optimization solver, and demonstrated through experiments with PR2 and Atlas robots that the warm start can improve the solver performance. Function approximators and dimensionality reduction were used to learn the mapping between the task descriptor and the corresponding robot path. Three function approximators were considered: k-NN as baseline, GPR, and BGMR, and their different characteristics have been discussed. The use of PCA also improved the solution, although not very significantly, while reducing the memory storage. We have also shown that we can use the memory of motion as a metric to choose optimally between several alternative goals, and this resulted in a significantly improved performance for the case of Cartesian goal planning. Finally, the different function approximators were combined as an ensemble method, which boosted the success rate significantly.

4 Learning How to Walk: Warm Starting an Optimal Control Solver

In Chapter 3 we have discussed the ensemble method for initializing an optimization-based motion planner. In this chapter, we propose to use a memory of motion for warm starting an optimal control solver for the locomotion task of a humanoid robot. We use HPP Loco3D, a versatile locomotion planner, to generate offline a set of dynamically consistent whole-body trajectories to be stored as the memory of motion. The learning problem is formulated as a regression problem to predict a single-step motion given the desired contact locations, which is used as a building block for producing multi-step motions. The predicted motion is then used as a warm start for the fast optimal control solver Crocoddyl. We have shown that the approach manages to reduce the required number of iterations to reach the convergence from ∼9.5 to only ∼3.0 iterations for the single-step motion and from ∼6.2 to ∼4.5 iterations for the multi-step motion, while maintaining the solution quality.

This chapter is a result of the joint work with C. Mastalli, P. Fernbach, and N. Mansard. The co-authors helped in generating the data and providing the codes for the optimal control solver.

Publication Note

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

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

Supplementary Material

Video related to this chapter is available at:
https://youtu.be/YqDZwbZXBGQ

4.1 Introduction

Legged locomotion is often achieved by first computing a sequence of contacts, generating a stable centroidal trajectory, and finally computing a whole-body motion through inverse dynamics [94, 95]. However, this approach cannot properly regulate angular momentum because (a) centroidal trajectory optimization [96, 97] does not consider the limbs momenta, and (b) whole-body control [6, 98] is instantaneous control action that theoretically cannot properly regulate the angular momentum since it represents a nonholonomic constraint on the multibody dynamics [99].

Optimal control is getting more attention in legged robots due to (a) its ability to properly control angular momentum [37], and (b) they can be deployed for real-time control [46, 100]. Recently, an efficient multi-contact optimal control framework called Crocoddyl [44] has been proposed. This framework relies on a novel multiple-shooting optimal control solver called FDDP. Crocoddyl can generate highly-dynamic maneuveurs for various legged robots such as iCub, Talos and ANYMal.

As a locally optimal solver, providing *warm starts* (i.e., good initial guesses) to Crocoddyl can improve its real-time performance in Model Predictive Control (MPC) setting significantly. In this chapter, we use the concept of memory of motion to generate the warm starts for Crocoddyl, to avoid poor local optima while speeding up the convergence. Using HPP Loco3D [101], a versatile locomotion framework for legged robots, we build offline a database of humanoid walking motions. We then train function approximators using the database to generate the warm starts for Crocoddyl.

The memory of motion concept has been used in other works, e.g. for bicopter and quadcopter [70], serial manipulators [69, 84, 83], and humanoid manipulation task [10]. However, none of them involves locomotion tasks except in [102], where a trajectory library is constructed for the LittleDog robot. Their work does not involve warm starting an optimization solver. Instead, the sequence of joint configurations retrieved from the library is used directly, with an integral controller to correct for errors. The library is created by using a joystick to move LittleDog across the terrain. Compared to [102], our approach of using HPP Loco3D to build the library, function approximations to learn the motion, and the optimal control solver Crocoddyl to optimize the motion is more versatile and applicable to higher DoF robots with more complex dynamics, such as the Talos humanoid robot [103] considered in this work.

The contribution in this chapter is as follows. Firstly, we propose a framework for learning a memory of motion to warm start a multi-contact optimal control solver (Fig. 4.1). We generate offline a database of dynamically consistent and collision-free motions using HPP Loco3D to build the memory of motion, which is then used to warm start the optimal control solver Crocoddyl online. We study the effect of having different solvers for the offline and online computations, and describe how we tackle this problem. Finally,

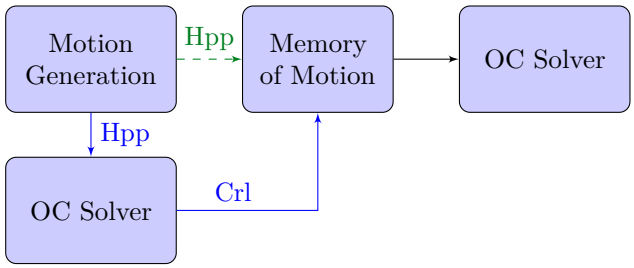


Figure 4.1 – Two approaches that are tested. In Approach 1 (dashed green), the motion generation using HPP Loco3D produces the dataset HPP, while in Approach 2 (solid blue), the dataset HPP is further optimized by the optimal control solver (Crocoddyl) to produce the dataset Crl. In each approach, the memory of motion is used for warm starting the optimal control solver.

we propose a method that learns single-step motions based on function approximations, which are then used to build multi-step motions.

The outline of the chapter is as follows. In Section 4.2, the overall framework and the learning method are presented. Section 4.3 presents the simulation results, both for the single-step and multi-step locomotions. Finally, Section 4.4 concludes the chapter.

4.2 Method

4.2.1 Problem Definition

We consider the problem of generating whole-body locomotion of a biped robot from one location to another in a known environment and a flat terrain. The desired output consists of the robot joint configuration trajectory $\boldsymbol{q} \in \mathbb{R}^{D_q \times T}$ and the control input trajectory $\boldsymbol{u} \in \mathbb{R}^{D_u \times T}$, where D_q, D_u, T are the joint configuration dimension, the control dimension, and the number of time steps. $\boldsymbol{q}$ includes both the trajectory of the root (i.e., the hip joint) and the joint positions, while $\boldsymbol{u}$ consists of the joint torques at each joint. The joint velocity and acceleration trajectories can also be included, but we only consider the joint configuration and the control trajectory here. To generate these trajectories, we use a fast optimal control solver Crocoddyl as the online solver. Our aim in this work is to provide good warm starts to Crocoddyl using a memory of motion such that the number of solver iterations to reach convergence can be reduced.

4.2.2 Overall Framework

In the proposed memory of motion approach, the choice of the offline and online solver is crucial. The offline solver is to build the database of motions, while the online solver is to control the robot in real time. The online solver has to be fast and efficient, hence

it is usually only locally optimal, such as Crocoddyl in this work. The offline solver, on the other hand, is used to generate the database offline, so the speed is less important. There are two alternatives: either the offline solver is the same as the online solver, or a different one. The advantage of choosing the same solver is that the motion in the database would then have the same characteristics (e.g., optimality) as the online solver, which is desirable. However, since the solver would only be locally optimal, it is difficult to use it for generating a good database without providing warm starts to the solver.

The second alternative uses a different solver for building the database, e.g., a more global planner that can solve various tasks without requiring warm starts. This is the approach that we take, as can be seen in Fig. 4.1. HPP Loco3D [104], a locomotion framework for multi-contact locomotion, is used as the offline solver. The framework is versatile and has been applied to both biped and quadruped robots. HPP Loco3D can compute the sequence of stable contacts to achieve the locomotion task, which cannot be done yet in Crocoddyl. We use HPP Loco3D to generate motion samples corresponding to various tasks and store them as the memory of motion.

However, the issue with this approach is that the motion in the database might not be considered *optimal* by Crocoddyl, because HPP Loco3D does not properly regulate angular momentum and uses heuristics for defining the swing-foot trajectories. Indeed, there are qualitatively different motion characteristics between HPP Loco3D and Crocoddyl. The former generates conservative and stable motions, while the latter uses the full dynamics that optimally reduces the joint torques and contact forces. Therefore, warm starting Crocoddyl using HPP Loco3D output will require additional iterations during the online computation to refine the motion according to its optimality criteria, which is undesirable.

We overcome this problem by leveraging both HPP Loco3D and Crocoddyl for building the memory. That is, we take the motion samples generated by HPP Loco3D and optimize them using Crocoddyl. The resulting output is then saved as the new memory of motion. By doing this, we combine the benefits of both frameworks: we can have a sequence of stable footholds (HPP Loco3D) with optimal whole-body motions (Crocoddyl). We will demonstrate that this will yield improvement in the quality of the warm starts. In this work, we refer to the database containing the HPP Loco3D motion samples as Database Hpp, and the one containing the optimized Crocoddyl motion samples as Database Crl.

In what follows, we describe in details HPP Loco3D and Crocoddyl frameworks.

HPP Loco3D

The planning framework proposed in HPP Loco-3D generates dynamically consistent and collision free multi-contact whole-body motion for legged robot. It takes as input the model of the environment and an initial and goal poses of the robot's root. Optionally,

some additional constraints may be specified, such as velocity bounds or a set of initial and final contact positions. This framework decouples the motion planning problem into several sub-problems to be solved sequentially. First, the guide planning produces a rough initial guess of the root path of the robot [101][105]. Then, a contact planner produces a feasible sequence of contacts following the root path [101][106]. After that, an optimal centroidal trajectory satisfying the centroidal dynamic constraints for the given contact points is computed [96]. Finally, a second order inverse dynamic solver[1] generates a whole-body motion, following the references of the contact locations and the centroidal trajectory.

Crocoddyl

It is a framework for multi-contact optimal control. Given a predefined sequence of contacts, it computes efficiently the state trajectory and control policy by using sparse analytical derivatives and by exploiting the problem structure inherited from the dynamic programming principle. Its optimal control algorithm, called Feasibility-prone Differential Dynamic Programming (FDDP), has a great globalization strategy and similar numerical behavior to multiple-shooting methods [44]. During the numerical optimization, FDDP computes the search direction and length through backward and forward passes, respectively. Unlike the classical Differential Dynamic Programming (DDP), the backward pass accepts infeasible state-control trajectories which is a critical aspect to warm starting the solver from the memory of motion; the forward pass simulates properly infeasible search direction, obtained in the backward pass, which improves the algorithm exploration.

4.2.3 Learning Strategies

One important question that needs to be considered is, what do we expect the memory to learn? Should it learn directly how to move from one location to another? while this is indeed possible, such method does not generalize very well, even in a fix environment. Instead, we decompose the data into single-step motions, and we retrieve a multi-step trajectory as a combination of single-step motions.

Learning Single-step Motion

Following our previous work [8] as described in Chapter 3, we formulate the problem of learning the single-step motion as a regression problem to approximate the mapping $f : \boldsymbol{x} \rightarrow \boldsymbol{y}$, where $\boldsymbol{x}$ is the task and $\boldsymbol{y}$ is the corresponding trajectory output. We separate the database into left-leg and right-leg movements, as this yields better results than combining both and let the memory learns how to choose the leg. Each motion

[1] https://github.com/stack-of-tasks/tsid

starts with the root position at the origin.[2] The task is defined as the initial and goal foot poses, $\boldsymbol{x} = [\boldsymbol{c}_{l0}, \boldsymbol{c}_{r0}, \boldsymbol{c}_{*T}] \in \mathbb{R}^9$, where $\boldsymbol{c} \in \mathbb{R}^3$ is the foot pose (2D position and orientation), the subscripts l, r correspond to the left and right foot, while $*$ is either l for the left-leg or r for the right-leg database. T is the final time step. $\boldsymbol{y}$ can be either the joint configuration trajectory $\boldsymbol{q} \in \mathbb{R}^{D_q \times T}$ or the control trajectory $\boldsymbol{u} \in \mathbb{R}^{D_u \times T}$. We then apply dimensionality reduction and function approximation techniques to solve the regression problem.

Since the dimension of $\boldsymbol{y} \in \mathbb{R}^{D \times T}$ is high, we need to find a smaller representation of $\boldsymbol{y}$. First, we represent the trajectory of each dimension of $\boldsymbol{y}$ as the weight of radial basis functions (RBFs), as commonly done in probabilistic movement primitives [30, 57]. Let $\boldsymbol{y}_i \in \mathbb{R}^T$ be the trajectory of the i_{th} dimension of $\boldsymbol{y}$. We can define the basis matrix $\boldsymbol{\Phi}$ as $[\boldsymbol{\phi}_0, \ldots, \boldsymbol{\phi}_{K-1}] \in \mathbb{R}^{T \times K}$, where $\boldsymbol{\phi}_k \in \mathbb{R}^T$ is the k_{th} basis function, defined as a Gaussian function centered at the k_{th} mean. The means are distributed equally along the time axis T, whereas the variance is chosen to be equal for all the basis functions and to have sufficient overlap. $\boldsymbol{y}_i$ can then be represented as the weights of these basis functions,

$$\boldsymbol{y}_i = \boldsymbol{\Phi} \boldsymbol{w}_i, \tag{4.1}$$

where $\boldsymbol{w}_i \in \mathbb{R}^K$ can be computed by standard linear least squares algorithm. This reduces the number of variables for each dimension from T, which is usually large, to K which can be much smaller. We can then stack the weights corresponding to all dimensions of $\boldsymbol{y}$ and obtain $\boldsymbol{w} = [\boldsymbol{w}_0, \ldots, \boldsymbol{w}_i, \ldots, \boldsymbol{w}_{D-1}] \in \mathbb{R}^{DK}$. Each $\boldsymbol{y}$ has the corresponding weight vector $\boldsymbol{w}$.

Now let's consider all the N samples in the database. We can apply principal component analysis (PCA) to further reduce the dimension of $\boldsymbol{w}$ over this database by keeping only M principal components to obtain $\hat{\boldsymbol{y}} \in \mathbb{R}^M$. We have thus reduced the dimension of $\boldsymbol{y}$ from DT to M. Inverse transformation from $\hat{\boldsymbol{y}}$ to $\boldsymbol{y}$ is a matrix multiplication that can be performed quickly. The regression problem then becomes $f : \boldsymbol{x} \rightarrow \hat{\boldsymbol{y}}$. To solve the regression problem, we consider two function approximation techniques: Gaussian Process Regression (GPR) and Gaussian Mixture Regression (GMR). More details about these two methods can be found in Section 2.2.1.

Constructing Multi-step Motion

To use the single-step motion for generating multi-step motions, we have to transform the coordinate system at each step. Let's first assume that we have the planned sequence of contacts (i.e., foot poses), $\{\boldsymbol{C}_i\}_{i=0}^{P-1}$, where $\boldsymbol{C}_i = (\boldsymbol{c}_{li}, \boldsymbol{c}_{ri})$ is the contacts at i_{th} footstep and P is the total number of footsteps. Assume, without loss of generality, that the motion starts at zero root position horizontally. The first step can be obtained by

[2] Without any loss of generality, as we can always transform the coordinate system

querying the single-step memory directly to move from $\boldsymbol{C}_0$ to $\boldsymbol{C}_1$, obtaining $\boldsymbol{y}^0$. To move from $\boldsymbol{C}_1$ to $\boldsymbol{C}_2$, we first need to shift the coordinate system such that the current root (based on the last configuration at $\boldsymbol{y}^0$) is in zero horizontal position. The motion from $\boldsymbol{C}_1$ to $\boldsymbol{C}_2$ can then be queried from the memory to obtain $\boldsymbol{y}^1$, and this is iterated until $\boldsymbol{C}_{P-1}$ to obtain $\{\boldsymbol{y}^i\}_{i=0}^{P-1}$. Finally, each trajectory $\boldsymbol{y}^i$ is transformed back to the original coordinate system and concatenated as a single trajectory $\boldsymbol{y}$.

The contact sequence $\{\boldsymbol{C}_i\}_{i=0}^{P-1}$ can be obtained from another planner, such as RB-PRM [101]. The alternative is to also learn it from the database. In this work, we assume that the contact sequence is already given.

4.3 Experiments

To evaluate the proposed approach, we conduct several experiments with the humanoid robot Talos in simulation. The robot joint configuration consists of 3-DoF root position, 3-DoF root orientation (described in quaternion), and 32 joint angles ($D_q = 39$), while the control input consists of 32 joint torques ($D_u = 32$). First, HPP Loco3D is used to generate $N = 1200$ samples of one-step motion (right-leg and left-leg movement in equal proportions), starting from the initial contact $(\boldsymbol{c}_{l0}, \boldsymbol{c}_{r0})$ to the goal contact $(\boldsymbol{c}_{lT}, \boldsymbol{c}_{rT})$. One sample thus consists of $\{(\boldsymbol{c}_{l0}, \boldsymbol{c}_{r0}), (\boldsymbol{c}_{lT}, \boldsymbol{c}_{rT}), \boldsymbol{q}, \boldsymbol{u}\}$. These are stored as Database Hpp.[3] Next, each sample is optimized using Crocoddyl, and the resulting samples are stored as Database Crl. The cost function in Crocoddyl consists of state and control regularization (around a standing pose and zero, respectively), and contact placement. Since we need high-quality database for the memory of motion, we use a small convergence threshold of 10^{-5} and the maximum number of iterations is set to be 50. The time interval in HPP Loco3D is 1 ms and 40 ms, respectively, so the HPP Loco3D data is subsampled to Crocoddyl's interval for the optimization. Both databases will be used and the performance will be compared. The databases are split into the training and the test dataset, with $N_{\text{train}} = 1000$ and $N_{\text{test}} = 200$. We applied RBF and PCA to reduce the dimensions of $\boldsymbol{q}$ and $\boldsymbol{u}$ with $K = 60$ and $M = 60$, as determined empirically.

We proceed as follows. First, we evaluate the accuracy performance of GPR and GMR in approximating the mapping f. The warm starts generated by GPR and GPR are then compared to the cold-start in terms of the Crocoddyl performance, i.e., the number of iterations until convergence and the resulting trajectory cost. Finally, we also compare the result of warm starts using only $\boldsymbol{q}$ to using both $\boldsymbol{q}$ and $\boldsymbol{u}$. For all comparisons, we use both the databases Hpp and Crl and compare their performance.

Figure 4.2 – Examples of predicting single-step motions. Warm start produced by *Top:* GPR, *Middle:* GMR, and *Bottom:* k-NN. *Left:* left foot movement. *Right:* right foot movement. The green, blue and red box correspond to the initial left, the initial right, and the goal contact pose, respectively.

4.3.1 Comparing GPR and GMR Accuracy

We train GPR and GMR on the reduced-dimension dataset $\{\boldsymbol{x}, \hat{\boldsymbol{y}}\}$ for both databases Hpp and Crl with N_{train} samples. The task $\boldsymbol{x}$ is defined as the initial and goal foot poses, $\boldsymbol{x} = [\boldsymbol{c}_{l0}, \boldsymbol{c}_{r0}, \boldsymbol{c}_{*T}]$ where $* = l$ for the left-leg movement and $* = r$ for the right-leg movement. The output $\boldsymbol{y}$ is defined here as the joint configuration trajectory $\boldsymbol{q}$, and $\hat{\boldsymbol{y}}$ is its smaller dimension representation. For each $\boldsymbol{x}$ in the N_{test} samples, we use GPR and GMR to predict the corresponding trajectory $\boldsymbol{y}$, and the accuracy is evaluted against the true trajectory in the database. The trajectory error (rad) is calculated as the difference between the true and predicted trajectory, i.e. $\frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} ||\boldsymbol{y}^i - \tilde{\boldsymbol{y}}^i||_2$, whereas the contact error (m) is defined as the difference between the foot poses of the true and predicted trajectory, $\frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} ||\boldsymbol{C}^i - \tilde{\boldsymbol{C}}^i||_2$.

The results can be seen in Table 4.1 (averaged over the left and right leg movements). We compare against k-NN with $k = 1$ as a baseline, to demonstrate that the proposed algorithm indeed generalizes well and the good performance is not due to having a very dense database. GPR overall has the lowest errors in both criteria and both databases. GMR has higher errors than GPR but still outperforms the baseline k-NN by a large margin. Some example of the motions can be found in Fig. 4.2. We can see that motion

[3]The database is available at https://github.com/MeMory-of-MOtion/docker-loco3d.

Table 4.1 – Comparing The Accuracy of GPR and GMR

	Database Hpp		Database Crl	
Method	**Traj. Err.**	**Contact Err.**	**Traj. Err.**	**Contact Err.**
GPR	9.53 ± 4.63	0.07 ± 0.03	13.04 ± 6.15	0.04 ± 0.02
GMR	12.39 ± 5.00	0.13 ± 0.05	18.51 ± 6.80	0.09 ± 0.04
k-NN	18.78 ± 4.96	0.49 ± 0.15	29.93 ± 9.02	0.51 ± 0.10

predicted by GPR fits the desired contact locations very well.

Furthermore, for the subsequent results we will use the subscripts Hpp and Crl for the function approximators trained on the databases Hpp and Crl, respectively.

4.3.2 Single-step Motion: Warm start vs Cold start

In this section we compare the results of warm starting Crocoddyl using the function approximators against the cold-start, i.e., using zero initial guess. For each $\boldsymbol{x}$ in the N_{test} samples, GPR and GMR are queried to obtain the initial guess $\boldsymbol{y}$, defined here as the joint configuration trajectory $\boldsymbol{q}$. We do not predict the trajectory of the control input here, but instead calculate it from $\boldsymbol{q}$ assuming quasi-static movement. This computed control input trajectory is denoted as $\boldsymbol{u}_0$. The initial guess is used to warm start Crocoddyl, and the result is compared against the cold-start. We also compare the effect of using Database Hpp and Crl. We use a large convergence threshold (10^{-2}) for Crocoddyl here, based on the assumption that at online computation a very refined optimal motion is not really necessary. The number of iterations is also limited to 20. The query time is ∼5ms for GPR and ∼10ms for GMR in python implementation.

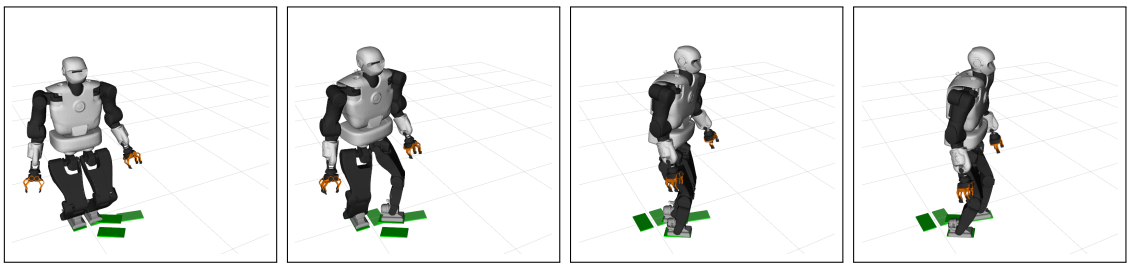
The results can be seen in Table 4.2. The solver is considered successful if it finds a feasible trajectory within the maximum number of iterations. We see from the table that the warm starts results consistently outperform the cold-starts; it justifies our assumption that warm starting Crocoddyl can speed-up the computation time for online MPC.

Although in Table 4.1 we see that the accuracy of GPR outperforms GMR quite substantially, the warm starting performance in Table 4.2 turns out to be very similar, with GPR having very slightly better performance. This is due to the fact that the initial guesses produced by GPR and GMR still go through an optimization process in Crocoddyl, and hence the small difference between the predictions does not affect the results much. This means that while in standard regression tasks the accuracy is highly important, it may be less important in tasks such as producing warm starts, as long as the predicted initial guesses are sufficiently close to the optimal solutions.

Finally, we compared the results between the function approximators trained on Database

Table 4.2 – Comparing Warm start vs Cold start: Single-step

Method	Success Rate	Cost	Num. Iteration
Cold-start	98.50	55.56 ±8.32	9.52 ± 3.94
GPR_{Hpp}	99.00	55.27 ±8.01	5.01 ± 0.74
GMR_{Hpp}	100.00	55.31 ±7.99	4.85 ± 0.66
GPR_{Crl}	100.00	55.32 ±7.99	3.04 ± 0.41
GMR_{Crl}	100.00	55.32 ±7.99	3.06 ± 0.45



(a)

Figure 4.3 – Examples of predicting a multi-step motion by GPR.

Hpp (GPR_{Hpp}, GMR_{Hpp}) and Crl (GPR_{Crl}, GMR_{Crl}). Those trained on the database Crl have lower number of iterations, which justify our step of optimizing the HPP Loco3D dataset by Crocoddyl. The dataset in Crl contain motion samples that are optimal according to Crocoddyl, and therefore they perform better in warm starting Crocoddyl.

4.3.3 Single-step Motion: Evaluating Warm Starts Components

In Section 4.3.2, we only predict the joint configuration trajectory $\boldsymbol{q}$ for warm starting Crocoddyl while the control trajectory $\boldsymbol{u}$ is computed based on the predicted $\boldsymbol{q}$. In this section we evaluate the different performances if we also predict $\boldsymbol{u}$, or use zero control trajectory as warm start. We train two different GPRs for the prediction, one for predicting $\boldsymbol{q}$ and the other one for predicting $\boldsymbol{u}$.

Table 4.3 shows the comparison results. $(\boldsymbol{q}, \boldsymbol{u}_0)$ denotes the warm starts using the predicted $\boldsymbol{q}$ while the control trajectory is computed as $\boldsymbol{u}_0$, the same as in the previous subsection. $(\boldsymbol{q})$ denotes the warm starts using only the joint configuration trajectory while the control trajectory is set to be zero. While the cost remains the same, the number of iterations increases when the control trajectory is omitted from the warm starts. $(\boldsymbol{q}, \boldsymbol{u})$ denotes the warm starts using both predicted joint configuration and control trajectory, which has similar results to $(\boldsymbol{q}, \boldsymbol{u}_0)$ except for the slightly lower number of iterations. Finally, $(\boldsymbol{u})$ denotes the warm starts using only the control trajectory while the joint configuration trajectory is set to zero, and cold-start means warm starting using zero joint configuration and control trajectory.

Table 4.3 – Comparing Warm Start Components: Single-step

Method	Success Rate	Cost	Num. Iteration
$(\boldsymbol{q})$	96.50	55.06 ±8.19	5.30 ± 2.19
$(\boldsymbol{q}, \boldsymbol{u}_0)$	100.00	55.32 ±7.99	3.04 ± 0.41
$(\boldsymbol{q}, \boldsymbol{u})$	100.00	55.32 ±7.99	2.93 ± 0.45
$(\boldsymbol{u})$	97.50	55.36 ±7.95	6.83 ± 2.46
Cold-start	98.50	55.56 ±8.32	9.52 ± 3.94

Table 4.4 – Comparing Warm Start vs Cold Start: Multi-Step

Method	Success Rate	Cost	Num. Iteration
Cold-start	100.00	86.36 ±23.16	6.17 ± 1.52
GPR_{Hpp}	100.00	86.39 ±23.07	6.40 ± 0.73
GMR_{Hpp}	100.00	86.38 ±23.12	7.29 ± 1.32
GPR_{Crl}	100.00	86.47 ±23.16	4.54 ± 0.55
GMR_{Crl}	100.00	86.51 ±23.22	4.71 ± 0.70

Comparing $(\boldsymbol{q}, \boldsymbol{u})$ and $(\boldsymbol{q}, \boldsymbol{u}_0)$, we conclude that predicting the control trajectory from the memory does not give a significant benefit compared to computing it based on $\boldsymbol{q}$. However, control trajectory is still important to be included in the warm starts, as omitting them in $(\boldsymbol{q})$ increases the number of iterations. Finally, comparing $(\boldsymbol{q})$ and $(\boldsymbol{u})$, we conclude that warm starting using only the joint configuration trajectory performs better than using only control trajectory, which has higher cost and number of iterations.

4.3.4 Multi-step Motion: Warm Start vs Cold Start

Finally, we use the single-step motions as a building block for multi-step locomotion, as described in Section 4.2.3. We use HPP Loco3D to generate 50 contact sequences, each consists of $P = 3$ footsteps. GPR_{Hpp}, GMR_{Hpp}, GPR_{Crl} and GMR_{Crl} generate the initial guesses of the joint configuration trajectory $\boldsymbol{q}$ while the control trajectory is computed based on $\boldsymbol{q}$, as in Section 4.3.2. The warm starts are then given to Crocoddyl, and the results are presented in Table 4.4. Interestingly the Hpp database does not perform better than the cold-start (indeed, it is the opposite). This could be due to the fact that we build the motion using a concatenation of three single-step motions, each step starts and ends with zero velocity. The nonoptimality of the Hpp database, added with the nonoptimality of the multi-step motion strategy, seem to render the warm start to be not useful at all. On the contrary, warm starting using the Crl database still speeds-up the convergence, although not as much as in the single-step case (Section 4.3.2). An example of multi-step locomotion warm start produced by GPR is given in Fig. 4.3.

4.4 Conclusion and Future Work

We have presented a framework for learning a memory of motion to warm start an optimal control solver. The proposed approach manages to reduce the average number of solver iterations from ∼9.5 to only ∼3.0 iterations for the single-step motion and from ∼6.2 to ∼4.5 iterations for the multi-step motion, while maintaining the solution quality.

This chapter shows a preliminary result of warm starting an optimal control solver. In the current formulation, Crocoddyl does not include constraints such as torque limit or obstacle avoidance, and we are working towards this direction. When the optimal control formulation becomes more realistic and complex, the solver would need even higher computational time, and the proposed warm starting approach would potentially be even more useful. The final goal is to use the whole framework to control the real robot using MPC.

In this work, we decompose the multistep locomotion into single-step motions, with the initial and goal contact locations as the task that needs to be provided by some other methods. Another potential strategy is to define the task to be the final root pose instead of the contact location. The memory will then determine the contact location to reach the desired root pose. This may allow more flexibility in generating the multi-step motions, as only the root trajectory needs to be provided instead of the contact sequence. Another issue with the single-step decomposition strategy is that the predicted motion has zero velocity at the beginning and the end of each step. We can include the initial and goal root velocity in the task definition, but the task space and hence the required number of samples will increase. While random sampling is used in this work to generate the tasks, active learning [107] can reduce the required number of samples.

Part II Probability Density Estimation Approaches

5 Probabilistic iLQR for Short Time Horizon MPC

In the previous chapters, we have discussed various supervised learning methods for warm starting motion planning and optimal control. Starting from this chapter, we consider the second formulation, i.e., by first transforming the cost function into an unnormalized PDF and approximating the density using a surrogate model. In this chapter, we show that we can obtain a trajectory distribution from an iLQR solver in the form of a Gaussian distribution. This is similar to using Laplace approximation to approximate the PDF using a Gaussian distribution, where the Gaussian is centered at the mode (i.e., the optimal solution) and the covariance is the inverse of the cost function's Hessian, except that we use an approximation of the Hessian (since we approximate the dynamics as linear at every iLQR iteration). We then show that tracking this distribution by a short-horizon MPC controller is more cost-efficient and robust compared to tracking the mean or using an iLQR feedback controller. The proposed method is validated with kinematic control of 7-DoF Franka Emika manipulator and dynamic control of 6-DoF quadcopter in simulation.

Publication Note

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

- T. S. Lembono and S. Calinon, "Probabilistic iterative LQR for short time horizon MPC," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2021, pp. 579–585

Supplementary Material and Source Codes

Video related to this chapter is available at:
https://youtu.be/vjQKqiFCuBg

Source codes related to this chapter are available at:
https://github.com/teguhSL/optimal_control_distribution

5.1 Introduction

Optimal control is a versatile problem formulation that has a large number of applications. With the increasing computational power, it can be used in more complex systems with high degrees of freedom. While the term *control* suggests that the formulation is used to find an optimal control input of a given problem, it is often used also for planning, i.e., to find the state trajectory that minimizes the cost function, typically for a long time horizon to anticipate future events. For example, optimal control is used for planning multiple quadcopters trajectories in a constrained space [109], biped walking generation [110, 111], centroidal dynamics trajectory [41, 112], whole-body motion planning [37, 44], and visual servoing [113]. The planned trajectory is usually tracked by some other controller, e.g., PID controller, or shorter time horizon MPC [113]. The formulation is convenient as one can derive the cost function from the desired behavior and obtain the corresponding optimal state and control trajectory.

However, most optimal control approaches focus on finding only a single optimal output. When the optimal control problem (OCP) is only one part of a multi-step process, which is often the case when it is used for planning, this can be an important limitation. For example, the optimal trajectory may not be feasible for the subsequent step due to the presence of a new obstacle or model errors. In this chapter, we consider a probabilistic formulation of OCP that allows us to obtain not only a single output, but a probability distribution of the output that minimizes the OCP cost.

Among many algorithm variants to solve OCP, iterative Linear Quadratic Regulator (iLQR) [7] is often used in robotics due to its computational efficiency. By iteratively approximating the cost function and the dynamics as quadratic and linear, respectively, an LQR subproblem is solved at each step. It has been used for high dimensional systems such as quadruped and humanoid robots [44], as shown in Chapter 4. We show that a probabilistic solution of iLQR can be obtained efficiently by using the information provided by a standard iLQR solver.

The probabilistic treatment of OCP is discussed by Kappen *et al.* [114], who formulate it as minimizing Kullback-Leibler (KL) divergence. The optimal control solution is the product of the free dynamics and the exponentiated cost (i.e., the exponent of the cost function is considered as an unnormalized probability distribution). Toussaint [115] shows that using an approximate inference method (similar to expectation propagation) to solve an optimal control problem results in an algorithm that is similar to iterative Linear Quadratic Gaussian (iLQG). These works, however, concentrate on improving the solver's efficiency, and not many works actually use the probabilistic solution, except in Guided Policy Search (GPS) where the probabilistic distribution of the iLQR solution is used to provide off-policy samples for reinforcement learning [116].

In this chapter, we propose to use the probability distribution in the context of a tracking

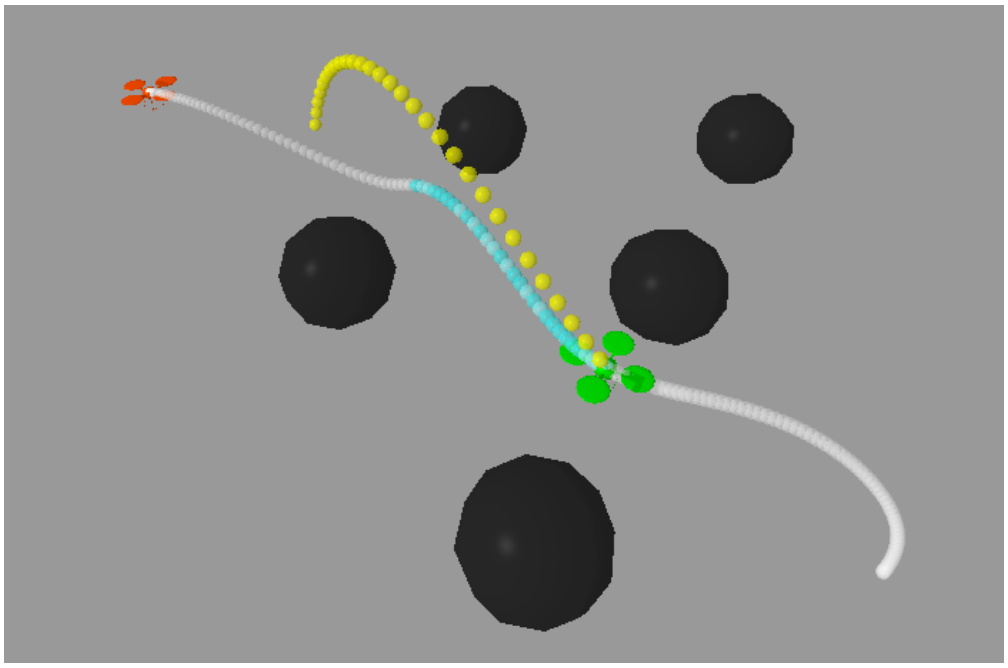


Figure 5.1 – Tracking an iLQR trajectory of a quadcopter. The current position, the goal position, the initial planned trajectory, and the obstacles are shown in green, red, white, and black, respectively. At the current state, an upward velocity disturbance is introduced to the system. For MPC_{marg}, the short horizon OCP reference trajectory (shown in cyan) remains along the planned trajectory because it does not depend on the current state. For MPC_{cond}, the reference trajectory is calculated by conditioning the trajectory distribution on the current state. Since the disturbance adds an upward velocity, the reference trajectory adjusts accordingly, as shown in yellow.

controller. Instead of tracking only the optimal solution, we use a short horizon Model Predictive Control (MPC) to track the trajectory distribution. We show that it improves the tracking performance significantly, with lower cost and better stability. Given a disturbance, a controller tracking only the mean will require the controller to react stiffly to perturbations in all directions, while a controller tracking the distribution can react more intelligently, as it knows in which direction it can move without increasing the cost function too much.

In short, our contribution is twofold. First, we show how to obtain a probability distribution of iLQR solution as a Gaussian distribution using the terms available from a standard iLQR solver. Then we propose a tracking strategy with adaptive gains to follow this distribution using a short time horizon MPC controller, and show that it improves the tracking performance in terms of the total cost and stability, compared to tracking only the mean.

The outline of the chapter is as follows. In Section 5.2 we first explain how to obtain the probability distribution of an LQR problem. In Section 5.3, we extend this approach to

iLQR and show how to track the resulting distribution using a short time horizon MPC. In Section 5.4, we evaluate the method qualitatively on two different systems: manipulator and quadcopter moving around obstacles, and compare the proposed controller against baselines. Section 5.5 concludes the chapter.

5.2 Background

5.2.1 Optimal Control Problem (OCP)

A general discrete OCP consists of a cost function

$$C(\boldsymbol{x}, \boldsymbol{u}) = \sum_{t=0}^{T-1} c_t(\boldsymbol{x}_t, \boldsymbol{u}_t) + c_T(\boldsymbol{x}_T, \boldsymbol{u}_T), \tag{5.1}$$

subject to the dynamics

$$\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t). \tag{5.2}$$

OCP may also have equality and inequality constraints. The objective of solving OCP is to find the sequence of state and control trajectories $(\boldsymbol{x}^*, \boldsymbol{u}^*)$ that minimizes the cost function while respecting the dynamics. In robotics, given the system's high degree of freedom and complexity, most researchers rely on numerical optimization to solve the problem. One of the popular methods is iterative LQR [7] due to its fast computation time. More discussion on OCP can be found in Section 2.1.3.

5.2.2 Probabilistic Solution of Time-Varying Finite Horizon Linear Quadratic Regulator (LQR)

A time-varying finite horizon LQR problem is a subclass of OCP with time-varying linear dynamics

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t,$$

and quadratic costs

$$C(\boldsymbol{x}, \boldsymbol{u}) = \sum_{t=0}^{T-1} (\boldsymbol{x}_t^\top \boldsymbol{Q}_t \boldsymbol{x}_t + \boldsymbol{u}_t^\top \boldsymbol{R}_t \boldsymbol{u}_t) + \boldsymbol{x}_T^\top \boldsymbol{Q}_T \boldsymbol{x}_T.$$

For such class of problems, the solution can be obtained analytically. We focus here on the batch least-squares solution of LQR. We can write the cost function in batch form as

$$C(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{x}^\top \boldsymbol{Q}_s \boldsymbol{x} + \boldsymbol{u}^\top \boldsymbol{R}_s \boldsymbol{u}, \tag{5.3}$$

where $\boldsymbol{Q}_s = \text{blockdiag}(\boldsymbol{Q}_0, \boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_T)$ and $\boldsymbol{R}_s = \text{blockdiag}(\boldsymbol{R}_0, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_{T-1})$ are block diagonal matrices, and the state trajectory $\boldsymbol{x}$ is a function of the control sequence

$\boldsymbol{u}$, i.e.,

$$\boldsymbol{x} = \boldsymbol{S}_x \boldsymbol{x}_0 + \boldsymbol{S}_u \boldsymbol{u}. \tag{5.4}$$

Substituting (5.4) to (5.3), we obtain

$$C(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{u}^\top (\boldsymbol{S}_u^\top \boldsymbol{Q}_s \boldsymbol{S}_u + \boldsymbol{R}_s)\boldsymbol{u} + 2\boldsymbol{u}^\top \boldsymbol{S}_u^\top \boldsymbol{Q}_s \boldsymbol{S}_x \boldsymbol{x}_0 + \boldsymbol{x}_0^\top \boldsymbol{S}_x^\top \boldsymbol{Q}_s \boldsymbol{S}_x \boldsymbol{x}_0.$$

Note that the cost is quadratic in $\boldsymbol{u}$. As discussed in Section 2.2.2, we can view this probabilistically and obtain the probability distribution of $\boldsymbol{u}$ as a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu_u}, \boldsymbol{\Sigma_u})$, where

$$\boldsymbol{\mu_u} = -(\boldsymbol{S}_u^\top \boldsymbol{Q}_s \boldsymbol{S}_u + \boldsymbol{R}_s)^{-1} \boldsymbol{S}_u^\top \boldsymbol{Q}_s \boldsymbol{S}_x \boldsymbol{x}_0, \tag{5.5}$$

$$\boldsymbol{\Sigma_u} = (\boldsymbol{S}_u^\top \boldsymbol{Q}_s \boldsymbol{S}_u + \boldsymbol{R}_s)^{-1}. \tag{5.6}$$

The mean is obtained as the optimum of the cost function in (5.3) by setting the gradient equal to zero, while the covariance matrix is the inverse of the cost function's Hessian.

This distribution tells us that $\boldsymbol{u}$ has the highest probability at the mean $\boldsymbol{\mu_u}$ (corresponding to the point with the lowest cost), and $\boldsymbol{\Sigma_u}$ explains how the probability changes along any direction. With this distribution, we know how to move away from the mean while avoiding significant increase in the cost function. We can also obtain the distribution of the state trajectory, since $\boldsymbol{x}$ is a linear transformation of $\boldsymbol{u}$ according to (5.4), namely

$$p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x} | \boldsymbol{S_x} \boldsymbol{x}_0 + \boldsymbol{S_u} \boldsymbol{\mu_u}, \boldsymbol{S_u} \boldsymbol{\Sigma_u}^{-1} \boldsymbol{S_u}^\top). \tag{5.7}$$

More details about the probability distribution of LQR can be found in [117, 118].

5.3 Method

In this section, we first describe how to obtain the solution of an iLQR problem as a Gaussian distribution. The key insight is to note that each step of iLQR solves an LQR subproblem, of which we can find the probability distribution of the solution. We then show how to track this distribution using a short time horizon MPC.

5.3.1 Probabilistic Solution of iLQR

iLQR can be used to solve more general problems than LQR involving non-quadratic cost functions and nonlinear dynamics. Starting from an initial guess $(\boldsymbol{x}_0, \boldsymbol{u}_0)$, iLQR iteratively refines this guess by approximating the dynamics as linear and the cost function as quadratic around the current solution, and solve an LQR problem to improve

the solution. More details on solving an iLQR problem can be found in Section 2.1.3.

Let us assume that we have reached convergence and obtain the optimal solution as $(\boldsymbol{x}^*, \boldsymbol{u}^*)$. If we again approximate the cost function and the dynamics to obtain a new LQR subproblem around this solution, its solution would be $\delta\boldsymbol{u}^* = \mathbf{0}$, because the optimization gradient is zero at local optima, and the cost function cannot be reduced further. However, the optimal solution $\delta\boldsymbol{u}^* = \mathbf{0}$ does not contain much information about the cost function and the underlying optimization problem. Since this is an LQR problem, we can obtain not only the optimal solution but also the distribution of the solution, as discussed in Section 2.2.2.

We consider the quadratic cost functions and the linear dynamics at the final iteration $k = K$. As explained in Section 5.2.2, we can compute the probability distribution of $\delta\boldsymbol{u}$ as $p(\delta\boldsymbol{u}) = \mathcal{N}(\mathbf{0}, \Sigma_{\delta\boldsymbol{u}^*})$, where $\Sigma_{\delta\boldsymbol{u}^*}$ is calculated using (5.6). The precision matrices $\boldsymbol{Q}_s$ and $\boldsymbol{R}_s$ are computed from the cost derivatives, i.e.

$$\begin{aligned}\boldsymbol{Q}_s &= \text{blockdiag}(\boldsymbol{c}_{\boldsymbol{xx},0}, \boldsymbol{c}_{\boldsymbol{xx},1}, \cdots, \boldsymbol{c}_{\boldsymbol{xx},T}),\\ \boldsymbol{R}_s &= \text{blockdiag}(\boldsymbol{c}_{\boldsymbol{uu},0}, \boldsymbol{c}_{\boldsymbol{uu},1}, \cdots, \boldsymbol{c}_{\boldsymbol{uu},T-1}).\end{aligned}$$

Now the mean of this $p(\delta\boldsymbol{u})$ is a zero vector, corresponding to an optimal solution at convergence. The covariance $\boldsymbol{\Sigma}_{\delta\boldsymbol{u}^*}$ tells us how to move away from the mean while keeping a low cost. Since $\boldsymbol{u} = \boldsymbol{u}^* + \delta\boldsymbol{u}$, we obtain $p(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{u}^*, \boldsymbol{\Sigma}_{\boldsymbol{u}^*})$ where $\boldsymbol{\Sigma}_{\boldsymbol{u}^*} = \boldsymbol{\Sigma}_{\delta\boldsymbol{u}^*}$. That is, the distribution of $\boldsymbol{u}$ is centered around the optimal solution $\boldsymbol{u}^*$, with the same covariance as $\delta\boldsymbol{u}$.

Since $\delta\boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{x}}\delta\boldsymbol{x}_0 + \boldsymbol{S}_{\boldsymbol{u}}\delta\boldsymbol{u}$, the probability distribution of $\delta\boldsymbol{x}$ can be computed as

$$p(\delta\boldsymbol{x}) = \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{\delta\boldsymbol{x}^*}), \quad \text{with} \quad \boldsymbol{\Sigma}_{\delta\boldsymbol{x}^*} = \boldsymbol{S}_{\boldsymbol{u}}\boldsymbol{\Sigma}_{\delta\boldsymbol{u}^*}\boldsymbol{S}_{\boldsymbol{u}}^\top. \tag{5.8}$$

Furthermore, $\boldsymbol{x} = \boldsymbol{x}^* + \delta\boldsymbol{x}$, so $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}^*, \boldsymbol{\Sigma}_{\boldsymbol{x}^*})$, where $\boldsymbol{\Sigma}_{\boldsymbol{x}^*} = \boldsymbol{\Sigma}_{\delta\boldsymbol{x}^*}$. Note that the probability distribution is computed based on the terms that are available from a standard iLQR solver, i.e. the derivatives of the costs and the dynamics. Indeed, we can just run a standard solver until convergence, and extract all the dynamics and cost derivatives to construct the trajectory distribution $\mathcal{N}(\boldsymbol{x}^*, \boldsymbol{\Sigma}_{\boldsymbol{x}^*})$.

The probability distribution is obtained by approximating the cost function and the dynamics. Technically, a Gaussian distribution propagated by nonlinear dynamics would not remain as Gaussian distribution. We obtain a final Gaussian distribution of the overall trajectory because of the linear approximation of the dynamics at each iLQR step, similar to what is done in Extended Kalman Filter [119][1]. Therefore, this distribution

[1] In Bayesian inference, we can view our probabilistic solution as Laplace approximation of the posterior distribution, but we use Gauss-Newton approximation to approximate the Hessian.

only holds locally near the optimal solution $(\boldsymbol{x}^*, \boldsymbol{u}^*)$. Nevertheless, it still contains important information on the local behavior of the cost function and the dynamics around this optimal solution. We will show that this information will be beneficial for the next step, i.e., tracking the optimal trajectory.

5.3.2 Tracking Distribution using Short Time Horizon MPC

From the previous section, we obtain the probability distribution of the state trajectory $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}^*, \boldsymbol{\Sigma}_{\boldsymbol{x}^*})$. Most optimal control approaches only consider the optimal solution $\boldsymbol{x}^*$. As discussed in Section 5.1, the OCP is often an intermediary step to generate a trajectory, which is then tracked by using another controller. This controller can be a simple PID controller or a short time horizon MPC [113] that tracks $\boldsymbol{x}^*$. We argue that tracking only the optimal solution is suboptimal because it does not contain sufficient information about the underlying cost functions. When the system faces disturbances, the controller will force the system to go back to the optimal solution, although the disturbance may be acceptable according to the desired behavior.

Consider as an example the problem of controlling a bicopter to reach a goal position. The mean and the samples from the trajectory distribution are shown in Fig. 5.2c. The main objective is the position of the bicopter at the final time step to be at the goal. This results in a wide trajectory distribution in-between, signifying that it is acceptable to deviate from the mean in the middle of the trajectory. When there is a disturbance, a controller tracking the mean will force the system to come back to the mean, although this is not necessary according to our cost function (which reflects the desired behavior). In contrast, if the controller knows about the distribution, it knows when a disturbance is acceptable and hence does not apply strong correction, following a *minimal intervention principle* [117, 120, 121].

To track the distribution $p(\boldsymbol{x})$, we propose to use a short time horizon MPC. At each time step, we solve an OCP with horizon T_s, which is much shorter than the long horizon T used to plan the trajectory in Section 5.3.1. We solve this short horizon OCP with iLQR just as we do for the long horizon OCP, but in practice we can use any OCP solver for this part. The cost function

$$c_t(\boldsymbol{x}_t, \boldsymbol{u}_t) = (\boldsymbol{x}_t - \bar{\boldsymbol{x}}_t)^\top \boldsymbol{Q}_t (\boldsymbol{x}_t - \bar{\boldsymbol{x}}_t) + l(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{u}_t^\top \boldsymbol{R} \boldsymbol{u}_t \tag{5.9}$$

is designed to track a reference trajectory, where $\bar{\boldsymbol{x}}_t$ is the reference state at time t, $l(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is the collision cost, and the precision matrices $\boldsymbol{Q}_t$ are designed to correspond to the probability distribution of $\bar{\boldsymbol{x}}_t$. If $\bar{\boldsymbol{x}}_t$ has a large variance, we do not want to track this state too precisely, so $\boldsymbol{Q}_t$ should be small, and vice-versa.

How to relate $\bar{\boldsymbol{x}}_t$ and $\boldsymbol{Q}_t$ to the trajectory distribution $\mathcal{N}(\boldsymbol{x}^*, \boldsymbol{\Sigma}_{\boldsymbol{x}^*})$ that we find earlier? One way is to use the marginal distribution $p_m(\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{x}_t^*, \boldsymbol{\Sigma}_{\boldsymbol{x}^*,t})$, where $\boldsymbol{x}_t^*$ is the

Algorithm 4 Tracking iLQR distribution

1: Solve the long horizon (T) OCP by iLQR;
2: Calculate $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}^*, \boldsymbol{\Sigma}_{\boldsymbol{x}^*})$;
3: **for** $t = \tau < T$ **do**
4: Estimate the current state $\boldsymbol{x}_\tau$;
5: Extract the probability $p(\boldsymbol{x}_{\tau:\tau+T_s})$ from $p(\boldsymbol{x})$;
6: Compute the conditional probability $p_c(\boldsymbol{x}_{\tau+1:\tau+T_s}|\boldsymbol{x}_\tau)$;
7: Extract the ref. traj. distribution $p_c(\boldsymbol{x}_t|\boldsymbol{x}_\tau)$ from $p_c(\boldsymbol{x}_{\tau+1:\tau+T_s}|\boldsymbol{x}_\tau)$ for $t = \tau + 1$ to $\tau + T_s$;
8: Construct the cost function of the short horizon (T_s) OCP (Eq. 5.9);
9: Solve the short horizon (T_s) OCP using iLQR;
10: Execute the first control input $\boldsymbol{u}_t$;
11: **end for**

component of $\boldsymbol{x}^*$ at time t, and $\boldsymbol{\Sigma}_{\boldsymbol{x}^*,t}$ is the corresponding block matrix, so that $\bar{\boldsymbol{x}}_t = \boldsymbol{x}_t^*$ and $\boldsymbol{Q}_t = \boldsymbol{\Sigma}_{\boldsymbol{x}^*,t}^{-1}$. However, when we do this, we neglect the correlation between the different time steps in $\boldsymbol{\Sigma}_{\boldsymbol{x}^*}$. Instead, we can use the conditional distribution based on the current state.

At time step $t = \tau$, we observe the current state at $\boldsymbol{x}_\tau$. We first extract the probability $p(\boldsymbol{x}_{\tau:\tau+T_s})$ from $p(\boldsymbol{x})$, and write this in partition format

$$p\begin{pmatrix}\boldsymbol{x}_\tau \\ \boldsymbol{x}_{\tau+1:\tau+T_s}\end{pmatrix} = \mathcal{N}\left(\begin{pmatrix}\boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2\end{pmatrix}, \begin{pmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22}\end{pmatrix}\right).$$

We can then condition on $\boldsymbol{x}_\tau$ to obtain $p_c(\boldsymbol{x}_{\tau+1:\tau+T_s}|\boldsymbol{x}_\tau) = \mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$ where

$$\boldsymbol{\mu}_c = \boldsymbol{\mu}_2 + \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}(\boldsymbol{x}_\tau - \boldsymbol{\mu}_1), \quad \boldsymbol{\Sigma}_c = \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}.$$

The conditional distribution $p_c(\boldsymbol{x}_t|\boldsymbol{x}_\tau) = \mathcal{N}(\boldsymbol{\mu}_{c,t}, \boldsymbol{\Sigma}_{c,t})$ can then be obtained from $p_c(\boldsymbol{x}_{\tau+1:\tau+T_s}|\boldsymbol{x}_\tau)$ for $t = \tau + 1$ to $\tau + T_s$. $\boldsymbol{\mu}_{c,t}$ is the t_{th} element of $\boldsymbol{\mu}_c$, with the corresponding block diagonal $\boldsymbol{\Sigma}_{c,t}$. This can be used to set $\bar{\boldsymbol{x}}_t = \boldsymbol{\mu}_{c,t}$ and $\boldsymbol{Q}_t = \boldsymbol{\Sigma}_{c,t}^{-1}$ in (5.9).

We demonstrate in the next section that formulating the tracking controller to follow the reference trajectory distribution will improve the tracking performance. The complete algorithm is given in Table 4.

It is also possible to use the long horizon OCP in MPC fashion. However, the long horizon requires longer computational time, making it difficult to be used in real-time. The computation time is linear with respect to the time horizon T. Formulating the controller as a short time horizon MPC to track the distribution speeds up the computational time significantly, while still being able to consider the future events using the distribution. With shorter computational time, the short horizon MPC can better adapt to real time changes such as new obstacles along the trajectory. We can also add hard constraints to the OCP, e.g., using the augmented Lagrangian method [122].

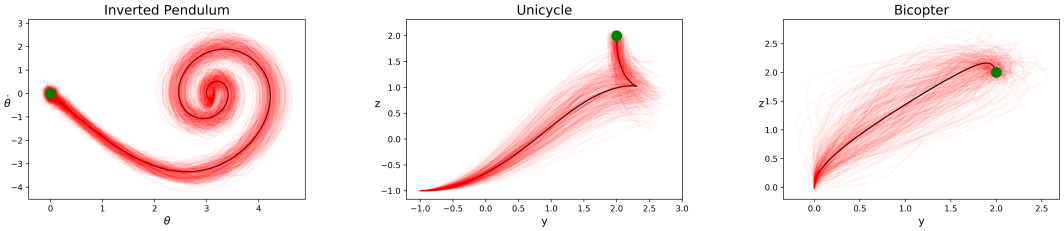


Figure 5.2 – Trajectory distribution for different systems, shown at selected axes. The mean trajectory is shown as a black line, and the trajectory samples drawn from the distribution are shown as red thin lines. The goal is shown in green.

5.4 Experiments

Fig. 5.2 shows the probability distribution on several systems, i.e., inverted pendulum (1 DoF), unicycle (3 DoF), and bicopter (3 DoF). The system description can be found in [120, 49, 70]. The distribution is determined by both the dynamics and the cost function. As we put a high cost at $t = T$ to reach the goal and low cost at $t < T$, the distribution is narrow around the goal and quite wide in the middle. Thus the tracking controller knows that deviation from the mean is more tolerable in the middle of the trajectory, which agrees with our intuition.

We quantitatively evaluated the proposed algorithm on a robotic manipulator (i.e., Franka Emika robot) and a quadcopter, with the task to move from an initial configuration to a goal location while avoiding obstacles. Following Algorithm 4, a long horizon iLQR is first solved to obtain the trajectory distribution $p(\boldsymbol{x})$ that reaches the goal. The cost function is defined as

$$C(\boldsymbol{x}, \boldsymbol{u}) = \sum_{t=0}^{T-1} \Big((\boldsymbol{x}_t - \boldsymbol{x}_{\text{goal}})^\top \boldsymbol{Q} (\boldsymbol{x}_t - \boldsymbol{x}_{\text{goal}}) + \boldsymbol{u}_t^\top \boldsymbol{R} \boldsymbol{u}_t + l(\boldsymbol{x}_t, \boldsymbol{u}_t) \Big) + (\boldsymbol{x}_T - \boldsymbol{x}_{\text{goal}})^\top \boldsymbol{Q}_T (\boldsymbol{x}_T - \boldsymbol{x}_{\text{goal}}), \tag{5.10}$$

where $\boldsymbol{Q}$ and $\boldsymbol{Q}_T$ are the precision matrices, $\boldsymbol{Q}$ being much smaller than $\boldsymbol{Q}_T$. $l(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is a collision avoidance cost formulated as a potential field that is active only when the quadcopter is in collision. After running the iLQR solver until convergence, we can compute the state trajectory distribution $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}^*, \boldsymbol{\Sigma}_{\boldsymbol{x}^*})$. Section 5.4.1 describes various tracking algorithms to be compared, with the results discussed in Section 5.4.2.

5.4.1 Tracking Algorithms

To illustrate the benefit of tracking the trajectory distribution, we compare four different algorithms:

iLQR Feedback Control ($\text{iLQR}_{\text{feed}}$)

$\text{iLQR}_{\text{feed}}$ tracks the mean trajectory $\boldsymbol{x}^*$ using the feedforward control input $\boldsymbol{u}^*$ and the feedback term $\boldsymbol{K}_t$,

$$\boldsymbol{u}_t = \boldsymbol{u}_t^* + \boldsymbol{K}_t(\boldsymbol{x}_t - \boldsymbol{x}_t^*),$$

where $\boldsymbol{x}_t$ is the current observed state. While this requires very little computation, the feedback gain is only good around the planned trajectory $(\boldsymbol{x}^*, \boldsymbol{u}^*)$, and it can be unstable for large disturbance.

Short time horizon MPC tracking the mean trajectory (MPC_{mean})

MPC_{mean} solves an OCP problem at each time step with the cost function in (5.9). The reference state is obtained from the planned trajectory, i.e., $\bar{\boldsymbol{x}}_t = \boldsymbol{x}_t^*$, whereas the precision matrix $\boldsymbol{Q}_t$ is set to be the same as $\boldsymbol{Q}_T$ in (5.10), i.e., a high-gain tracking.

Short time horizon MPC tracking the marginal distribution (MPC_{marg})

MPC_{marg} solves an OCP problem at each time step with the cost function in (5.9). $\bar{\boldsymbol{x}}_t$ and $\boldsymbol{Q}_t$ are set according to the marginal distribution $p_m(\boldsymbol{x}_t)$ as described in Section 5.3.2.

Short time horizon MPC tracking the conditional distribution (MPC_{cond})

MPC_{cond} solves an OCP problem at each time step with the cost function in (5.9). $\bar{\boldsymbol{x}}_t$ and $\boldsymbol{Q}_t$ are set according to the conditional distribution $p_c(\boldsymbol{x}_t|\boldsymbol{x}_\tau)$ as described in Section 5.3.2.

5.4.2 Tracking Comparison

We run the experiments on two systems, the 7-DoF Franka Emika manipulator and 6-DoF quadcopter. The manipulator task is kinematic control to reach a desired end effector pose where the state $\boldsymbol{x} \in \mathbb{R}^{14}$ consists of the joint angles and velocities, and the control $\boldsymbol{u} \in \mathbb{R}^7$ is the joint acceleration command. The dynamics of the system is therefore linear (i.e., double integrator), but the cost is non-quadratic as it involves the end effector pose. The quadcopter task is dynamic control to reach a desired goal location where the state $\boldsymbol{x} \in \mathbb{R}^{12}$ consists of the position, orientation, and its corresponding velocities, while the control $\boldsymbol{u} \in \mathbb{R}^4$ consists of the four propellers thrusts. For both systems, the horizon is set to be $T = 150$ and $T_s = 30$ for the long and short horizon, respectively, with 50 ms interval. The long horizon is set to be long enough to reach the goal at the end of the trajectory, while the short horizon is set to be as short as possible while still managing to obtain good performance. As the computation time of DDP is linear with respect to the

horizon length, the iteration time for short horizon DDP is around 5 times faster than the long horizon DDP. The cost function is defined in (5.10). Fig. 5.1 shows an example of the optimal iLQR trajectory $\boldsymbol{x}^*$ for the quadcopter, shown in white.

After solving the long horizon iLQR to obtain the trajectory distribution $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}^*, \boldsymbol{\Sigma}_{\boldsymbol{x}^*})$, we track the trajectory using the algorithms in Section 5.4.1. During tracking we introduce external velocity disturbance to the system, and evaluate how well the algorithms overcome the disturbance with varying magnitude. We consider two types of disturbance: *impulse* and *time-varying* disturbance, each of which has three levels of disturbance: *small, medium, and large*. For the impulse disturbance, we introduce external velocity disturbance for a short time, i.e., during 2 time steps, with the magnitude of $(0.5, 1.5, 3.)$ for manipulator and $(0.2, 0.7, 1.5)$ for quadcopter (the units are m/s and rad/s for linear and angular velocity, respectively) in random direction. For the time-varying disturbance, we introduce time-varying velocity disturbance between $t = 30$ and $t = 100$ with the magnitude of $(0.1, 0.3, .6)$ for manipulator and $(0.07, 0.2, 0.4)$ for quadcopter. These numbers are chosen to specifically demonstrate the different relative performance among the controllers at each level of disturbance, as will be discussed shortly after this. After each task completion, we evaluate the cost of the resulting state and control trajectories by using the original cost function in (5.10). For each disturbance level, we run N=50 experiments with disturbance in random direction. As the costs vary greatly between each experiment, we normalize the cost by the minimum cost achieved at each experiment, so a cost of value 1.0 means that the method achieves the best cost for that particular experiment. The mean and standard deviation of the normalized cost for each method is given in Table 5.1 and 5.2.

The results from both tables give the same conclusions, i.e., the relative performance of the controllers are the same for both impulse and time-varying disturbance. For the Franka Emika experiment, $\text{iLQR}_{\text{feed}}$ has the best performance for all disturbance levels. However, it only performs best at small disturbance for the quadcopter. As the disturbance increases, $\text{iLQR}_{\text{feed}}$ becomes less reliable and the cost increases. We observe that $\text{iLQR}_{\text{feed}}$ often becomes very unstable at large disturbance, resulting in diverging movements (these samples were not considered in the results of Table 5.1 and 5.2). Large disturbance move the system far from the planned trajectory where the feedback gain is no longer valid. This is especially true for underactuated systems such as bicopter and quadcopter where each control is not directly associated with a particular state, because the optimal feedback gain changes its sign (not only magnitude) according to the current system state. Note that the standard deviation of $\text{iLQR}_{\text{feed}}$ cost at large disturbance in Table 5.1 and 5.2 for quadcopter is very high. On the other hand, the kinematic control of manipulator involves a linear dynamic and fully actuated system, so the feedback gain remains good at large disturbance.

The three remaining controllers are more stable even at large disturbance. MPC_{mean} has the largest cost among the three because it tries to track the mean trajectory precisely

Table 5.1 – Tracking performance cost comparison (impulse disturbance)

System	Method	Disturbance		
		Small	Medium	Large
Franka Emika	$\text{iLQR}_{\text{feed}}$	**1.00 ± 0.0**	**1.00 ± 0.0**	**1.00 ± 0.0**
	MPC_{mean}	1.02 ± 0.0	1.03 ± 0.0	1.05 ± 0.0
	MPC_{marg}	1.02 ± 0.0	1.03 ± 0.0	1.07 ± 0.0
	MPC_{cond}	1.01 ± 0.0	1.01 ± 0.0	1.01 ± 0.0
Quadcopter	$\text{iLQR}_{\text{feed}}$	**1.00 ± 0.0**	1.10 ± 0.3	1.99 ± 2.9
	MPC_{mean}	1.48 ± 0.1	2.58 ± 1.0	3.41 ± 1.2
	MPC_{marg}	1.26 ± 0.0	1.23 ± 0.1	1.25 ± 0.2
	MPC_{cond}	1.08 ± 0.0	**1.05 ± 0.0**	**1.20 ± 0.4**

Table 5.2 – Tracking performance cost comparison (time-varying disturbance)

System	Method	Disturbance		
		Small	Medium	Large
Franka Emika	$\text{iLQR}_{\text{feed}}$	**1.00 ± 0.00**	**1.00 ± 0.0**	**1.00 ± 0.0**
	MPC_{mean}	1.02 ± 0.00	1.03 ± 0.0	1.05 ± 0.0
	MPC_{marg}	1.02 ± 0.00	1.03 ± 0.0	1.06 ± 0.0
	MPC_{cond}	1.01 ± 0.00	1.01 ± 0.0	1.01 ± 0.0
Quadcopter	$\text{iLQR}_{\text{feed}}$	**1.01 ± 0.06**	1.33 ± 0.7	2.60 ± 4.4
	MPC_{mean}	1.54 ± 0.18	3.06 ± 1.3	4.49 ± 2.0
	MPC_{marg}	1.26 ± 0.03	1.25 ± 0.1	1.26 ± 0.2
	MPC_{cond}	1.07 ± 0.02	**1.05 ± 0.1**	**1.20 ± 0.6**

without knowing the underlying cost function and the desired behavior. MPC_{marg} performs better because it takes into account the variance of the planned trajectory, but it ignores the correlations between different time steps. The smallest cost is achieved by MPC_{cond} because it computes the future reference trajectory by considering the current state, i.e., by computing the conditional distribution. This enables the controller to adapt to the disturbance better. In practice, it means that the controller exerts less control to overcome the disturbance while still achieving the objective.

Fig. 5.1 illustrates the difference with the quadcopter experiment. The planned trajectory $\boldsymbol{x}^*$ is shown in white. At the current state, an upward velocity disturbance is introduced to the system. Since MPC_{mean} and MPC_{marg} do not consider the current state when computing the reference trajectory, the disturbance does not affect its reference trajectory (shown in cyan), which is always along the planned trajectory $\boldsymbol{x}^*$. On the other hand, the reference trajectory for MPC_{cond} is computed by conditioning on the current state. The algorithm knows that the quadcopter is moving with an additional upward velocity, and it adjusted the reference trajectory accordingly (shown in yellow). While MPC_{mean} and

MPC_{marg} force the quadcopter to go back to the mean, MPC_{cond} use the information from the distribution more effectively to move according to the desired behavior. Note that the cost function in (5.10) dictates that the important task is to reach the goal at the end, while the path in-between is less important. This is well represented by the trajectory distribution.

However, the conditional distribution is also obtained from local approximation. This means that for very large disturbance, it can still result in undesired behavior, such as producing a reference trajectory that has a high cost. We indeed find that MPC_{marg} is more stable than MPC_{cond} at very large disturbance. We can see in Table 5.1 and 5.2 that the normalized cost of MPC_{cond} increases to almost the same as MPC_{marg} at the large disturbance level. At even larger disturbance, we observe that MPC_{marg} performs the best and most stably. However, the local approximation does not affect MPC_{cond} as much as $\text{iLQR}_{\text{feed}}$, because the conditional distribution is only used as the reference trajectory, while the underlying controller in MPC_{cond} still considers the actual dynamics around the current state to compute the control command during the tracking. In contrast, for $\text{iLQR}_{\text{feed}}$, the local approximation from the planning step directly determines the controller gain, which remains unchanged during tracking. This poor approximation results in unstable controllers at large disturbance, especially for underactuated and highly nonlinear systems such as quadcopter.

5.4.3 Discussion

In this work, we have shown an example of kinematic control of a serial manipulator with non-quadratic cost functions. We did not use dynamic control for this system because the derivative of the dynamics (the matrix $\boldsymbol{A}_t$) is often unstable, i.e., some of its eigenvalues are outside the unit circle. Since computing the distribution involves the multiplication of the matrices $\prod_{t=1}^{T-1} \boldsymbol{A}_{T-t}$, the unstable eigenvalues causes the resulting matrix to be numerically poor and the distribution cannot be computed. More research still needs to be done to handle this issue.

The proposed framework can also be extended to control systems with both state and control constraints. Augmented Lagrangian iLQR (AL-iLQR) is discussed in [122] to handle such systems. Note that each iteration in AL-iLQR still solves an LQR problem, so we can still obtain the distribution as we do here. The resulting distribution would take the constraints into consideration, i.e., having higher probability when the trajectory satisfies the constraints.

5.5 Conclusion and Future Work

In this chapter, we have shown that by obtaining the distribution of solution from iLQR and tracking this distribution, the resulting controller is more cost-efficient and robust to disturbance. The tracking performance has been shown to be better than tracking only the mean trajectory or using the iLQR feedback controller. The latter is very unstable when moving far from the planned trajectory due to large disturbance, especially for underactuated systems such as quadcopter. The iLQR distribution can be calculated using the information available from a standard iLQR solver. The method can also be extended to constrained OCP methods such as Augmented Lagrangian iLQR [122].

6 Learning Constrained Distributions of Robot Configurations

In high-dimensional robotic systems, the manifold of the valid configuration space often has a complex shape, especially under constraints such as end effector orientation or static stability. In this chapter, we propose a generative adversarial network approach to learn the distribution of valid robot configurations under such constraints. It can generate configurations that are close to the constraint manifold. We present two applications of this method. First, by learning the conditional distribution with respect to the desired end effector position, we can do fast inverse kinematics even for very high degrees of freedom systems. Then, we use it to generate samples for a sampling-based constrained motion planner to reduce the necessary projection steps, speeding up the computation. We validate the approach in simulation using the 7-DoF Franka Emika manipulator and the 28-DoF Talos humanoid robot.

This chapter is a result of the joint work with E. Pignat and J. Jankowski. E. Pignat helped in formulating and providing the codes for GAN, and J. Jankowski helped in the experiment with the Franka Emika manipulator.

Publication Note

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

- T. S. Lembono, E. Pignat, J. Jankowski, and S. Calinon, "Learning constrained distributions of robot configurations with generative adversarial network," *IEEE Robotics and Automation Letters (RA-L)*, vol. 6, no. 2, pp. 4233–4240, 2021

Supplementary Material and Source Codes

Video related to this chapter is available at:
https://youtu.be/tEsclKWS38

Source codes related to this chapter are available at:
https://github.com/teguhSL/learning_distribution_gan

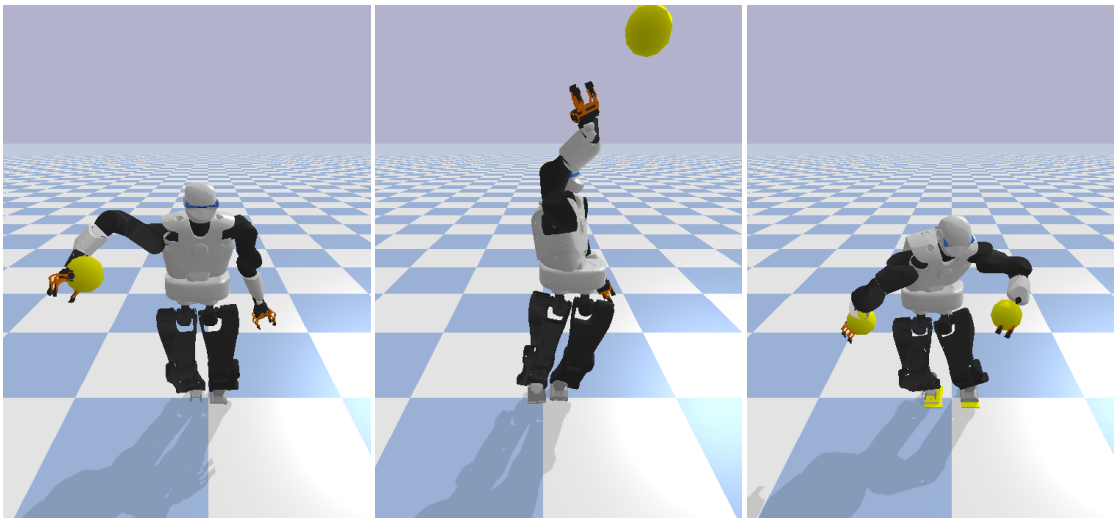
6.1 Introduction

Generative Adversarial Network (GAN) [67] is a powerful method to learn complex distributions. It is particularly popular in computer vision to learn the distribution of images from a dataset. Some of the applications include generating high resolution images [123], text-to-image synthesis [124], and interactive art [125]. Considering the recent success of deep learning techniques in robotics, e.g., [126], we propose to adapt GAN to the context of robotics, i.e., to learn the distribution of valid robot configurations in constraint manifolds.

In robotics, *configuration space* refers to the space of possible robot configurations that may include joint angles for revolute joints, joint translations for prismatic joints, and the base pose for floating-based robots. This concept is very important in motion planning, because the planning often needs to be done in the configuration space. Due to the presence of constraints, the valid configurations lie in some manifold of the configuration space, the shape of which can be complicated and often cannot be parameterized explicitly. Inequality constraints, such as obstacle avoidance, result in a manifold with non-zero volume, and the standard approach to sample from this manifold is to do rejection sampling. For equality constraints (such as fixed feet locations or end effector orientation), however, the manifold volume is reduced to zero, as the manifold has lower dimension than the original space. Rejection sampling does not work in this case because there is zero probability that the random sample will be on the manifold. A common approach is to project the random samples to the manifold, and this projection step takes a significant portion of the planning computation time. By using GAN to learn the distribution of valid robot configurations on a manifold, we can sample from this manifold effectively, such that the generated samples are close to the desired manifold.

Having learned the valid distributions, we show that it can be used in two applications: inverse kinematics (IK) and sampling-based constrained motion planning. Analytical IK is typically only available for robots with 6-DoF or lower. For higher DoF robots, the most common method is to use numerical IK where we start with an initial robot configuration (often selected randomly by uniform sampling), and rely on optimization to find the configuration that reach the desired pose while satisfying the constraints. In our proposed framework, we show that GAN can produce good initial configurations that are close to achieving the target, resulting in a faster numerical IK with higher success rate as compared to uniform sampling initialization. Furthermore, sampling-based constrained motion planning also involves a high number of projections of random samples to the constraint manifold. We show that by replacing the uniform sampling with GAN, the planning time can be reduced significantly.

One particular difficulty of learning a distribution is when the target distribution is multimodal, as is the case in many robotic systems. For example, the conditional distribution of a 6-DoF manipulator given a desired end effector pose is multimodal



(a) (b) (c)

Figure 6.1 – Using GAN for obtaining approximate IK solutions. The targets are depicted in yellow. In (a), the target is reachable. When the target is out of reach (b), GAN still outputs a configuration close to the constraint manifold. We can also give the four limbs position as the IK targets (c).

because there are many different configurations that are associated with the pose. To overcome this, we propose to use an ensemble of neural networks as the generator in GAN. Each neural network can converge to a different mode, and we get better coverage of the distribution as compared to using a single network.

The remainder of the chapter is as follows. In Section 6.2, we review existing work on learning sampling distributions and constrained-based motion planning. Section 6.3 describes the GAN framework and how we use it for inverse kinematics and constrained motion planning. The experiments with 7-DoF Franka Emika and 28-DoF Talos [103] are presented in Section 6.4. Finally, we conclude with a few remarks in Section 6.5.

6.2 Related Work

6.2.1 Learning Sampling Distribution

In [127, 128], Gaussian Mixture Model (GMM) is used to learn the sampling distribution based on previously planned paths. The distribution is used to either generate biased samples for the sampling algorithm or to construct a repetition roadmap that guides towards finding the solution. It speeds up the computation compared to uniform sampling, but it is difficult to generalize to different situations (e.g., different obstacles). Moreover, GMM does not scale well to higher dimensional systems. GMM is also used in [129]

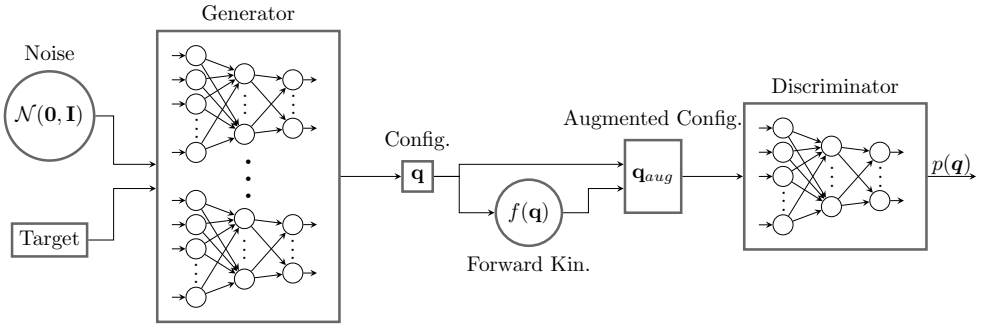


Figure 6.2 – The proposed GAN framework for learning the distribution of valid robot configurations. The generator consists of an ensemble of N_{net} neural networks, while the discriminator consists of a single neural network. Besides a Gaussian noise as in standard GAN, we also add the end effector target(s) as additional input to the generator. The output of the generator is then augmented by additional features, i.e., the corresponding end effector poses, before being given to the discriminator.

to learn the feasibility constraint of center of mass (CoM) position with respect to the whole-body dynamics. Conditional Variational Autoencoder (CVAE) is used in [130], also trained based on data from demonstrations or planned paths, but it is not implemented on constrained systems. Convolutional Autoencoder is used in [131] to learn the motion manifold of human motion, but not in the context of motion planning.

In contrast to the above approaches, we propose to use Generative Adversarial Network (GAN) to learn the sampling distribution of constrained robotic systems and apply it to the 7-DoF Franka Emika and the 28-DoF Talos. GAN scales better with higher dimensions compared to GMM, and it is easy to learn a conditional distribution with respect to some task (such as end effector pose). We use it to initialize IK very efficiently while considering various constraints (joint limit, static stability, foothold location). A similar effort to initialize IK is done in [132] by storing previously computed end effector configurations in an octree data structure, indexed by the end effector positions. However, in that work, each limb is treated separately from the body, and only the kinematics is considered without any stability criterion when retrieving the initial guess. Our GAN approach produces configurations that are already close to the constraint manifold, which can include the stability criterion. In [133], GAN is successfully used to learn inverse kinematics and dynamics of 8-DoF robot manipulator with the real data, but it does not consider any constraint on the robotic system. Furthermore, we show that the GAN sampler can also be used to improve the sampling-based constrained motion planning algorithm.

6.2.2 Constrained Motion Planning

A review of various approaches in sampling-based motion planning for constrained systems can be found in [134]. Among the others, projecting the samples to the constraint manifold is a simple but very generalizable way of extending the standard Rapidly-exploring Random Tree (RRT) to constrained problems. Instead of doing rejection sampling, the samples are projected to the constraints manifold [135, 34]. While it works well, the projection steps take most of the planning time. In [35], Yang *et al.* compare various algorithms for motion planning of humanoid robot. They report that the projection step takes more than 95% of the planning time. Several research lines attempt to reduce this time computation. For example, in [136], Stilman *et al.* use the tangential direction of the constraint manifold to always move while staying close to the manifold. In [137], Kanehiro *et al.* simplify the humanoid structure by splitting it into multiple 6-DoF structures and then perform analytical IK.

In our proposed method, we can generate samples that are already close to the constraints manifold, and the approach is generalizable to most robotic systems. This will reduce the necessary number of projection steps significantly and hence lower the computation time. Additionally, optimization-based approaches such as CHOMP [3] and TrajOpt [4] can solve constrained planning quickly by including the constraints in the optimization problem. However, since the problem is highly nonlinear, these methods often require good initial guesses, otherwise they may have a lower success rate even for simple problems [8]. In contrast, sampling-based motion planning can find a global solution with probabilistic completeness guarantee [34], provided that the sampler can cover the entire feasible configuration space, which is the case for uniform sampling.

6.3 Method

In this section, we present the proposed GAN framework for learning the robot distribution. We then propose two applications: inverse kinematics and constrained motion planning.

6.3.1 Generative Adversarial Framework

In the generative adversarial framework [67], a generator $G(\boldsymbol{z}; \boldsymbol{\theta_G})$ is trained to transform the input noise $\{\boldsymbol{z}\}$ drawn from $p_z(\boldsymbol{z})$ (typically a unit Gaussian) into samples $\{\boldsymbol{q}\}$ that look similar to the data distribution. To do this, a discriminator $D(\boldsymbol{q}; \boldsymbol{\theta_D})$ is trained in parallel to output the probability $p(\boldsymbol{q})$ that tells whether $\boldsymbol{q}$ comes from the dataset or the generator. The training of GAN is therefore like a game between the generator and the discriminator where one tries to beat the other. The generator and discriminator are neural networks (with parameters $\boldsymbol{\theta_G}$ and $\boldsymbol{\theta_D}$, respectively) trained with Stochastic Gradient Descent (SGD).

In our approach, GAN is trained to generate configurations $\{\boldsymbol{q}\}$ that lie in some constraints manifold. The following sections explain some changes to standard GAN that we propose to better suit robotic applications. Unlike in images, we can more easily incorporate several forms of prior knowledge of what constitutes good configurations in the form of additional cost functions and transformations. To better handle multimodal distributions, we use an ensemble of neural networks as the generator. The framework is depicted in Fig. 6.2.

Additional inputs

In standard GAN, the input to the generator is a noise sampled from a Gaussian distribution. To obtain a conditional distribution, we include the task parameters as additional inputs to the generator. The task parameters here correspond to the desired end effector pose(s), but other additional tasks are also possible.

Additional costs

The training cost for the generator normally consists of the cost of tricking the discriminator to classify its samples as dataset samples. In robotics, however, we can add other costs that can be used to evaluate the quality of the samples based on the knowledge of the robotic system. Here we include several costs:

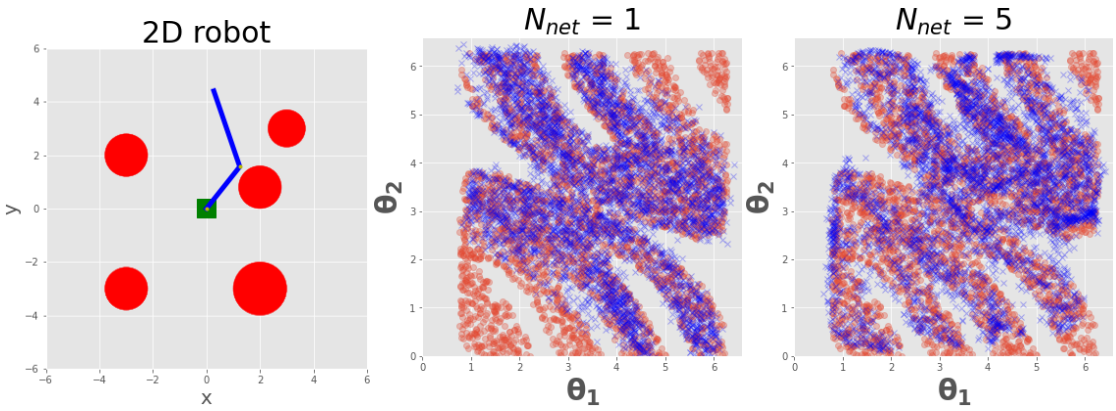
- The cost of end effector targets $c_{\text{ee}}(\boldsymbol{q})$. From the samples generated by G, we can compute the forward kinematics to obtain the end effector positions and compare this to the desired end effector target (given as the input to the generator).

- The cost of static stability $c_{\text{s}}(\boldsymbol{q})$. To achieve static stability, the CoM projection on the ground should be located inside the foot support polygon.

- The cost of joint limits $c_l(\boldsymbol{q})$. The cost is zero if it is within the limit, and increasing outside the limit.

Output Augmentation

Instead of feeding the configurations directly to the discriminator, we augment the configurations by some transformations, e.g., end effector poses. This helps the discriminator to discern between good and bad samples according to the relevant features. Other transformations such as CoM location can also be added.



(a) (b) (c)

Figure 6.3 – Illustrative example of a 2-DoF robot with obstacles. (a) shows the robot with circular obstacles. The configurations (i.e., joint angles) that are not in collision are plotted in (b) and (c) as red circles. We learn this distribution using GAN. In (b), we use one neural network as the generator, and the GAN samples are plotted as blue crosses. We see that the samples do not cover the whole distribution. Using an ensemble of 5 networks in (c), we manage to cover most of the distribution.

Ensemble of networks

When the desired distributions are multimodal, GAN often converges to only some modes of the distribution. This is known as the Helvetica scenario or mode collapse [67]. For example, when the desired distribution is a GMM, GAN may converge to only some of the mixture components, and not all of them. This is a major problem in the robotics context, especially for motion planning, because omitting some portion of the configuration space means reducing the probability of finding feasible solutions. To overcome this, we use an ensemble of N_{net} neural networks as the generator. Given an input, each network generates a corresponding robot configuration, and each one is trained as a standalone generator. When the desired distribution is multimodal, each network may converge to a different mode.

The advantage of using an ensemble of networks as the generator can clearly be seen using an illustrative 2-link robot example. Fig. 6.3a shows the setup of the robot surrounded by obstacles. The configuration consists of the two joint angles. The valid configurations (i.e., the ones without collision) are plotted in Fig. 6.3b and Fig. 6.3c as red circles, and GAN is trained to learn this distribution. When using only one network for the generator, GAN converges to only some part of the configuration space, as depicted in Fig. 6.3b (the GAN samples are plotted as blue crosses). Using $N_{\text{net}} = 5$ networks, we manage to cover most of the configuration space (Fig. 6.3c).

6.3.2 Inverse Kinematics

The computation time for numerical IK really depends on how close the initial guess is to the optimal solution. As discussed in Section 6.3.1, we can obtain good initial guesses by sampling from GAN while giving the end effector poses as additional inputs to the generator. The resulting configurations would be close to the desired poses and the constraint manifold, reducing the required number of IK iterations significantly.

Finally, although the formulation here is presented for inverse kinematics, the same formulation can be used to project robot configurations to the constraint manifold by omitting the cost on the end effector position. Further details will be provided in Section 6.4.

6.3.3 Constrained Motion Planning

The standard approach in sampling-based constrained motion planning [34] is based on RRT, with the addition of the projection step; each sample added to the tree is projected first to the constraint manifold. The projection step is formulated as an optimization problem as discussed in Section 6.3.2. Algorithm 5 describes the steps for constrained RRT (cRRT) for reaching a goal in task space, starting from an initial configuration $\boldsymbol{q}_0$. We refer to [34] for more details of the algorithm.

First, we start with a tree G initialized with the node $\boldsymbol{q}_0$. From the given goal task in Cartesian space, we compute K goal configurations (by numerical IK). The following iterations then attempt to extend the tree to one of these goals. At each iteration, a random sample $\boldsymbol{q}_{\text{rand}}$ is generated. Nearest neighbor algorithm is used to find the nearest node $\boldsymbol{q}^a_{\text{near}}$ in the tree, and we then extend the tree from $\boldsymbol{q}^a_{\text{near}}$ to $\boldsymbol{q}_{\text{rand}}$. The last configuration obtained from the extension step is denoted as $\boldsymbol{q}^a_{\text{reached}}$. Next, we extend the tree from $\boldsymbol{q}^a_{\text{reached}}$ to one of the goal configurations, $\boldsymbol{q}_{g,k}$, chosen to be the one nearest to $\boldsymbol{q}^a_{\text{reached}}$. The last configuration obtained from the extension step is denoted as $\boldsymbol{q}^b_{\text{reached}}$. If $\boldsymbol{q}^b_{\text{reached}}$ is equal to $\boldsymbol{q}_{g,k}$, we stop the iteration, and compute the path from the root node $\boldsymbol{q}_0$ to $\boldsymbol{q}_{g,k}$. Otherwise, we continue with the next iteration until the goal is reached or the maximum number of iteration is exceeded.

In the extension step, we iteratively move from $\boldsymbol{q}^a_{\text{near}}$ to $\boldsymbol{q}_{\text{rand}}$ with a step size Δq_{step}, project the resulting configuration to the manifold, and check for collision. The extension stops when the projection fails or it is in collision.

GAN sampling

As reported in [35], the projection steps dominate the computation time with more than 95% of the total time. Instead of uniform sampling, we propose to use the GAN

Algorithm 5 Constrained RRT with goal sampling

INPUT: $\boldsymbol{q}_0, \boldsymbol{x}_g$
OUTPUT: the path $\{\boldsymbol{q}_i\}|_{i=0}^{T}$

$G.init(\boldsymbol{q}_0)$
$\{\boldsymbol{q}_{g,i}\}|_{i=0}^{K} \leftarrow$ SampleGoal($\boldsymbol{x}_g$)
while $n <$ max_iter **do**
  $\boldsymbol{q}_{\text{rand}} \leftarrow$ SampleConfig()
  $\boldsymbol{q}_{\text{near}} \leftarrow$ NearestNeighbor($G, \boldsymbol{q}_{\text{rand}}$)
  $\boldsymbol{q}^a_{\text{reached}} \leftarrow$ ConstrainedExtend($G, \boldsymbol{q}_{\text{near}}, \boldsymbol{q}_{\text{rand}}$)
  $\boldsymbol{q}_{g,k} \leftarrow$ NearestNeighbor($\{\boldsymbol{q}_{g,i}\}|_{i=0}^{K}, \boldsymbol{q}^a_{\text{reached}}$)
  $\boldsymbol{q}^b_{\text{reached}} \leftarrow$ ConstrainedExtend($G, \boldsymbol{q}^a_{\text{reached}}, \boldsymbol{q}_{g,k}$)
  if $\boldsymbol{q}^b_{\text{reached}} = \boldsymbol{q}_{g,k}$ **then**
    P $\leftarrow$ ExtractPath($T, \boldsymbol{q}_0, \boldsymbol{q}_{g,k}$)
    return P
  end if
end while

framework to generate the samples. This will give us samples that are already quite close to the manifold and hence reduce the computation time significantly.

To generate samples from the GAN framework, we first determine the task space region of interest, i.e., a box that covers the reachable points of the robot's end effector. We sample points inside this box and use it as the target for the generator. Together with the Gaussian noise, we can then obtain a set of configurations that are near to the target and require fewer projection steps. As the generator consists of N_{net} neural networks, we choose one out of the N_{net} configurations randomly as the output of the sampler.

6.4 Experimental Results

We implemented the proposed method on two systems: the 7-DoF Franka Emika and 28-DoF Talos. The dataset is created by uniformly sampling random configurations and projecting them to the constraints manifold. A data point corresponds to a configuration that satisfies the specified constraints. We use $N = 25000$ samples to train GAN for both Franka Emika and Talos. The training time takes under 15 minutes, which is quite fast due to the additional cost functions in Section III.A that help the convergence of the GAN training. The generator consists of $N_{\text{net}} = 10$ neural networks with 2 hidden layers, each has 200 nodes, while the discriminator is a neural network with 2 hidden layers (each has 20 nodes for Franka Emika and 40 nodes for Talos). N_{net} is determined by training the generator several times with different numbers of neural networks and choose the one with the best performance on the motion planning task. ReLu is used as the activation function. We train the networks using Stochastic Gradient Descent (SGD). All experiments are run on a processor Intel i7-8750H CPU @ 2.20GHz $\times$ 12.

Table 6.1 – Comparing Projection and IK initialized by uniform sampling vs GAN sampling. The asterisk signifies that the corresponding values are computed only for the successful trials.

Robot	Task	Sampling	Success	$T_{\text{ave}}(ms)$	$T^*_{\text{ave}}(ms)$	Opt. Steps*
Franka Emika	Projection	Uniform	98.6	4.2 ± 6.8	3.6 ± 3.6	6.5 ± 6.2
		GAN	100.0	1.0 ± 0.2	1.0 ± 0.2	1.9 ± 0.4
	IK	Random	76.4	29.3 ± 43.5	8.8 ± 7.5	7.9 ± 5.4
		GAN	88.6	12.5 ± 30.2	2.2 ± 2.2	2.9 ± 1.7
Talos	Projection	Uniform	84.4	20.5 ± 25.3	10.5 ± 9.8	8.7 ± 7.3
		GAN	100.0	2.1 ± 1.2	2.1 ± 1.2	2.1 ± 1.0
	IK	Uniform	82.6	28.7 ± 31.1	15.7 ± 13.0	10.4 ± 7.8
		GAN	100.0	2.8 ± 0.7	2.8 ± 0.7	2.5 ± 0.6

Table 6.2 – Comparing constrained RRT using uniform sampling vs GAN sampling.

Robot	Sampling	Success	$T_{\text{ave}}(s)$	Opt. Steps	E
Franka Emika	Random	100.0	1.44 ± 1.23	2065.7 ± 1763.2	116.5 ± 93.6
	GAN	99.0	0.74 ± 0.66	902.5 ± 796.5	59.7 ± 60.7
	Hybrid	100.0	0.90 ± 0.77	1200.3 ± 1036.8	68.1 ± 56.6
Talos (Task 1)	Uniform	100.0	1.20 ± 0.99	464.4 ± 390.7	16.2 ± 12.4
	GAN	100.0	0.28 ± 0.13	74.0 ± 34.3	10.2 ± 7.3
	Hybrid	100.0	0.43 ± 0.25	131.4 ± 80.0	13.5 ± 10.0
Talos (Task 2)	Uniform	100.0	0.92 ± 0.82	327.8 ± 306.7	13.9 ± 13.1
	GAN	100.0	0.60 ± 0.35	127.0 ± 74.9	36.7 ± 29.4
	Hybrid	100.0	0.66 ± 0.44	182.3 ± 130.0	25.9 ± 19.5
Talos (Task 3)	Uniform	100.0	3.94 ± 3.63	1154.0 ± 1083.2	98.7 ± 72.4
	GAN	100.0	1.05 ± 1.43	179.0 ± 197.0	52.1 ± 158.6
	Hybrid	100.0	1.11 ± 0.94	228.7 ± 203.0	40.8 ± 51.1

6.4.1 Projection and Inverse Kinematics

As described in Section 6.3.2, the projection and IK are formulated as optimization problems by defining several cost functions based on the desired constraints. The optimization problem is solved using Gauss-Newton algorithm. μ and $\bar{\mu}$ are set to 10^{-4} and 10^{-6}. We compare initializing the projection and IK by GAN sampling against uniformly sampling random configurations within the joint limits, which we denote as *uniform* sampling. We set a certain threshold for each cost function, and the optimization is run until all the residuals are below the thresholds.

Franka Emika: The robot configuration consists of 7 joint angles. The main cost function for IK consists of 3 terms: a) joint limit, b) EE orientation constraint, and c) EE position. The orientation is constrained such that the gripper is always in the horizontal position. We also add a secondary cost function, i.e., a posture cost that regularizes around a nominal configuration. The projection has the same set of cost functions as IK, except for the EE position.

Talos: The robot configuration consists of 28 joint angles (7 for each arm, 6 for each leg, and 2 for the torso) and 6 numbers for the base pose. The main cost function for IK consists of the following terms: a) joint limit, b) static stability, c) the feet pose, and d) the right-hand position. The secondary cost function is defined as the posture cost around the nominal configuration, which is chosen to be the initial configuration given to the IK solver except for the left arm (which is regularized around a default posture). The EE is set to be at the right hand. The feet are constrained to remain at the same location. The task is to reach the desired location of the right hand, while respecting the constraints. For the projection, we omit the cost on the EE position.

We evaluate the methods with $N = 500$ tasks, and the result is shown in Table 6.1. $\boldsymbol{T}_{\text{ave}}$ is the average computation time when considering all tasks, while $\boldsymbol{T}^*_{\text{ave}}$ and Opt. Steps* denote the average computation time and the number of optimization steps of only the successful tasks. We can see that using GAN speeds up both projection and IK computation significantly, around 2-5 times faster than uniform sampling, even when considering only the successful results. GAN samples only require around 2-4 optimization steps to achieve convergence. Uniform sampling also has a lower success rate, as can be expected for a nonlinear optimization problem (starting far from the optimal solution reduces the success rate). When the optimization cannot find the solution, it continues optimizing until reaching the maximum iteration, hence spending high computational time (namely, $\boldsymbol{T}_{\text{ave}}$ is higher than $\boldsymbol{T}^*_{\text{ave}}$). In practice, $\boldsymbol{T}_{\text{ave}}$ is the one we actually observe, since there is no way to avoid bad random samples.

Besides the quantitative results shown in Table 6.1 and 6.2, we observe that GAN produces configurations that are still close to the manifold even when the target is infeasible, i.e., too far from the arm reach (Fig. 6.1b). Additionally, we can also extend the target variables to include the left hand and both feet, so that we can do approximate IK for all the four limbs simultaneously using GAN (Fig. 6.1c). We refer to the supplementary video for better visualization of the infeasible target IK and the multi-limbs IK.

6.4.2 Motion Planning

To use the GAN sampling in a sampling-based motion planner, the sampler must have good coverage of the distribution, especially when it is multimodal. We have shown in Fig. 6.3 that using the ensemble of neural networks help GAN to cover a complex

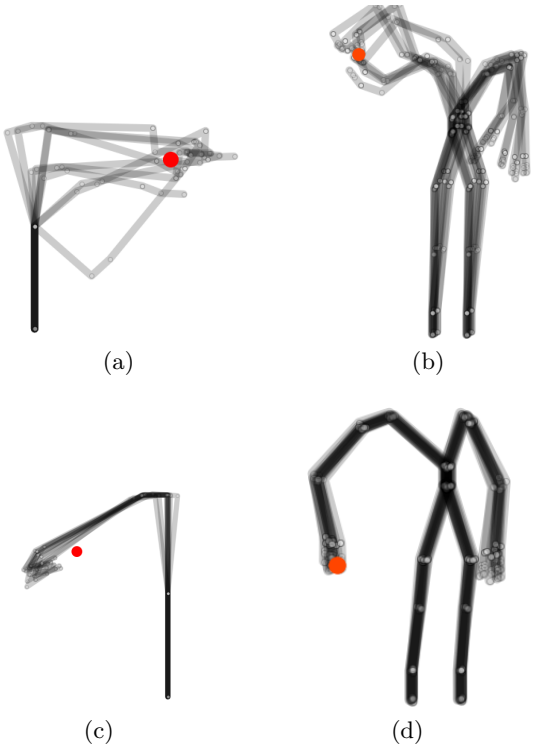
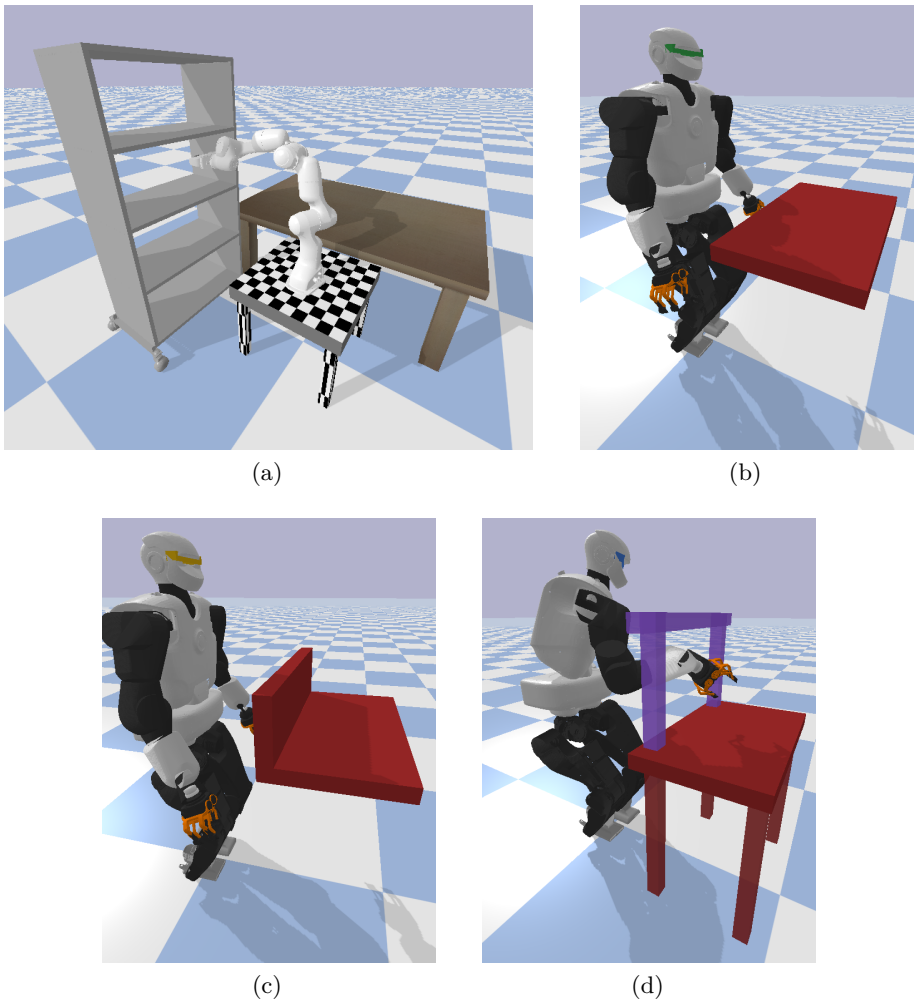


Figure 6.4 – Samples generated by GAN for Franka Emika (a) and Talos (b) using the proposed GAN framework. The samples correspond to the desired end effector positions as shown in red. GAN manages to generate multimodal configurations with large variance. In contrast, (c) and (d) shows the samples generated by the same framework but when the discriminator is omitted. We see here that the ensemble of networks converges to only one mode with very low variance both for Franka Emika (c) and Talos (d).

distribution. In addition to this, the discriminator also helps to increase the coverage. Indeed, without providing the dataset to the discriminator and train it together with the generator, even the ensemble of neural networks cannot have good coverage of the distributions, and often they converge to one mode only. Fig. 6.4a-b shows samples generated by GAN for Franka Emika and Talos by giving the desired EE position (shown in red). For both robots, the generated samples belong to multiple modes with good variance. In contrast, Fig. 6.4c-d show samples generated by GAN while removing the discriminator and training using only the additional cost functions in Section III.A (to be strict in terminology, this makes it no longer an adversarial network, but this is done to demonstrate the necessity of the discriminator). Without the discriminator, the samples do not have large variance and converge to only one mode, despite using the ensemble of networks.

We then use GAN sampling in constrained motion planning for both Franka Emika and Talos, as described in Algorithm 5. For Franka Emika, the task is to move the



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

Figure 6.5 – Constrained motion planning tasks for Franka Emika (a) and Talos ((b), (c) and (d)).

end effector from above the table to one of the shelves, as shown in Fig. 6.5a, while maintaining the gripper in the horizontal position. For Talos, we consider three different environments in Fig. 6.5b-d in increasingly more difficult order, similar to the ones used in [35]. The task is to move from a random initial configuration above a table to reach a specified target position with the right hand below the table while satisfying the static stability and joint limits. For each environment, we run $N = 100$ random tasks. We compute $K = 10$ goal configuration for each task. Each task is run until it is successful or it reaches the maximum number of extension steps, which is set to be 500. In the case of failure, the planning is repeated until a maximum of two times and the total time is taken.

We compare three different sampling methods: *uniform*, *GAN*, and *hybrid* sampling. Hybrid sampling combines the uniform and GAN sampling. It outputs GAN samples

with a probability of p and uniform samples with a probability of $1 - p$. By including the uniform sampling, we can keep the probabilistic completeness guarantee of the planner. From the experiment, we observe that a value of $p = 0.8$ performs the best. An adaptive value is also possible, i.e., starting with $p = 1$ and decreasing it as the number of extensions grows, such that it converges to uniform sampling when the planner still cannot find any solution after a long time.

The result can be seen in Table 6.2. $\boldsymbol{T}_{\text{ave}}$, Opt. Steps, and $\boldsymbol{E}$ denote the average planning time, the number of optimization steps, and the number of extension steps in planning (*ConstrainedExtend* in Algorithm 5) for one task.

For both Franka Emika and Talos, GAN results in a significant reduction in the computation time, around 2-4 times faster than the uniform sampling. This gain is mainly due to the fewer optimization steps necessary to project the resulting configurations, as discussed in the previous section. The high success rate of GAN sampling also indicates that it manages to cover a good proportion of the configuration space, at least for the tasks that we consider here.

On the other hand, the comparison between GAN and hybrid sampling strategy is interesting. In most cases, GAN is faster than hybrid sampling. However, hybrid sampling sometimes requires fewer extension steps than GAN. We also observe that GAN sometimes fails a task, while hybrid sampling is successful, such as the case in the Franka Emika experiment. This shows that hybrid sampling can explore the distribution more effectively than GAN in these tasks. Its overall computation time is still higher than GAN, though, because it requires more optimization steps due to the uniform sampling components.

6.4.3 Discussion

From the experiment results, we showed that GAN can generate good quality samples close to the desired manifold. In addition, we can conclude from the motion planning results that it has good coverage over the distribution. There is no guarantee, however, that the distribution is perfectly covered, so in some rare cases the motion planning may fail to find a feasible solution, but using a hybrid sampling strategy recovers the completeness guarantee.

We managed to get good performance even with quite a basic GAN structure. Note that the GAN framework was trained without considering collision, unlike in the example of the 2-link robot in Section III.A. This means that the resulting sampler does not depend on the environment, and it works directly in any environment even with moving obstacles. We also showed that the framework is easily adapted to different robots by the experiments on the Franka Emika and Talos robot. Given any new robot and its constraints, we only need to formulate the cost functions corresponding to the constraints,

generate the dataset, and train the GAN quickly.

The stability criteria that we used here is static stability. In [133] and [138], GAN has been shown to work well for problems with dynamics, so it is possible to extend our approach to generate more dynamical motions by including dynamic stability criteria such as Zero Moment Point (ZMP) [39] or Contact Wrench Cone Margin [139].

Since GAN is a very flexible tool, there are a lot of potential improvements. As in [130], by generating many planning data in a given environment, we can condition GAN on the initial and goal configurations to obtain samples that are relevant for the task. GAN also works well with high dimensional inputs such as images, so it would be possible to train with the environment representation (e.g., voxel data or heightmap) to generate samples that avoid collisions in different environments. Furthermore, since we can condition GAN on the desired end effector pose, we could combine the task-space biasing strategy [140, 141] with the GAN sampler.

In this chapter, we used an ensemble of networks as the generator to cover the multimodal distribution. There are also other methods that attempt to handle this mode collapse issue, e.g., using Wasserstein loss [142] or unrolled GANs [143], which could be investigated in further work.

6.5 Conclusion

We have presented a GAN framework to learn the distribution of valid robot configurations under various constraints. The method was then used for inverse kinematics and sampling-based constrained motion planning to speed up computational time. We validated the proposed method on two simulated platforms: 7-DoF Franka Emika manipulator and 28-DoF Talos humanoid robot. In all settings, the proposed method managed to reduce the computational time significantly (up to 5 times faster) with a higher success rate. The method is very general and easily applicable to other robotic systems.

7 Tensor Train for Global Optimization Problems in Robotics

In this chapter, we consider techniques from multilinear algebra, i.e., tensor methods, to approximate the unnormalized PDF induced by the cost function. By using an efficient algorithm, we can approximate a high-dimensional function using a compact representation and with efficient computation. Unlike in previous chapters where the database construction, data representation, and the learning methods are separate, our proposed method provides a unified way to do all those tasks by performing tensor operations. The proposed method can generate approximate solutions for a given task in the order of milliseconds. In many problems that we consider here (including inverse kinematics and motion planning), the approximate solutions have low costs and come from multiple modes. During the training, our method does not require any gradient information from the cost function, and hence is quite robust at finding the global optima or at least good local optima.

This chapter is a result of the joint work with S. Shetty and T. Löw. My part was to formulate the optimization problems for evaluating the method and to draw the connection with other existing works, e.g., Variational Inference for multimodal trajectory optimization and database approaches for building a memory of motion. S. Shetty and T. Löw worked on the tensor formulation and performed the experiments.

Publication Note

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

- S. Shetty, T. S. Lembono, T. Löw, and S. Calinon, "Tensor train for global optimization problems in robotics," *Submitted article under review*, 2022, https://sites.google.com/view/ttgo/home

Supplementary Material and Source Codes

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

7.1 Introduction

In the first part of this thesis (Chapter 3 and 4), we have discussed several approaches to build a memory of motion based on function approximation and a database of good solutions. While such approaches are easy to formulate, they suffer from two issues. Firstly, 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. We have presented several ideas that address these two issues, e.g., building the dataset in two stages (using a global planner and then a local optimizer), and using a mixture model (BGMR) to handle multimodal data. In this chapter, we propose another approach based on tensor methods we call Tensor Train for Global Optimization (TTGO). Unlike the database approaches, TTGO does not require an explicit dataset built by another method, and it handles multimodal problems naturally.

The proposed approach combines several different techniques, mainly: Tensor Train (TT) decomposition for function approximation [144], sampling from TT model [145], and numerical optimization using cross approximation technique [146]. 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 [144], a technique from multilinear algebra. TT models, as shown recently by [145], 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.

A cost function is usually 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., [66, 147, 64]) 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 parameters. 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 (the gradient is not required), 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 allows 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.

The chapter layout is as follows. In Section 7.2, we provide a literature survey on warm starting numerical optimization, multimodal optimization, and tensor methods. Section 7.3 explains the necessary background on tensor train modeling that is used in this chapter. Then, in Section 7.4, we describe the TTGO method proposed in this chapter.

Section 7.5 presents the evaluation of our algorithm. We first test it on benchmark functions for numerical optimization. To showcase the relevance of our approach for robotics, we then apply it to inverse kinematics and motion planning problems with manipulators. In Section 7.6 and 7.7, we conclude the chapter 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.

7.2 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.

7.2.1 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 [3], STOMP [5], TrajOpt [148], GPMP [149]). Inverse kinematics for high dimensional robots is usually formulated as nonlinear least squares optimization [2] or Quadratic Programming (QP) [27]. In control, optimization-based controllers take the form of Task Space Inverse Dynamics (TSID) controller formulated as QP problem [6], or finite horizon optimal control [44, 48]. 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 [8]. 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.

7.2.2 Predicting good initialization

A majority of works that attempt to predict good initialization rely on a database approach, often called trajectory library [9] or memory of motion [70]. The idea is to

first build a database of precomputed solutions offline. This database can be constructed from expert demonstrations [102], using the optimization solver itself [84], or using the combination of a global planner and the optimization solver [45]. 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 [45, 93], 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 [8] or Mixture Density Networks [150], 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.

7.2.3 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 [66, 147, 64]. 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 [66]. 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. [66] 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 [66] attempts to obtain multiple solutions from finitely many modes, [147] 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 [66]) or uncountably many solutions (as in [147]), as will be shown in Section 7.5.

7.2.4 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 [151, 152] for general surveys, and see [153, 154] 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 [155, 156, 157].

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 [158]. TT decomposition has been used to solve problems involving high-dimensional integration of multivariate functions in [159, 155]. [145] 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) [160, 161, 162, 163].

In [164, 146], 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 work; 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.

7.3 Background

In this section, we first describe what a tensor is (Section 7.3.1) and how a multivariate function can be approximated using a tensor (Section 7.3.2). 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 7.3.3 and 7.3.4) and proceed with the extension for the higher-order tensor (Section 7.3.5 and 7.3.6). When the target function is an unnormalized probability density function (PDF), we can construct a probability distribution from the tensor model (Section 7.3.7), allowing us to sample (Section 7.3.8-7.3.9) and condition (Section 7.3.10) on some of the dimensions.

7.3.1 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.

7.3.2 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.

7.3.3 Separation of Variables using Matrix Factorization

Consider a continuous 2D function

$$P(x_1, x_2)\colon \Omega_{\boldsymbol{x}} \subset \mathbb{R}^2 \to \mathbb{R}. \tag{7.1}$$

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 $(x_1^1, \dots, x_1^{n_1})$ and $(x_2^1, \dots, x_2^{n_2})$ be the corresponding discretization points of the two intervals. The discretization set is then given by $\mathcal{X} = \{\boldsymbol{x} = (x_1^{i_1}, x_2^{i_2})\colon i_k \in \{1, \dots, n_k\}, k \in \{1, 2\}\}$ and corresponding index set is $\mathcal{I}_{\mathcal{X}} = \{\boldsymbol{i} = (i_1, i_2)\colon i_k \in \{1, \dots, n_k\}, k \in \{1, 2\}\}$. The corresponding discrete analogue is then given by the matrix defined as

$$\boldsymbol{P}_{i_1,i_2} = P(x_1^{i_1}, x_2^{i_2}), \quad \forall (i_1, i_2) \in \mathcal{I}_{\mathcal{X}}. \tag{7.2}$$

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

$$\boldsymbol{P}_{i_1,i_2} \approx \boldsymbol{P}^1_{i_1,:}\, \boldsymbol{P}^2_{:,i_2}. \tag{7.3}$$

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 (7.3) 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

$$\begin{aligned} \boldsymbol{p}^1(x_1) &= \frac{x_1 - x_1^{i_1}}{x_1^{i_1+1} - x_1^{i_1}} \boldsymbol{P}^1_{i_1+1,:} + \frac{x_1^{i_1+1} - x_1}{x_1^{i_1+1} - x_1^{i_1}} \boldsymbol{P}^1_{i_1,:}, \\ \boldsymbol{p}^2(x_2) &= \frac{x_2 - x_2^{i_2}}{x_2^{i_2+1} - x_2^{i_2}} \boldsymbol{P}^1_{:,i_2+1} + \frac{x_2^{i_2+1} - x_2}{x_2^{i_2+1} - x_2^{i_2}} \boldsymbol{P}^2_{:,i_2}, \end{aligned} \tag{7.4}$$

where $x_k^{i_k} \le x_k \le x_k^{i_k+1}$, $\boldsymbol{p}^1(x_1)\colon \Omega_{\boldsymbol{x}_1} \subset \mathbb{R} \to \mathbb{R}^{1 \times r}$ 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,

$$\begin{aligned} P(x_1, x_2) &\approx \boldsymbol{p}^1(x_1)\boldsymbol{p}^2(x_2), \quad \forall (x_1, x_2) \in \Omega_{\boldsymbol{x}} \\ &= \sum_{j=1}^{r} \boldsymbol{p}_j^1(x_1)\boldsymbol{p}_j^2(x_2). \end{aligned} \tag{7.5}$$

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) [159, 155]. 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 [145]) which will be discussed in Section 7.3.8.

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 (7.5) 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.

7.3.4 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

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

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 7.1).

Now suppose we have a noisy version of the matrix $\boldsymbol{P} = \tilde{\boldsymbol{P}} + \boldsymbol{E}$ with $\|\boldsymbol{E}\| < \epsilon$ and $\tilde{\boldsymbol{P}}$ is of low rank. For a sufficiently small ϵ, $\operatorname{rank}(\tilde{\boldsymbol{P}}) = r$ so that the matrix $\boldsymbol{P}$ can be approximated with a lower rank r (i.e., $\operatorname{rank}(\boldsymbol{P}) \approx r$). Then, the choice of the submatrix $\boldsymbol{P}_{\boldsymbol{i}_1,\boldsymbol{i}_2}$ (or index vectors $\boldsymbol{i}_1, \boldsymbol{i}_2$) for the cross approximation requires several considerations. The maximum volume principle can be used in choosing the submatrix which states that the submatrix with maximum absolute value of the determinant is the optimal choice. If $\boldsymbol{P}_{\boldsymbol{i}_1^*,\boldsymbol{i}_2^*}$ is chosen to have the maximum volume, then by skeleton decomposition we have an approximation of the matrix $\boldsymbol{P}$ given by $\hat{\boldsymbol{P}} = \boldsymbol{P}_{:,\boldsymbol{i}_2^*} \boldsymbol{P}_{\boldsymbol{i}_1^*,\boldsymbol{i}_2^*}^{-1} \boldsymbol{P}_{\boldsymbol{i}_1^*,:}$. 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 [165] 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.
   (a) 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 $\hat{\boldsymbol{P}} \approx \boldsymbol{P}_{:,\boldsymbol{i}_2^*}\boldsymbol{P}_{\boldsymbol{i}_1^*,\boldsymbol{i}_2^*}^{-1}\boldsymbol{P}_{\boldsymbol{i}_1^*,:}$.

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 [165]. 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.
- Cross approximation algorithms directly find the factors without computing and storing the whole matrix.

7.3.5 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 7.2, 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:

$$\boldsymbol{\mathcal{P}}_{\boldsymbol{i}} = \boldsymbol{\mathcal{P}}^1_{:,i_1,:}\boldsymbol{\mathcal{P}}^2_{:,i_2,:} \cdots \boldsymbol{\mathcal{P}}^d_{:,i_d,:}, \tag{7.6}$$

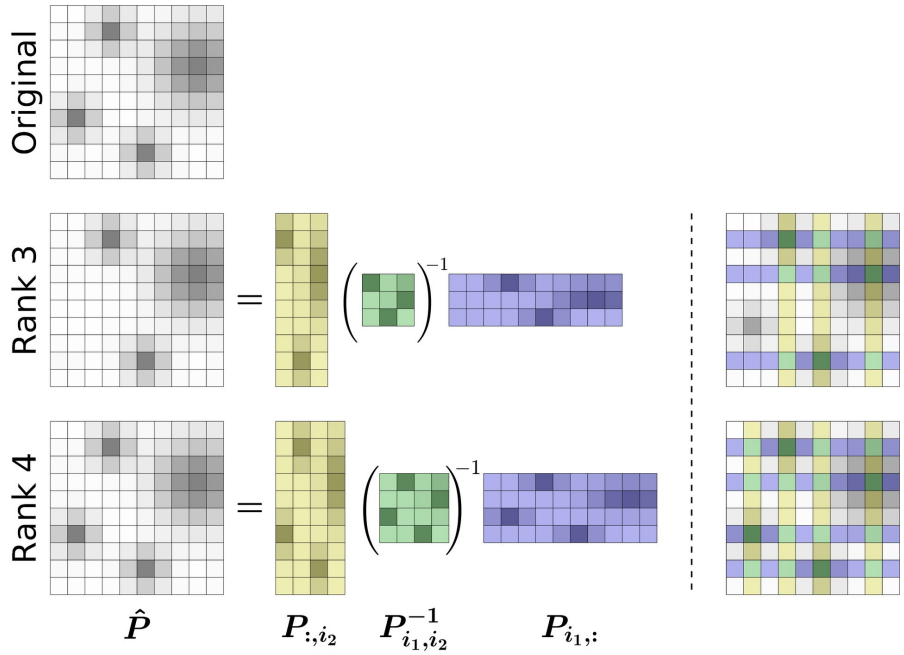


Figure 7.1 – For a given matrix $\boldsymbol{P}$ (top-left), suppose we know r independent columns indexed by $\boldsymbol{i}_2 = (i_{2,1}, \ldots, i_{2,r})$, i.e., $\boldsymbol{P}_{:,\boldsymbol{i}_2} \in \mathbb{R}^{n_1 \times r}$ and r independent rows indexed by $\boldsymbol{i}_1 = (i_{1,1}, \ldots, i_{1,r})$, i.e., $\boldsymbol{P}_{\boldsymbol{i}_1,:} \in \mathbb{R}^{r \times n_2}$, with their intersection $\boldsymbol{P}_{\boldsymbol{i}_1,\boldsymbol{i}_2} \in \mathbb{R}^{r \times r}$ being nonsingular. Then, by skeleton decomposition we have $\hat{\boldsymbol{P}} = \boldsymbol{P}_{:,\boldsymbol{i}_2} \boldsymbol{P}^{-1}_{\boldsymbol{i}_1,\boldsymbol{i}_2} \boldsymbol{P}_{\boldsymbol{i}_1,:}$. If $\operatorname{rank}(\boldsymbol{P}) = r$, then $\hat{\boldsymbol{P}} = \boldsymbol{P}$ (bottom row). For $r < \operatorname{rank}(\boldsymbol{P})$ we obtain a quasi-optimal approximation, $\hat{\boldsymbol{P}} \approx \boldsymbol{P}$ (middle row). The right figures show the rows and columns selected from the original matrix by the cross approximation algorithm to find the skeleton decomposition.

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 (7.6) [144].

Similarly to (7.5), we can also obtain a continuous approximation of the function P as

$$P(x_1, \ldots, x_d) \approx \boldsymbol{P}^1(x_1) \cdots \boldsymbol{P}^d(x_d), \tag{7.7}$$

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 (7.4) (see Appendix B.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

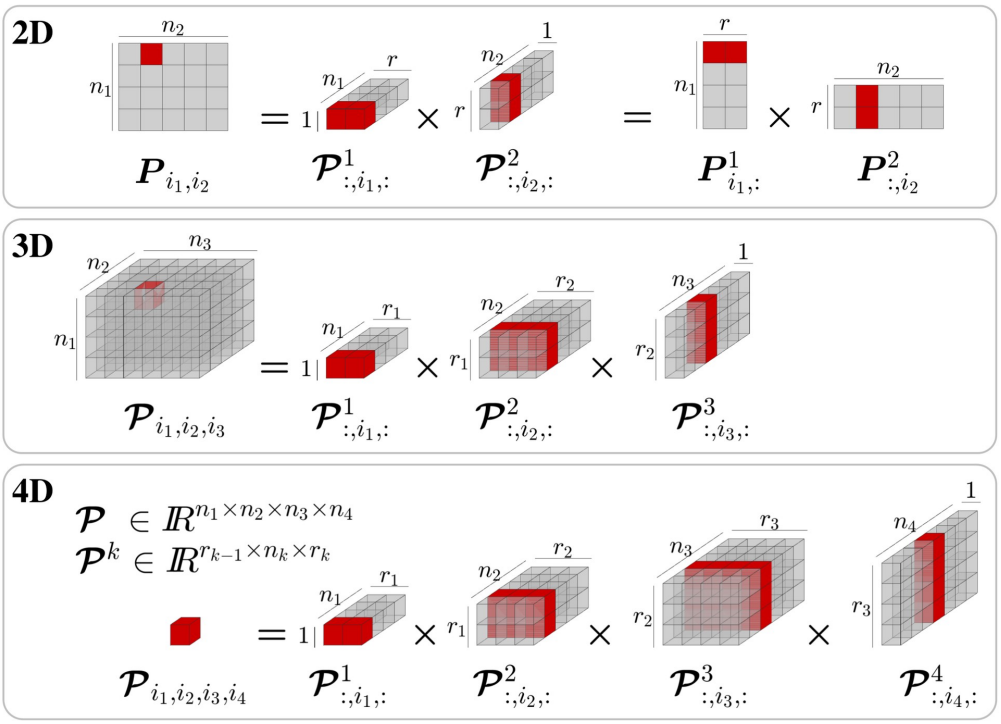


Figure 7.2 – TT decomposition is an extension of matrix decomposition techniques to higher dimensional arrays. With a matrix decomposition, we can access an element of the original matrix by multiplying appropriate rows or columns of the factors. Similarly, an element of a tensor in TT format can be 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.

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 7.3.3, 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).

7.3.6 TT-Cross

The popular methods to find the TT decomposition of a tensor are TT-SVD [144], TT-DMRG [159], and TT-cross [166]. 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 7.3.4 for obtaining the TT decomposition of a tensor. We refer the readers to Appendix B.2 for more detail on how the matrix cross-approximation algorithm described in Section 7.3.4 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 tensor $\boldsymbol{\mathcal{P}}$ at various query points, i.e., the fibers of the tensor $\boldsymbol{\mathcal{P}}$.

7.3.7 TT Distribution

Suppose we use the tensor $\boldsymbol{\mathcal{P}}$ 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,

$$\Pr(\boldsymbol{x}) = \frac{|\boldsymbol{\mathcal{P}}_{\boldsymbol{x}}|}{Z}, \quad \boldsymbol{x} \in \Omega_{\boldsymbol{x}}, \tag{7.8}$$

where Z is the corresponding normalization constant.[1] 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 [145].

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

7.3.8 Sampling from TT distribution

Consider a probability distribution given by (7.8). 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

$$\Pr(x_1, \dots, x_d) = \Pr_1(x_1)\Pr_2(x_2|x_1)\cdots\Pr_d(x_d|x_1, \dots, x_{d-1}), \tag{7.9}$$

where

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

is the conditional distribution defined using the marginals

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

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 $k = 1, \dots, d$ **do**
  $x_k \approx \mathtt{Pr}_k(x_k|x_1, \dots, x_{k-1})$
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 $\boldsymbol{\mathcal{P}}$ 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 TT model be given by the cores $(\boldsymbol{\mathcal{P}}^1, \dots, \boldsymbol{\mathcal{P}}^d)$, then we have

$$\begin{aligned}
\sigma_k(x_1, \dots, x_k) &= \sum_{x_{k+1}} \cdots \sum_{x_d} \Pr(\boldsymbol{x}), \\
&\approx \sum_{x_{k+1}} \cdots \sum_{x_d} |\boldsymbol{\mathcal{P}}_{\boldsymbol{x}}|, \\
&= \sum_{x_{k+1}} \cdots \sum_{x_d} |\boldsymbol{\mathcal{P}}^1_{:,x_1,:}| \cdots |\boldsymbol{\mathcal{P}}^k_{:,x_k,:}||\boldsymbol{\mathcal{P}}^{k+1}_{:,x_{k+1},:}| \cdots |\boldsymbol{\mathcal{P}}^d_{:,x_d,:}|, \\
&= |\boldsymbol{\mathcal{P}}^1_{:,x_1,:}| \cdots |\boldsymbol{\mathcal{P}}^k_{:,x_k,:}| \Big(\sum_{x_{k+1}} |\boldsymbol{\mathcal{P}}^{k+1}_{:,x_{k+1},:}| \Big) \cdots \Big(\sum_{x_d} |\boldsymbol{\mathcal{P}}^d_{:,x_d,:}| \Big),
\end{aligned} \tag{7.10}$$

where $\sum_{x_k} |\boldsymbol{\mathcal{P}}^k_{:,x_k,:}|$ is the summation of all the matrix slices (absolute values) of the third-

Algorithm 6 TT-CD Sampling with Sample Prioritization

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

Ensure: N $\alpha-$prioritized samples $\{(x_1^l, \ldots, x_d^l)\}_{l=1}^N$ 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 $l = 1, \ldots, N$ **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

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 [145], to efficiently get the samples from Pr.

7.3.9 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 (7.3.7). $\alpha = 0$ 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 6. The prioritized sampling can be relaxed by setting $\alpha = 0$ in the algorithm, resulting in the standard sampling procedure described in Section 7.3.8. Note that the algorithm allows parallel implementation to quickly generate a large number of samples.

7.3.10 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 (7.8).

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

$$(\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_t})^k = \begin{cases} \boldsymbol{\mathcal{P}}^k_{:,\boldsymbol{x}_{t_k},:}, & k \in \{1, \dots, d_1\} \\ \boldsymbol{\mathcal{P}}^k, & k \in \{d_1+1, d_1+d_2\} \end{cases} \tag{7.11}$$

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

$$\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1 = \boldsymbol{x}_t) = \frac{|\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_t}_{\boldsymbol{x}_2}|}{Z_1}, \forall \boldsymbol{x}_2 \in \Omega_{\boldsymbol{x}_2}. \tag{7.12}$$

Given $\boldsymbol{x}_1 = \boldsymbol{x}_t$, we can sample $\boldsymbol{x}_2$ from this distribution using Algorithm 6 with the conditional TT model $\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_1=\boldsymbol{x}_t}$.

7.4 Methodology

7.4.1 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 7.3) 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 chapter, 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 7.2, 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.

7.4.2 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 7.3.7) 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 7.3.6). 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 7.3.8). 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 7.3.9).

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

7.4.3 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 $\boldsymbol{x}_2$:

$$\begin{aligned} \boldsymbol{x}_2^* = &\arg\min_{\boldsymbol{x}_2} C(\boldsymbol{x}_1, \boldsymbol{x}_2) \\ \text{s.t. } &\boldsymbol{x}_1 = \boldsymbol{x}_t, \\ &\boldsymbol{x}_2 \in \Omega_{\boldsymbol{x}_2}. \end{aligned} \tag{7.13}$$

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,

$$\begin{aligned} \boldsymbol{x}_{\boldsymbol{2}}^* = &\arg\max_{\boldsymbol{x}_2} P(\boldsymbol{x}_t, \boldsymbol{x}_2) \\ \text{s.t. } &\boldsymbol{x}_1 = \boldsymbol{x}_t, \\ &\boldsymbol{x}_2 \in \Omega_{\boldsymbol{x}_2}. \end{aligned} \tag{7.14}$$

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 chapter we consider optimization problems to be of type (7.14) 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 (7.13). 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) $\hat{\boldsymbol{x}}_2^*$.

Approximating the unnormalized PDF using TT model

As described in Section 7.3, 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 7.3.6) 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 7.3.

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 (7.8), which requires the normalization constant as per the definition. However, as described in Section 7.3.8, 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 $\boldsymbol{\mathcal{P}}$.

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

Sampling

As described in Section 7.3.8, 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 6. Furthermore, a sampling parameter $\alpha \in (0, 1)$ can be chosen to the adjust the sampling priority (see Section 7.3.9). 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 $\hat{\boldsymbol{x}}_2^*$.

7.4.4 TTGO Algorithm

1. Training Phase (Offline):

   (a) Given:

   - Cost function $C(\boldsymbol{x}_1, \boldsymbol{x}_2)$,

- Rectangular domain $\Omega_{\boldsymbol{x}} = \Omega_{\boldsymbol{x}_1} \times \Omega_{\boldsymbol{x}_2}$

(b) Transform the cost function into an unnormalized PDF $P(\boldsymbol{x}_1, \boldsymbol{x}_2)$.

(c) Discretize the domain Ω_x into $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2$.

(d) Using TT-Cross, construct the TT-Model $\boldsymbol{\mathcal{P}}$ as the discrete analogue of $P(\boldsymbol{x})$ with discretization set $\mathcal{X}$.

2. Inference Phase (Online):

(a) Given: The task-parameter $\boldsymbol{x}_1 = \boldsymbol{x}_t \in \Omega_{\boldsymbol{x}_1}$, the desired number of solutions K.

(b) Construct the conditional TT Model $\boldsymbol{\mathcal{P}}^{\boldsymbol{x}_t}$ from $\boldsymbol{\mathcal{P}}$ (see Section 7.3.10).

(c) 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 6).

(d) Evaluate the cost function at these samples and choose the best-K samples as approximation for the optima $\{\hat{\boldsymbol{x}}_2^{*l}\}_{l=1}^K$.

(e) Fine-tune the approximate solutions using gradient-based approaches on $C(\boldsymbol{x}_t, \boldsymbol{x}_2)$ to obtain the optima $\{\boldsymbol{x}_2^{*l}\}_{l=1}^K$.

7.5 Experiments

In this section, we evaluate the performance of the proposed algorithm with several applications.[2] 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 performance as follows. First, we generate 100 random test cases within the task space. For each test case, we generate N

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

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 7.1-7.3 and will be explained in the corresponding sections.

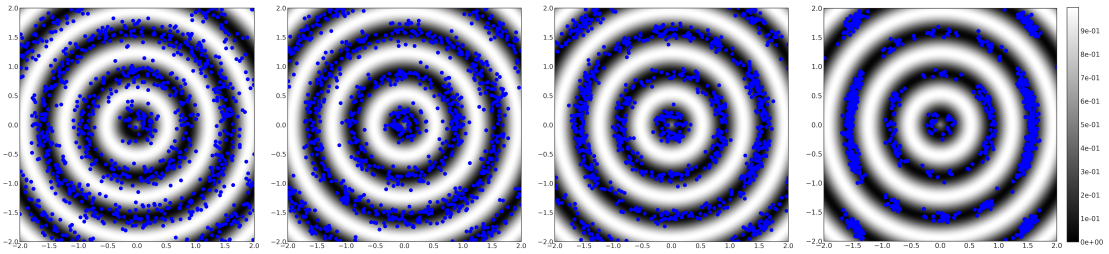
7.5.1 Benchmark functions

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 chapter. 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 7.3- 7.8, 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 7.3-7.7 and the density function in Figure 7.8, 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



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

Figure 7.3 – 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.

get the optima using TTGO with few samples. In fact, by using α close to 1, we could find the global optima with just one sample from the TT distribution.

Sinusoidal Function:

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

where $\boldsymbol{x} = \boldsymbol{x}_2 = (y_1, y_2)$, , $\Omega_{\boldsymbol{x}_2} = [-2, 2]^2$ with no task parameters. For this function, finding the optima is not a difficult problem. However, as the cost function has uncountably many global optima (on the circles separated by one period of the sinusoidal function), we use it to test the approximation power of TT-model and check the multimodality in the TTGO samples. As we can see in Figure 7.3 for $d = 2$, the samples from the TT model mainly come from the modes corresponding to the optima and the nearby region with cost values comparable to the optimal cost. At $\alpha = 0$, we can still observe a few samples in the white area (low density region), and as we increase α, the samples become more concentrated in the dark area, i.e., high-density region.

Rosenbrock Function:

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

where $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$, $\boldsymbol{x}_1 = (a, b)$, $\boldsymbol{x}_2 = (y_1, \ldots, y_{d_2})$, $\Omega_{\boldsymbol{x}_1} = [-1.5, 1.5] \times [50, 150]$, $\Omega_{\boldsymbol{x}_2} =$

$[-2, 2]^{d_2}$. The function is similar to a banana distribution which is quite difficult to approximate. The cost function $C(\boldsymbol{x})$ for a specified (a, b) has a unique global minima at $(a, a^2, \ldots, a, a^2)$. However, if we do not initialize the solution from the parabolic valley area (see Figure 7.4), a gradient-based solver will have difficulty in converging to the global optima quickly. We can see from Figure 7.4 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 7.5 shows how the task parameters $\boldsymbol{x}_1 = (a, b)$ change the shape of the function with respect to $\boldsymbol{x}_2$ and consequently the location of the global optima. After the offline training, we condition our TT model on these task parameters and sample from the conditional distribution $\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1 = (a, b))$. We can see in this figure that TTGO can adapt to the new task parameters easily, as the samples are concentrated around the new global optima.

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

Himmelblau's function:

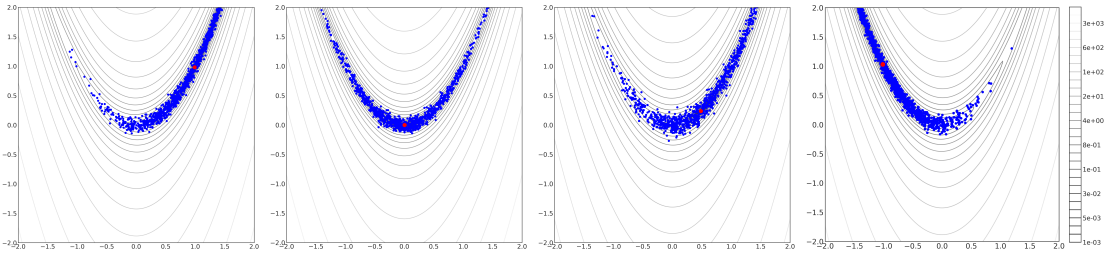
$$C(a, b, y_1, y_2) = (y_1^2 + y_2 - a)^2 + (y_1 + y_2^2 - b)^2$$
$$P(\boldsymbol{x}) = \exp\Big(-C(\boldsymbol{x})^2\Big),$$

where $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$, $\boldsymbol{x}_1 = (a, b)$, $\boldsymbol{x}_2 = (y_1, y_2)$, $P(\boldsymbol{x}) = \exp(-C(\boldsymbol{x}))$, $\Omega_{\boldsymbol{x}_1} = [0, 15]^2$, $\Omega_{\boldsymbol{x}_2} = [-5, 5]^2$. The cost function $C(a, b, y_1, y_2)$ for a given (a, b) has multiple distinct global optima and many local optima. The samples from the TT distribution $\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1 = (a, b))$ are shown in Figure 7.6–7.7 for different choice of task parameters and the prioritized sampling parameters α. We can see that TTGO can generate samples from all of the modes consistently according to the task parameters.

Mixture of Gaussians:

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

We use an unnormalized mixture of Gaussian functions to define the probability function $P(\boldsymbol{x})$ to test our framework for high-dimensional multimodal functions. For verification, we design the mixture components so that we know the global optima *a priori* by carefully



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

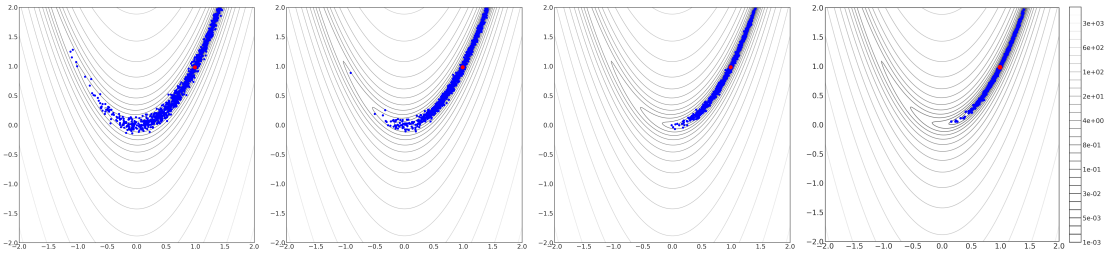
Figure 7.4 – 1000 samples from the conditional TT distribution of a Rosenbrock function for various choices of the task parameters (a, b) and $\alpha = 0$. The function has a unique global optimum at (a, a^2) as shown in red. As the task parameters change, the global optimum moves accordingly, but TTGO is still able to sample from the high-density regions.

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 7.8 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.

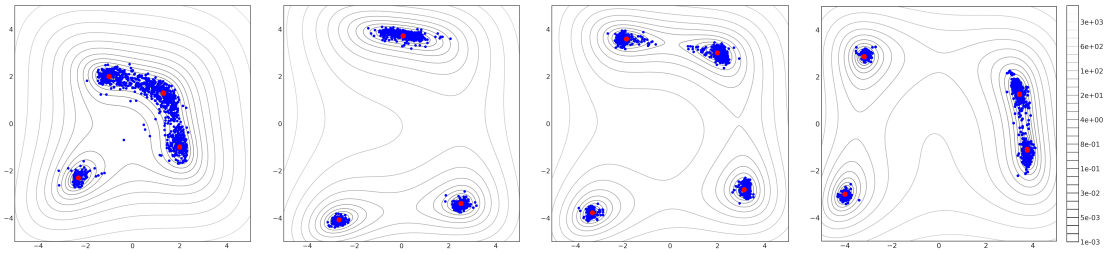
7.5.2 Inverse Kinematics

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.



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

Figure 7.5 – 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).



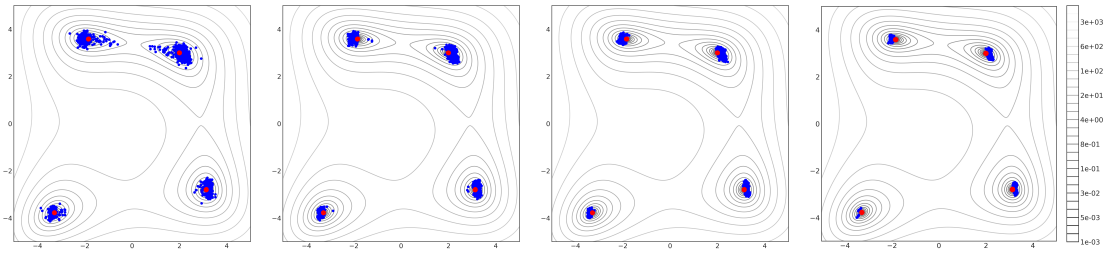
(a) $(a, b) = (13, 5)$ (b) $(a, b) = (7, 11)$ (c) $(a, b) = (3, 3)$ (d) $(a, b) = (3, 14)$

Figure 7.6 – 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.

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 $P(\boldsymbol{x}) = \exp(-C(\boldsymbol{x})^2)$.

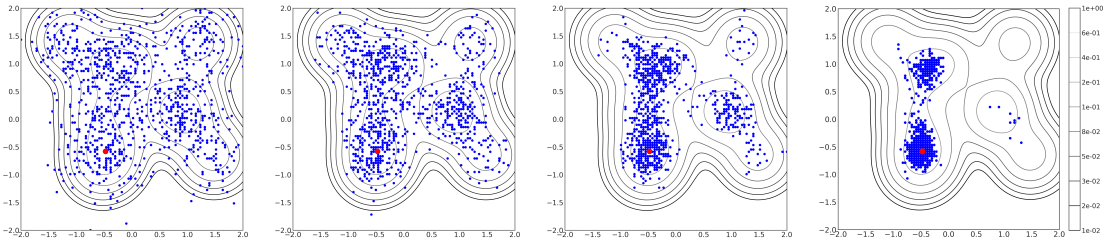
Inverse Kinematics for 6-DoF Robot:

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.



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

Figure 7.7 – 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.



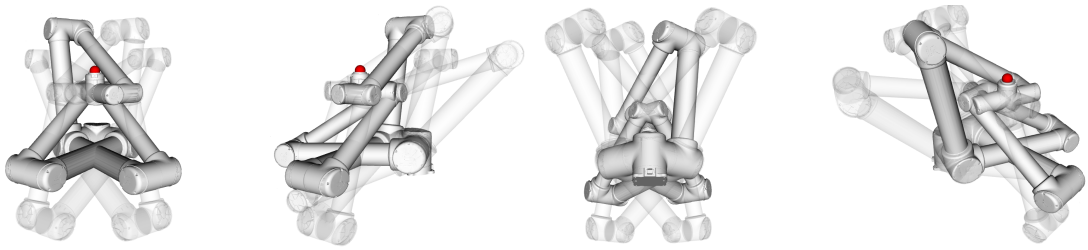
(a) $\alpha = 0$ (b) $\alpha = 0.5$ (c) $\alpha = 0.9$ (d) $\alpha = 0.99$

Figure 7.8 – 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.

We observe that TTGO is able to retrieve most of the 8 IK solutions for a given end effector pose. Figure 7.9 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.

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 [3], 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



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

Figure 7.9 – 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.

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 7.10 and Figure 7.11 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 7.10). As we decrease α to 0.8 and retrieve more samples, we can see that multiple solutions can be obtained easily (Figure 7.11). Note that even without the refinement step, all samples reach the goal closely while being collision-free.

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

Figure 7.12 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 7.1. 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.

7.5.3 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 [30, 57]. 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 B.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

The performance measures for three different applications with the Franka Emika manipulator. We compare the performance of TTGO for warm starting 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 7.1 – 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 7.2 – 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 7.3 – 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

planning formulation can be found in Appendix B.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 $\boldsymbol{\theta}_0$.

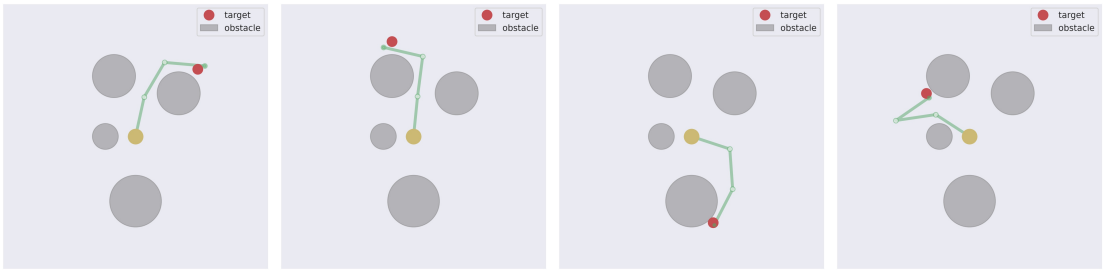
For the target reaching problem, the task parameter is the target location $\boldsymbol{x}_1 = \boldsymbol{p_d}$ and the decision variables $\boldsymbol{x}_2 = (\boldsymbol{\theta}_1, \boldsymbol{w})$. Here, $\boldsymbol{\theta}_1 \in \Omega_{\boldsymbol{\theta}} \subset \mathbb{R}^m$ is the joint angle 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., $(w_{min}^k, w_{max}^k) = (\theta_{min_k}, \theta_{max_k})$.

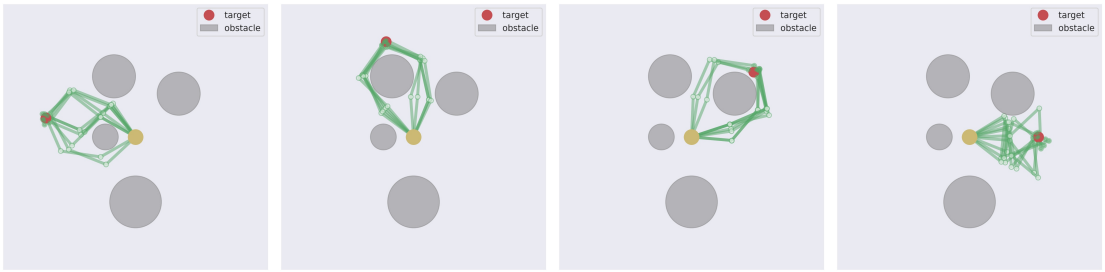
Figure 7.13 shows some examples of a reaching task for a 3-DoF planar manipulator. We can see here that the TTGO samples lead to good solutions, i.e., they avoid collisions while reaching the target quite accurately. In comparison, random sampling initialization often results in poor local optima, where the final solutions still have collisions even after the refinement. Figure 7.14 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 7.16 and 7.17.

The results are presented in Table 7.2 and 7.3. 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



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

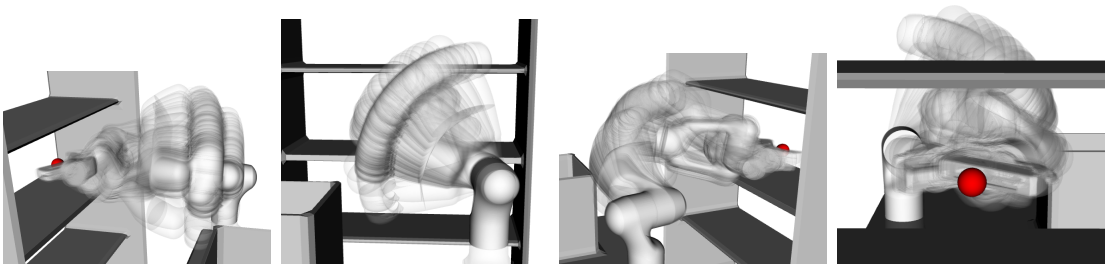
Figure 7.10 – A single sample taken from a conditional TT distribution with $\alpha = 1$ for inverse kinematics of a 3-link planar manipulator in the presence of obstacles (gray spheres). The yellow circle and the green segments depict the base and the links of the robot, respectively. The target end effector positions are shown in red. The samples are very close to the targets and collision-free, even without refinement.



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

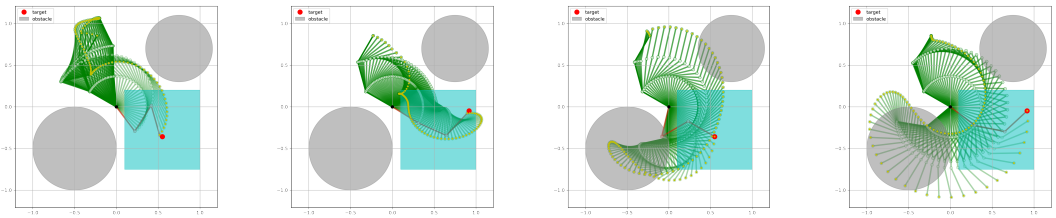
Figure 7.11 – Best 10 out of 50 samples taken from a conditional TT distribution with $\alpha = 0.8$ for inverse kinematics of a 3-link planar manipulator in the presence of obstacles. The samples are already close enough to the optima even without refinement and the multimodality of the solutions is clearly visible.

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.



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

Figure 7.12 – 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.



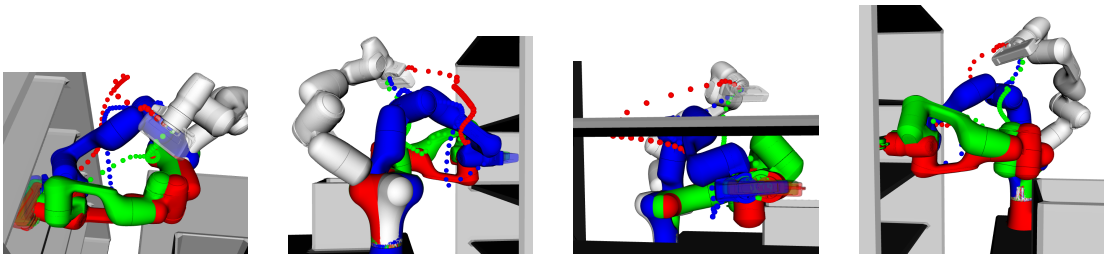
(a) TTGO Task-1 (b) TTGO Task-2 (c) Random Task-1 (d) Random Task-2

Figure 7.13 – 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).

7.6 Discussion

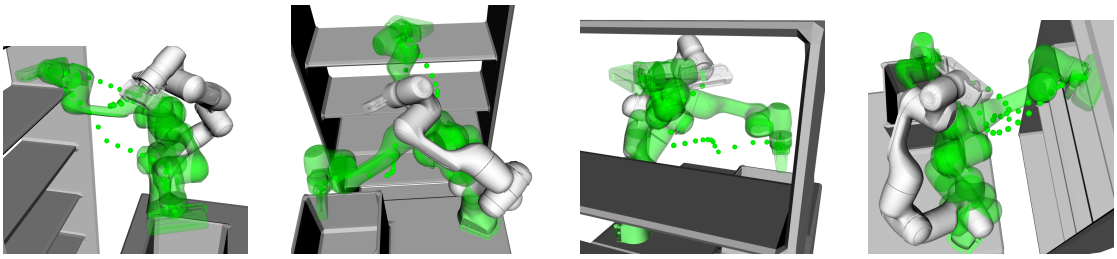
7.6.1 Quality of the Approximation

In this chapter, 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.



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

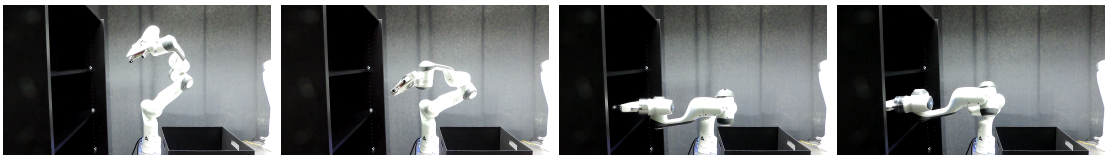
Figure 7.14 – Best 3 out of 1000 samples taken from a conditional TT distribution with $\alpha = 0.75$ for the reaching task of a manipulator in the presence of obstacles, after refinement. The initial configuration is shown in white, while the final configuration is shown in red, green, and blue, for 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)

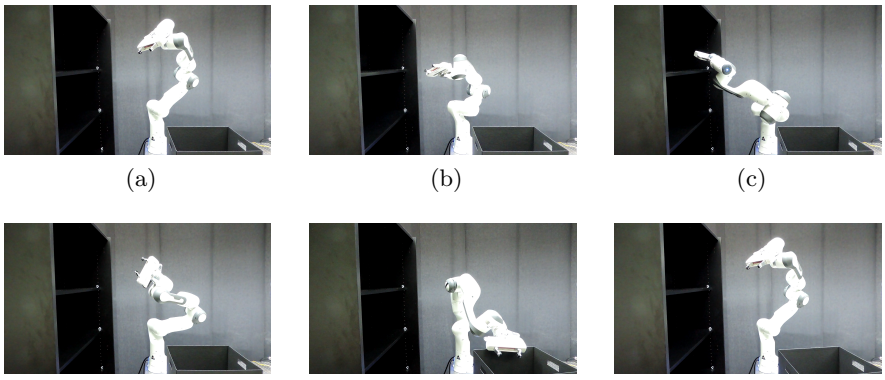
Figure 7.15 – A sample taken from a conditional TT distribution for the pick-and-place task, after refinement. (a) to (d) represent the same motion in different perspectives. In green, we see the picking configuration (from the shelf) and placing configuration (on the box), while the initial configuration is shown in white. The end effector positions in the shelf and the box are the task parameters.

When training the TT model, we can evaluate the quality of the approximation by picking a set of random indices, computing the value of the approximate function at those indices, and comparing it against the actual function value. This is an important evaluation for most applications that aim at finding an accurate low-rank TT decomposition of a given tensor across the whole domain. For our case, though, we are only interested in the maxima of the function, and we do not really care about the approximation accuracy in the low-density region, i.e., the region with the high cost. Even if TT-cross cannot find an accurate low-rank TT representation across the whole domain (e.g., due to non-smoothness), it can still capture the maximal elements robustly [146, 165] 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.



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

Figure 7.16 – 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)

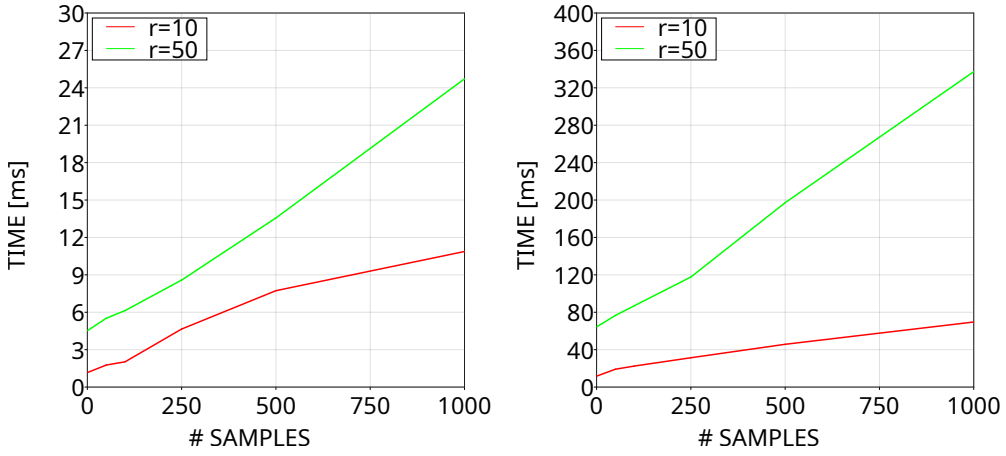
Figure 7.17 – Real robot implementation of one of the TTGO solutions for the pick-and-place task. (a) and (f) represent the initial and the final configuration of the robot (same in this case), (a) to (c) show the motion from the initial configuration to the picking configuration, (c) to (e) show the motion from the picking configuration to the placing configuration.

7.6.2 Computation Time

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

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.

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



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

Figure 7.18 – 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 ($r = 50$).

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 chapter we kept the rank r to be reasonably large (about $r = 60$ for IK and motion planning problems with manipulators) so that we could obtain a variety of solutions from TTGO.

For most of the 2D benchmark functions, it takes less than 0.01s to obtain the TT model. For the high-dimensional mixture of Gaussians and Rosenbrock functions with $d < 30$, we could obtain good enough TT-models in less than 60s. It takes about 30s for the inverse kinematics problem with the Franka Emika robot, which corresponds to 30 iterations of TT-cross. Finally, the target reaching task takes around 10 minutes while the pick-and-place task takes around 1 hour. The motion planning computation time is relatively slower due to the time for computing a single cost function since we compute the obstacle cost at small time intervals. It can be made faster by considering the continuous collision cost as done in TrajOpt [148], 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 7.18. 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.

7.6.3 Comparison With Previous Work Using Variational Inference

As described in Section 7.2.3, the work closest to our approach is SMTO [66] 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 chapter 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 7.13, 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 B.6.

In [147], 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 7.5.1 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 chapter. 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 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 even 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 problems. The method could potentially be applied to other robotics challenges that can be formulated as optimization problems such as task and motion planning or optimal control, as we plan to investigate in future work. we could find the optima of a 50D mixture of Gaussians with 5 components and 30D Rosenbrock considered in Section 7.5.1 in less than 2 seconds. In this way, TTGO can be considered as a tool for global optimization that can offer multiple solutions. Appendix B.6 discuss this in more detail.

In this chapter, 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.

7.6.4 Multimodality

As we have shown in this chapter, TTGO is able to generate samples from multiple modes consistently. Furthermore, continuing the iteration of TT-cross will result in covering more modes as the rank of TT-model can be dynamically increased in the TT-cross

algorithm. However, unlike GMM, it is not easy to sample from only a specific mode, or to identify how many modes there are in a given problem. If we need to cluster the samples, standard clustering algorithms such as k-means clustering can be used.

7.6.5 Further possible extensions

As test cases for TTGO, we considered in this chapter 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 chapter 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 7.2.2), 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 [167] or footstep planner for legged robots [168] 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 chapter, 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 [160], [162], [163]. 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 chapter. However, in robotics applications as described in Section 7.1, 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 this chapter, we used an exponential function as the transformation function, however, a study on other possible transformation functions should be investigated in future work. Moreover, in many robotics applications, the user has the flexibility to design the cost function. This will also play a role in 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 chapter we used the standard cost functions and it was non-smooth due to the cost on collision avoidance. However, a smoother cost function could still potentially be designed for such applications. This could improve the performance and the computation time given in this chapter.

7.6.6 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 chapter, 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.

7.7 Conclusion

In this chapter, 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.

8 Discussion

8.1 Warm Starting Optimal Control Problem

In Chapter 4, we have discussed the problem of warm starting an OCP. In that chapter, we always performed the prediction at the beginning of the motion. In practice, however, we often solve an OCP in a receding-horizon MPC fashion. This implies that the memory of motion may need to be queried at every MPC step, instead of only at the beginning, and the time budget for refining the initial guess is limited by the MPC control frequency. Furthermore, when running an MPC controller, it is common to initialize every MPC iteration with the solution from the previous iteration. When there is no big change in the OCP between two subsequent iterations, such initialization would already be very good, and memory initialization would not be necessary. When there is a significant change in the OCP, though, such as a moving obstacle, a moving target, or a large disturbance, initialization using the previous solution may not be sufficient anymore, and the memory of motion may provide a better initialization. In [45], we investigated the problem of warm starting a whole-body MPC controller for the Talos humanoid robot. The task was to move the robot's hand to track a moving target while avoiding a collision object, i.e., a pole. Instead of predicting the trajectory for the whole movement, we only predict for the MPC horizon. To do that, we construct the dataset by running the MPC controller itself in simulation, initialized by a constrained-RRT planner [34] such that the solver can find solutions that are collision-free. While the collision object is very simple, the MPC solver cannot perform well without a good initialization, i.e., the hand will simply stop in front of the pole without knowing that it can get around the obstacle. In [45], we showed that the initialization from the memory helps the MPC controller to avoid the poor local optima. Figure 8.1 shows some snapshots of both the simulation and the experiment.

We also implemented a similar idea for the case of a whole-body MPC controller to control Anymal locomotion for climbing stairs with trotting motion. Similarly to Section 8.1, we also used the MPC controller to generate the dataset. Figure 8.2 shows an example of

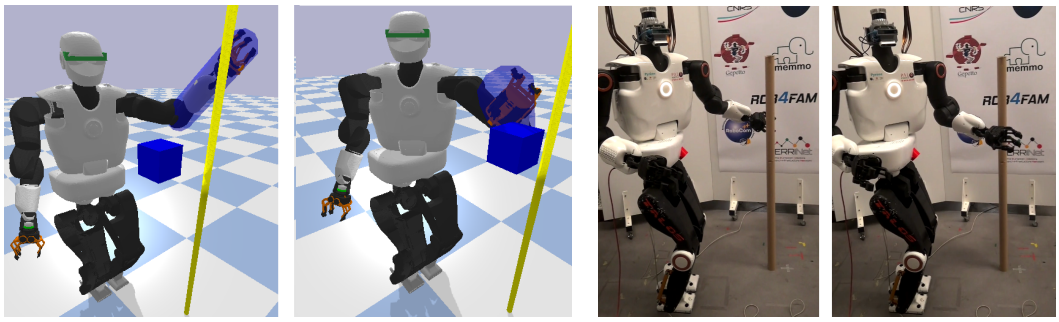


Figure 8.1 – The humanoid robot Talos 1 performing reactive collision avoidance while following a moving target, driven by a whole-body MPC controller initialized by the memory of motion at 100 Hz.

the Anymal climbing stairs. Similarly to [45]), the solver has some challenges in avoiding the obstacles without good initialization. In [169], we showed that the initialization from the memory of motion improves the success rates of the trotting motion.

8.2 Environment Representation

In Chapter 3-7, we always assumed that the environment is static. It implies that when the environment changes, we need to re-generate the predictive models. However, in many applications, the environment often remains unchanged except for a few moving obstacles (e.g., a personal robot working in a kitchen will find that the kitchen will be mostly the same except for some moving objects). We could potentially consider the positions of these moving obstacles as additional inputs to the predictive model when building the database. Moreover, since some of our proposed models can provide many solutions, we could filter these solutions according to their collision status.

A more ambitious target is to consider handling major changes in the environment. To do this, we need to use a good environment representation as an additional input to the model. In some applications such as legged robot navigation, a 2.5D representation such as a heightmap is often used [170, 171, 172]. Such a representation, however, is not suitable for manipulation tasks where we need to consider the 3D obstacles. Voxel-based representation exists where we generate a 3D grid of the task space and specify whether each grid point has an obstacle, but the amount of parameters to represent it becomes quite large. In [173], Strudel *et al.* use PointNet architecture to find a good representation of a point cloud in the context of motion planning.

In [150], we considered an alternative environment representation called Tensor-Train - Signed Distance Function (TT-SDF). For a given environment, we first computed a 3D tensor that contains the SDF values (i.e., the distance to the nearest obstacle surface) of all of the discretized points in the environment. We then used a Tensor Train (TT)

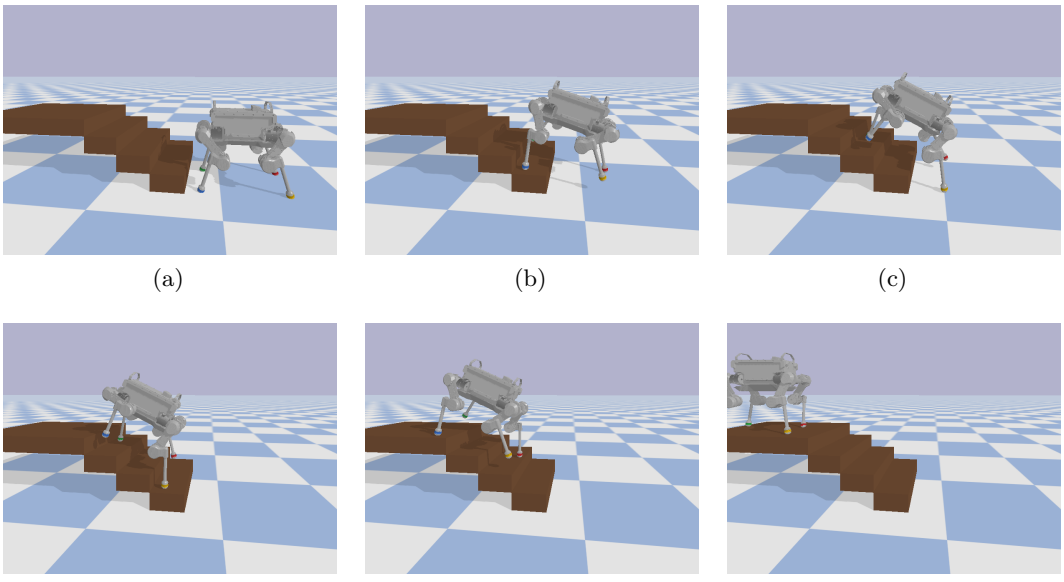


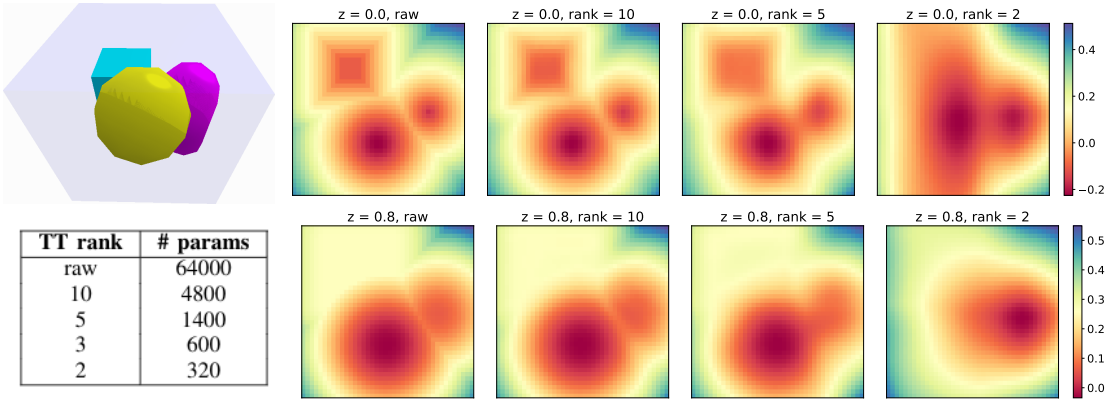
Figure 8.2 – Anymal climbing stairs with 15cm height and 30cm width with the whole-body MPC controller.

decomposition, the same technique used in TTGO, to find a low-rank representation of the tensor. This allows us to have a more compact representation of the environment which is then used as an input to the predictive model. Figure 8.3 shows some slices of the SDF tensor with different ranks representing the environment on the left of the figure. We can see that as the rank is lower, the resulting SDF matrix slice loses the detail. For the trajectory prediction, however, we do not need to know the fine detail of the environment, so a low-rank representation can still be useful to capture the main features of the environment. Note that the rank-2 tensor only requires 320 parameters compared to 64000 in the original tensor (2x2x2m environment with 0.05m discretization). In [150], we showed that such representation is enough to predict good initializations in a given random environment. We have tested it on a point mass and a quadcopter motion planning. Since the data is multimodal, we used MDN as the function approximator.

8.3 Predicting Value Function

In most of this thesis, we concentrated on using the memory of motion to provide a good initialization. However, a memory of motion could also potentially be used to predict some other useful information. In Chapter 3, for example, we considered using the memory of motion to choose between several possible goals. Another interesting possibility, especially in the context of MPC, is to predict the *value function*.

A value function provides the information about how valuable it is to be in a state $\boldsymbol{x}$ at



TT rank	# params
raw	64000
10	4800
5	1400
3	600
2	320

Figure 8.3 – Examples of SDF function plots associated to the environment (left) made of three obstacles at different height. From left to right are the raw SDF and the low-rank approximation with rank 10, 5, and 2, respectively. The table at the bottom left shows the number of parameters for a given rank.

time t (for finite-horizon problem) when considering the long-term goal. If we have a perfect value function, we can reduce an optimal control problem from a finite horizon problem to a one time step optimization, since the value function can tell us where to move optimally with respect to the goal. However, computing the value function is often intractable. This motivates some works that try to predict the value functions based on previous experience [174, 175], which can be seen as another form of memory of motion. Since the value function would not be approximated perfectly, we cannot reduce the horizon to one time step, but we can reduce the length of the horizon quite significantly.

In [113], we applied the idea to Visual Predictive Control (VPC), i.e., visual servoing using an MPC formulation. We used the memory of motion to predict not only the initialization but also the waypoints, i.e., the local goals (in the forms of visual features) to be reached by the robot at the end of the current horizon. These waypoints were then used to formulate the terminal cost function of the VPC problem, effectively serving as the approximate value function. The terminal cost informs the solver that the robot should try to reach this waypoint at the end of the horizon. This allowed us to reduce the horizon from 30 time steps to only 3 time steps while achieving even better performance than the long horizon VPC. Without the value function, the solver would not be able to consider the long-term goal and get stuck at poor local optima.

In [176], we applied a similar idea to Multi-Contact Receding-Horizon Planning (RHP), where the optimization solver plans the center-of-mass trajectory and the footstep locations of a biped walking through an unstructured environment full of rubbles. We used the memory of motion to predict a local objective, i.e., the optimal center of mass position and the footstep location at the end of the horizon, and construct a terminal cost function to reach this local objective. Again, we found that the approximate value function prediction allows us to reduce the planning horizon from 2-3 steps to only 1

step. This makes the online RHP execution feasible since the computation time is now within the time budget. Figure 8.5 shows an example of the planned motion, where we used TSID to track the planned center-of-mass trajectory and the footstep locations.

8.4 Evaluating Warm Start Performance

Using machine learning to predict the initialization in a way removes some burden from the learning algorithms, as we do not require them to be very precise. For many applications considered in this thesis, accurate prediction (especially for control problems) using only machine learning is very challenging. On the other hand, evaluating the performance of the initialization is not very straightforward.

During the training of the model, especially in supervised learning, the most convenient way to test the prediction quality is to compare it against the ground truth, usually a test set separated from the training set, using standard metrics such as the L2 norm of the difference between the predicted value and the ground truth. For multimodal problems, however, such a metric does not provide very useful insight into the model performance, as we may be comparing a good prediction against the ground truth solution but coming from a different mode, resulting in a large prediction error. Furthermore, a model with better accuracy based on the test set does not necessarily perform significantly better than another model with less accuracy, as we saw in Chapter 4 with GPR and GMR. Indeed, the optimizer may require a similar number of iterations to converge when it is initialized by two different initial guesses, even when one is closer to the optima.

The number of iterations to convergence is also not very useful when evaluating the warm start performance. We observed that in many problems, even though the initial guess is very close to the optima, it only speeds up the convergence by at most two times compared to a standard initialization. This depends on both the optimization problems and the solvers. For an unconstrained convex problem, starting with a good initial guess close to the optima may require only 1-2 iterations, especially when using Newton method. In nonconvex problems with many constraints, on the other hand, the optimization solver may continue to improve the solution without converging, resulting in a large number of iterations even though the solution may already be feasible. Concerning the solver, some solvers can benefit from good initialization better than other solvers. For example, when evaluating the GAN sampler in Chapter 6, we did a comparison between using L-BFGS-B and Gauss-Newton as the optimization solver. When provided with good initialization, L-BFGS-B still requires around 10 iterations until convergence, while Gauss-Newton requires only around 3 iterations. This is because Gauss-Newton provides a good approximation of the Hessian for a nonlinear least-squares problem, while L-BFGS-B still requires several iterations to find a good approximation of the Hessian.

Some works [177, 178] consider the evolution of the cost function values over the op-

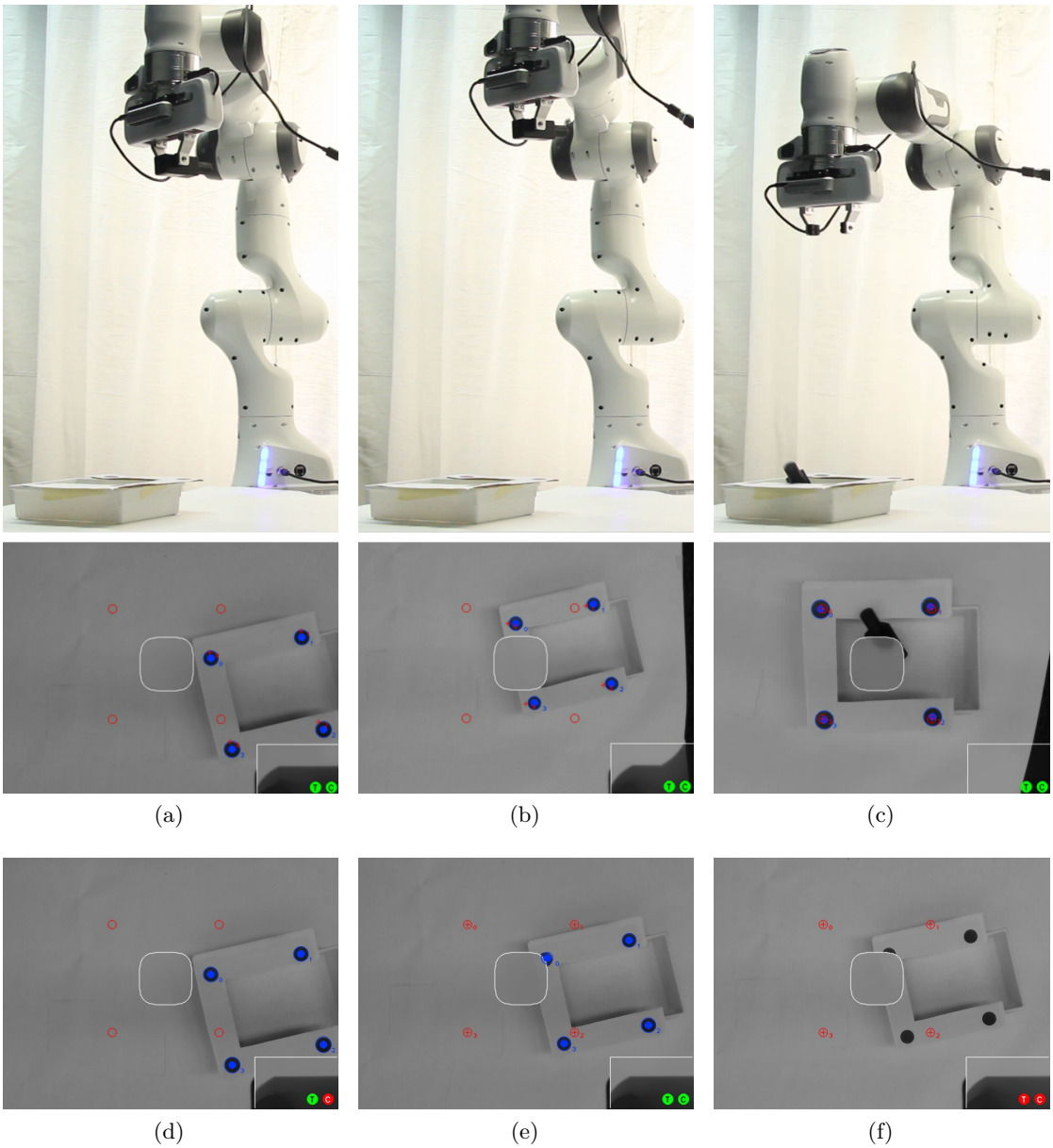


Figure 8.4 – Robot experiment using the memory of motion: VPC is able to avoid the occlusion and achieve the desired task. (a) to (c) show the robot moving the end effector such as the visual features (shown in blue) moves to match the desired visual features (shown in red) while avoiding the blurred area at the center (shown as white box). (d) to (f) shows the VPC failure when initialized by previous solutions, the robot is stuck at a local optima and does not move anymore.

timization iterations when initialized by the warm start or the standard initialization. This provides more useful information, as we can often see from the plots that the good initialization has a low initial cost value and reaches an acceptable value with few

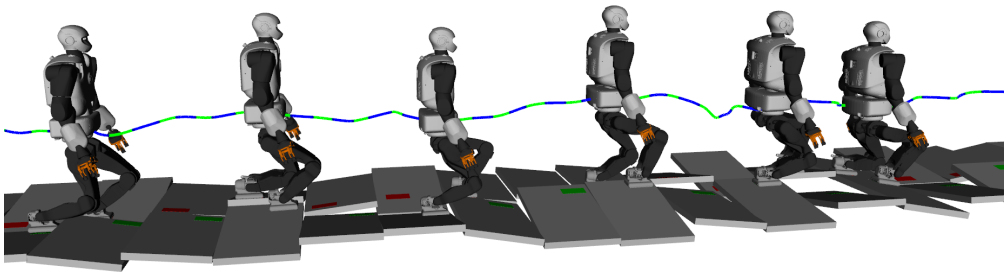


Figure 8.5 – Snapshots of our simulation on the terrain with moderate slopes (5-12 degrees).

iterations, even though the convergence may take longer. In robotics, we often do not need the solver to converge, as long as the solutions are feasible. In practice, we can set some thresholds with respect to the cost function values and stop the iterations as soon as the cost value is below the threshold regardless of the convergence.

Ultimately, the most useful way of evaluating the prediction quality is by implementing it on the real system and testing the actual target metric, e.g., the success rates or the final cost value. These metrics take much longer to compute compared to testing against the ground truth data, but they provide a more solid and useful evaluation. While it is more difficult to train the model using these metrics, some works construct the memory of motion by iterating between training the memory of motion based on the dataset and using the memory of motion in a real setting to gather more training data based on its real performance [70, 179].

8.5 Supervised Learning versus Probability Density Estimation

In the first part of this thesis, we have explored various supervised learning techniques to predict the initialization. The advantages of the supervised learning formulation are that it is easy to formulate and there are a lot of function approximation techniques available. However, as we have shown in Part I, using those techniques as a black box may result in poor prediction, especially for multimodal problems. While several models such as GMM and MDN can improve the prediction in such cases, we still need to ensure that the different modes are sufficiently covered in the dataset. Furthermore, standard supervised learning only uses the prediction accuracy with respect to the ground truth (i.e., the dataset) to train the predictive model. In multimodal problems, however, prediction accuracy may not correspond to the main objective, i.e., providing an initialization that is near to good local optima. In the motion planning case, for example, a model may predict a good motion that completes the task succesfully (e.g., reaching the target while avoiding the obstacles), but such a motion does not exist in the database. In that case,

the prediction accuracy of the model will be low, although the performance is actually good. This is because such supervised learning models do not consider the actual cost function when training the models, although we have access to the cost function.

In Part II, we explored other models that make better use of our knowledge about the cost function. We first transformed the cost function into a (possibly unnormalized) PDF and used various probability density estimation techniques to approximate the PDF. These techniques, in our view, provide much better predictions compared to supervised learning, as the information about the cost function is used to construct/train the predictive model. The model predicts approximate solutions that have good probabilities of having low-cost values instead of simply imitating the dataset. In the proposed GAN framework (Chapter 6), for example, we used the cost functions related to the IK to train the generator, on top of the standard cost function based on the discriminator output. It help the generator to avoid averaging the multimodal solutions, even when we use a standard MLP as the generator (compare this with training the MLP in a supervised manner, where it will average the different modes). In Chapter 7, we used a tensor to approximate the unnormalized PDF that corresponds to the cost function, allowing us to sample approximate solutions from high-density regions, i.e., regions with low-cost values. In conclusion, using the cost functions for training the predictive model helps in providing good and rich initializations, especially when such cost functions are readily available for the given tasks.

While Chapter 5 is limited to optimal control problems solved by iLQR, the methods in Chapter 6 and 7 are more general. Given an optimization problem, it is possible to approximate the corresponding PDF by using either GAN or TT model. However, there are several important differences between the two methods. While it is easier for GAN to scale up to higher dimensions (it has been used for generating videos), GAN requires a dataset of good solutions generated by some other methods. This makes it more difficult to apply it on challenging optimization problems, as generating a good dataset that covers the whole solution space is not trivial. In contrast, TTGO does not require a dataset generated by another method. It only requires the cost function definition, making it more easily applicable to general optimization problems. Furthermore, as described in Chapter 6, GAN often converges to only a subset of the modes. While this was mitigated in Chapter 6 by using an ensemble of networks as the generator, it remains heuristics and there is no guarantee that the different networks will converge to different modes. In contrast, TTGO handles multimodal problems naturally without any heuristics. As we continue the training and the maximum rank is increased, the capacity of the model will continue to grow and it can cover more modes (unlike GAN, which is limited by the structure of the generator and the discriminator). It is more difficult, however, to scale TTGO to very high-dimensional problems compared to GAN. Moreover, the sampling time for GAN is faster than TTGO. With these differences, the two methods are quite complementary and exploring both methods for other optimization problems in robotics would be an interesting future direction.

8.6 Comparison with Deep Reinforcement Learning

Recently, Deep Reinforcement Learning (Deep RL) has gained popularity due to its impressive performance on challenging tasks, including in legged robots locomotion [116, 170, 171, 180, 181, 182]. RL models can also be seen as another form of memory of motion where we use the experience to train a policy network. There is, however, an important difference with the approaches considered in this thesis. In most RL works, the aim is to use the learned model to directly control the robot. The model is hence required to be very precise and reliable, which are difficult to ensure. In most cases, they accomplish this at the expense of a huge computation cost and a long training time. On the contrary, in our approach, we only require the model to provide an initial guess that helps the solver to avoid poor local optima and to speed up the computation. The required precision is achieved by the optimization solver, which is model-based. This gives us a more predictable and reliable behavior as compared to RL methods. Additionally, for most of the works in this thesis, the computation only requires between a few seconds to at most a few hours with a standard computer.

9 Conclusion and Future Works

9.1 Conclusion

In this thesis, we have explored the idea of using a memory of motion to improve the performance of optimization solvers for a wide range of robotics problems in the field of motion planning and robot control. The memory of motion consistently improves the solver's performance in terms of the computational speed, the quality of the solutions, and the success rates. We considered two different formulations in this thesis, i.e., supervised learning and probability density estimation.

In Part I, we formulated the problem of constructing the memory of motion as a regression problem to learn the mapping from the task descriptors to the initial guess, and we used various supervised learning techniques to solve the regression problem. In Chapter 3, we proposed an ensemble method that combines different function approximators (i.e., k-Nearest Neighbors (k-NN), GMR, and GPR) to initialize an optimization-based motion planner. We showed that GMR can handle multimodal problems better than GPR, but GPR outperforms GMR when there is no multimodality. Combining several function approximators with different characteristics results in a significant increase in the success rates. In Chapter 4, we considered warm starting an optimal control problem for biped locomotion on a flat ground. Besides predicting the state trajectory, we also need to provide the initial guess of the control sequence. We considered different combinations of initialization elements and show that providing both the state trajectory and control sequence as initialization results in the best performance.

The techniques in Part I do not make use of the cost function in the training loop. As such, they only imitate the solutions in the database, and the resulting predictions do not necessarily have a low cost, especially in the case of multimodal problems. In Part II, we considered a different formulation by first transforming the cost function into an unnormalized PDF. We then considered several different techniques to approximate the PDF. In Chapter 5, we made use of the fact that iLQR solves an optimal control problem

by solving an LQR problem at each iteration. Since the solution of an LQR problem can be interpreted as a Gaussian distribution, we proposed an approximate solution to an iLQR problem as a Gaussian distribution centered at the optimal solution, while the covariance is the inverse of the (approximate) Hessian. This is similar to using Laplace approximation to approximate the posterior distribution, except for the fact that the Hessian is approximate (since the dynamics is approximated as a linear function). We showed that tracking this distribution with a short-term horizon MPC results in a more robust and control-efficient controller compared to tracking only the optimal solution (i.e., the mean of the distribution).

In Chapter 6, we used a modified GAN framework to approximate the probability density induced by the cost function. We first generated a dataset using the cost function, but instead of solving a regression problem as done in Part I, we used the dataset to train both a generator and a discriminator in an adversarial manner. Furthermore, we also trained the generator with the original cost function, in addition to the standard cost function based on the discriminator output. It helps the generator to avoid averaging multiple modes even when using a standard MLP. To help the generator to cover a larger portion of the distribution, we used an ensemble of networks that are proven to be able to produce multiple solutions. We showed that the proposed method can generate good samples from the constrained manifold in the robot configuration space.

In Chapter 7, we explored the use of tensor methods to approximate an unnormalized PDF induced by a cost function. The PDF is first discretized over a rectangular domain, and we consider a tensor as the discrete approximation of the PDF. We then use the TT-cross algorithm to compute the TT decomposition of the tensor, which allows us to have a compact representation of the tensor model. Moreover, TT-cross also allows us to construct the tensor model without having to compute and store all the elements in it. Since the method does not rely on gradient information, we observed that it can often find the global optima even for challenging problems. After the training, we can condition the TT model on a given task parameter to obtain the corresponding approximate solutions quickly, in the order of milliseconds. We tested the proposed method first on some benchmark functions and then on robotics problems including inverse kinematics with obstacles and motion planning. We showed that the methods can provide good approximate solutions that come from multiple modes when they exist.

We discussed in Chapter 8 that the use of cost functions in the training loop is important when predicting good initialization compared to simply imitating the dataset. We also discussed the challenge of evaluating the warm start quality. Finally, we presented several other works that are not included in the main chapters, including warm starting MPC solver, considering environment representation using a TT decomposition, and using the memory of motion to predict the value function to shorten the MPC horizon.

In conclusion, we have shown that the concept of memory of motion has a significant

benefit to the optimization solver. There are various methods and approaches available to provide such initialization, of which we have explored quite a few in this thesis. As more and more problems are cast as optimization problems, exploring the memory of motion idea will continue to be interesting and relevant to the robotics community. Finally, in the next sections, we consider some future directions that can be pursued in this field.

9.2 Future Works

9.2.1 Other Usages of a Memory of Motion

The basic idea of a memory of motion is that we use our previous experience to guide future tasks. We have so far considered several usages of a memory of motion, e.g., to provide the initialization, or to predict the value function. Other potential usages of memory of motion can also be explored. For example, in manipulation tasks, force interactions with the environment or the objects to be manipulated can be quite intricate, and using the memory to provide the expected force profile can be interesting to explore. Some work from the learning from demonstration field has explored this idea by using the Dynamical Movement Primitives (DMP) to encode both state and force trajectory [73]. As simulation tools and optimization-based solvers for contact manipulations improve, we can potentially generate a database of such force information and use the memory to predict the expected force profile for a given task. Another possibility is in the field of multi-contact locomotion [168], where the memory of motion can be used to predict the good contact locations or the optimal contact surfaces to make contact with. A similar problem in task and motion planning [167], where the robot needs to choose the optimal discrete sequence of tasks while optimizing the continuous motions at the same time, can also benefit from the memory of motion.

9.2.2 Data Generation

Generating a dataset of optimal solutions is not trivial for many optimization problems. The standard approach of random initialization and using the optimization solvers to obtain the optimal solution, as used in most of this thesis, can miss many important solutions, especially for challenging tasks. TTGO (Chapter 7) offers a nice solution to the data generation by integrating the dataset construction, representation, and learning in one model, i.e., the TT model. However, to scale up to higher dimensional problems (i.e., higher than 100 dimensions), further research has to be done. Iterating between data generation and model training, as done in [70, 179] can potentially be explored. Furthermore, active sampling strategy [183, 184] can also be investigated to build a minimally sufficient dataset.

In [150], we have attempted to consider an environment representation in the memory of

motion. Besides finding the proper environment representation, generating a random 3D environment is also not trivial. If we consider the voxel-based representation, there is practically intractable number of possible environments that can be generated. However, not all of those possibilities have real practical values. Instead, most works simply pick a set of possible environment primitives and generate a random environment by mixing those primitives. For example, in [185, 186], they generate a random table environment by picking several objects from a fixed set and dropping them randomly from above the table. In [170], Chemin *et al.* set some standard environment primitives such as stairs, flat ground, and rubbles, and generate a random combination of them. These ideas can be adapted to other applications, although it is not always easy to determine these primitives.

9.2.3 Using the Cost Functions in the Training

In Part II, we have considered using the cost functions to train the predictive model, specifically by transforming it into an unnormalized PDF. There are other ways to include it in the training, e.g., by performing iterative learning between training the model and generating new data using the trained model [70]. We can also use the cost functions as the additional loss functions to train the supervised learning model, especially if we can get access to the gradient of those cost functions, similar to what we did in Chapter 6. Finally, the emerging class of differentiable simulators [187, 188] opens the possibility of differentiating cost functions that were previously intractable, such as in the simulation of legged robot locomotion or robot manipulation involving contacts with the environment or the objects.

9.2.4 Trajectory Representation

In many applications, the output of the predictive model is a trajectory of some quantities, e.g., the robot state, the control command, etc. Such outputs are very high-dimensional, and the variables are usually highly correlated due to the smoothness property or some synergy between the different joints. It is difficult to ask the predictive model to output the whole trajectory in the raw state, as the correlations will be difficult to enforce. Finding a low-dimensional representation that keeps the correlation structure of the trajectory will help the predictive model to provide a better prediction.

In this thesis, we concentrated on using either Radial Basis Functions (RBFs) or Principal Component Analysis (PCA) to represent a trajectory. This simple representation manages to capture the basic correlation between the output variables and enforce the smoothness property, and in our applications, we found that they produce good enough results. More sophisticated nonlinear dimensionality reduction techniques such as Mixture of Probabilistic Principal Component Analyzers (MPPCA) [189] and Mixture of Factor

Analyzers (MFA) [190] can also be considered. However, these methods cannot ensure that the constraints are satisfied, e.g., joint limit or dynamic stability, or even nearly satisfied. In the case of locomotion task, it is especially difficult to ensure that the resulting predicted trajectory is located in the motion manifold, i.e., that the stationary foot remains in place while the other is moving, or that the trajectory is dynamically stable. In Chapter 6, our proposed GAN framework manages to learn a latent representation of the configuration space, such that samples generated from this latent space are still close to the constraint manifold. As emphasized earlier, the use of cost functions in the model training helps to constrain the predictions within the manifold. However, we have only applied the GAN framework to predict a specific configuration instead of a full trajectory. Other works also learn the latent representation of walking motion [191, 192, 193] and managed to obtain some meaningful results. Using these kinds of representations in the memory of motion can result in better performance as the predicted trajectory can retain the correlations that exist between the variables.

A List of Publications

The following is the list of publications by the author during the course of the doctorate study:

- T. S. Lembono, F. Suárez-Ruiz, and Q.-C. Pham, "SCALAR-simultaneous calibration of 2d laser and robot's kinematic parameters using three planar constraints," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2018, pp. 5570–5575
- ——, "SCALAR: Simultaneous calibration of 2-d laser and robot kinematic parameters using planarity and distance constraints," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 4, pp. 1971–1979, 2019
- T. S. Lembono, A. Paolillo, E. Pignat, and S. Calinon, "Memory of motion for warm-starting trajectory optimization," *IEEE Robotics and Automation Letters (RA-L)*, vol. 5, no. 2, pp. 2594–2601, April 2020
- E. Pignat, T. Lembono, and S. Calinon, "Variational inference with mixture model approximation for applications in robotics," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2020, pp. 3395–3401
- T. S. Lembono, C. Mastalli, P. Fernbach, N. Mansard, and S. Calinon, "Learning how to walk: Warm-starting optimal control solver with memory of motion," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2020, pp. 1357–1363
- A. Paolillo, T. S. Lembono, and S. Calinon, "A memory of motion for visual predictive control tasks," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2020, pp. 9014–9020
- T. S. Lembono, E. Pignat, J. Jankowski, and S. Calinon, "Learning constrained distributions of robot configurations with generative adversarial network," *IEEE Robotics and Automation Letters (RA-L)*, vol. 6, no. 2, pp. 4233–4240, 2021

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

- T. S. Lembono and S. Calinon, "Probabilistic iterative LQR for short time horizon MPC," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2021, pp. 579–585

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

- A. Razmjoo, T. S. Lembono, and S. Calinon, "Optimal control combining emulation and imitation to acquire physical assistance skills," in *Proc. IEEE Intl Conf. on Advanced Robotics (ICAR)*, 2021, pp. 338–343

- H. Girgin, T. S. Lembono, R. Cirligeanu, and S. Calinon, "Optimization of robot configurations for motion planning in industrial riveting," in *Proc. IEEE Intl Conf. on Advanced Robotics (ICAR)*, 2021, pp. 247–252

- J. Wang, T. S. Lembono, S. Kim, S. Calinon, S. Vijayakumar, and S. Tonneau, "Learning to guide online multi-contact receding horizon planning," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2022

- S. Shetty, T. S. Lembono, T. Löw, and S. Calinon, "Tensor train for global optimization problems in robotics," *Submitted article under review*, 2022, https://sites.google.com/view/ttgo/home

B Appendices to Chapter 7

B.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 7.3.3. 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 $k \in \{1, \ldots, d\}$,

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

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

$$P(x_1, \ldots, x_d) \approx \boldsymbol{P}^1(x_1) \cdots \boldsymbol{P}^d(x_d). \tag{B.2}$$

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

B.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 chapter. For more detail, we refer the readers to [197, 166, 159].

1. *Input*: d-th order tensor $\boldsymbol{\mathcal{P}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, approximation rank $\boldsymbol{r} = (r_1, \ldots, r_{d-1})$
2. *Set:* $r_0 = 1, r_d = 1, n_0 = 1, n_{d+1} = 1$

3. *Initialize:* randomly choose initial column index sets $\boldsymbol{j}^k = (j_1^k, \ldots, j_{r_k}^k) \subset (1, \ldots, n_{k+1} \cdots n_d)$, for $k = 1, \ldots, d-1$
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:
   (a) for $k = 1, \ldots, d-1$ (forward sweep)
      - using MAXVOL to find the row index set $\boldsymbol{i}^k = (i_1^k, \ldots, i_{r_k}^k) \subset (1, \ldots, n_k r_{k-1})$ corresponding to the maximum submatrix of $\boldsymbol{P}^k_{:,\boldsymbol{j}^k}$
      - reshape the matrix $\boldsymbol{P}^k_{\boldsymbol{i}^k,:}$ into matrix $\boldsymbol{P}^{k+1} \in \mathbb{R}^{n_{k+1} r_k \times n_{k+2} \cdots n_d}$.

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

   (c) for $k = d-1, \ldots, 1$ (backward sweep)
      - using MAXVOL to find the column index set $\boldsymbol{j}^k = (j_1^k, \ldots, j_{r_k}^k) \subset (1, \ldots, n_{k+1} \cdots n_d)$ 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 $k = 1, \ldots, d$

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 submatrices) 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 [166, 145] for the details.

B.3 Inverse Kinematics Formulation

The cost function for the inverse kinematics problem in Section 7.5.2 is given by

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

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.

B.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

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

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 7.5.2 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 B.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.

B.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.

B.6 TTGO with Constant Task Parameters

In this chapter, we described TTGO in its generic form, where we consider varying task parameters when training the TT model. This allowed us to produce approximate

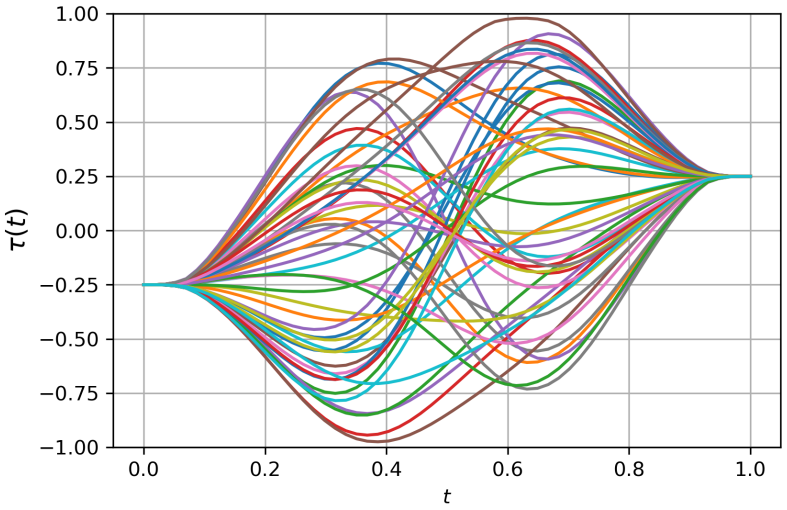


Figure B.1 – A distribution of 50 smooth trajectories generated by transforming trajectories generated by using two radial basis functions with weights chosen uniformly in the range $[-1, 1]$. The transformations are done to maintain a boundary condition $\tau_0 = -0.25, \tau_1 = 0.25$ and the limits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

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 chapter. 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.

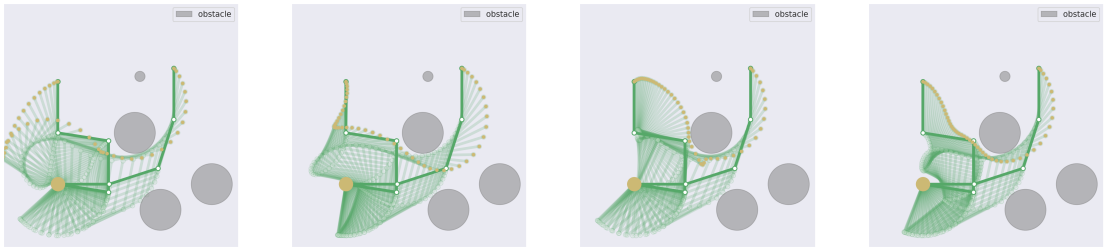
For such applications, our work is closely related to TTOpt [146] which is a gradient-free discrete optimization method based on TT-cross. The performance of the method has been shown to be competitive to evolutionary strategies. In this approach, the objective is to maximize a reward function (analogous 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 manipulatorLausanne-Gare, Lausanne, Switzerland. We set the initial and the desired final configurations, and TTGO finds the trajectory to move to the final configuration while avoiding the obstacles. The joint angle trajectory is represented using the motion primitives as described in Appendix B.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 [66], 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 B.2 shows four solutions obtained by TTGO after the refinement step. We can clearly see the multimodality of the solutions.

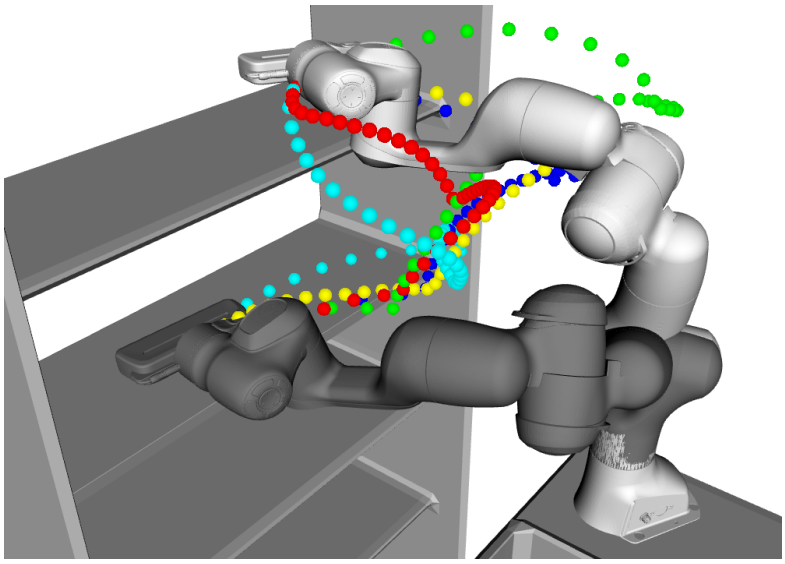
For the Franka Emika manipulator, we use the same setting as in Section 7.5, i.e., with the shelf, table, and box as the collision objects. In addition, we add a cost to maintain the end effector pose (horizontal) throughout the trajectory. The initial and final configurations are set such that both end effector positions are located within the shelf, and they are computed using TTGO for IK, as explained in Section 7.5.2. 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 B.3.

In comparison, SMTO [66] 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 [147] 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 B.2, 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 $\sim 500s$.



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

Figure B.2 – Four different solutions obtained by TTGO for a motion planning task with 4-link planar manipulator. The initial and final configuration are given (dark green) and the optimization variables are the weights of the basis functions (two basis functions per joint) that determine the joint angle trajectory.



(a)

Figure B.3 – 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.

Bibliography

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] T. Sugihara, "Solvability-unconcerned inverse kinematics by the levenberg–marquardt method," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 984–991, 2011.

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

[4] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization," in *Proc. Robotics: Science and Systems (R:SS)*, vol. 9, no. 1, 2013, pp. 1–10.

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

[6] A. Del Prete and N. Mansard, "Robustness to joint-torque-tracking errors in task-space inverse dynamics," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1091–1105, 2016.

[7] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems." in *International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, 2004, pp. 222–229.

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

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

[10] W. Merkt, V. Ivan, and S. Vijayakumar, "Leveraging precomputation with problem encoding for warm-starting trajectory optimization in complex environments," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[11] T. S. Lembono, E. Pignat, J. Jankowski, and S. Calinon, "Learning constrained distributions of robot configurations with generative adversarial network," *IEEE Robotics and Automation Letters (RA-L)*, vol. 6, no. 2, pp. 4233–4240, 2021.

[12] D. Surovik, O. Melon, M. Geisert, M. Fallon, and I. Havoutis, "Learning an expert skill-space for replanning dynamic quadruped locomotion over obstacles," in *Proc. Conference on Robot Learning*, vol. 155, 2021, pp. 1509–1518.

[13] S. Shetty, T. S. Lembono, T. Löw, and S. Calinon, "Tensor train for global optimization problems in robotics," *Submitted article under review*, 2022, https://sites.google.com/view/ttgo/home.

[14] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Iowa State University, Computer Science Department, Tech. Rep. TR 98-11, 1998.

[15] L. E. Kavraki, P. Svestka, J. . Latombe, and M. H. Overmars, "Probabilistic roadmaps for robot path planning," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[16] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Intl Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[17] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, 2006.

[18] M. Toussaint, H. Ritter, and O. Brock, "The optimization route to robotics—and alternatives," *KI-Künstliche Intelligenz*, vol. 29, no. 4, pp. 379–388, 2015.

[19] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *IEEE International Symposium on System Integrations (SII)*, 2019.

[20] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[21] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018.

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance

deep learning library," *Advances in Neural Information Processing Systems (NIPS)*, vol. 32, 2019.

[23] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.

[24] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

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

[26] R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, vol. 79, 2008.

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

[28] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[29] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *Intl Journal of Robotics Research*, vol. 33, no. 7, pp. 1006–1028, 2014.

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

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

[32] C. Park, J. Pan, and D. Manocha, "ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments," in *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

[33] M. Mukadam, X. Yan, and B. Boots, "Gaussian process motion planning," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2016, pp. 9–15.

[34] D. Berenson, S. Srinivasa, and J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *Intl Journal of Robotics Research*, vol. 30, no. 12, pp. 1435–1460, 2011.

[35] Y. Yang, V. Ivan, W. Merkt, and S. Vijayakumar, "Scaling sampling-based motion planning to humanoid robots," in *2016 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2016, pp. 1448–1454.

[36] R. Bischoff, J. Kurth, G. Schreiber, R. Koeppe, A. Albu-Schäffer, A. Beyer, O. Eiberger, S. Haddadin, A. Stemmer, G. Grunwald, *et al.*, "The kuka-dlr lightweight robot arm-a new reference platform for robotics research and manufacturing," in *ISR 2010 (41st international symposium on robotics) and ROBOTIK 2010 (6th German conference on robotics)*, 2010, pp. 1–8.

[37] R. Budhiraja, J. Carpentier, C. Mastalli, and N. Mansard, "Differential dynamic programming for multi-phase rigid contact dynamics," in *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, 2018.

[38] J. Wang, S. Kim, S. Vijayakumar, and S. Tonneau, "Multi-fidelity receding horizon planning for multi-contact locomotion," in *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, 2021, pp. 53–60.

[39] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa, "Biped walking pattern generation by using preview control of zero-moment point," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, vol. 2, 2003, pp. 1620–1626.

[40] P.-B. Wieber, "Trajectory free linear model predictive control for stable walking in the presence of strong perturbations," in *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, 2006, pp. 137–142.

[41] B. Ponton, A. Herzog, A. Del Prete, S. Schaal, and L. Righetti, "On time optimization of centroidal momentum dynamics," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2018, pp. 1–7.

[42] H. Dai, A. Valenzuela, and R. Tedrake, "Whole-body motion planning with centroidal dynamics and full kinematics," in *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, 2014, pp. 295–302.

[43] B. Ponton, M. Khadiv, A. Meduri, and L. Righetti, "Efficient multicontact pattern generation with sequential convex approximations of the centroidal dynamics," *IEEE Transactions on Robotics*, vol. 37, no. 5, pp. 1661–1679, 2021.

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

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

[46] J. Koenemann, A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard, "Whole-body model-predictive control applied to the hrp-2 humanoid," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2015, pp. 3346–3351.

[47] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber, "Fast direct multiple shooting algorithms for optimal robot control," in *Fast motions in biomechanics and robotics*. Springer, 2006, pp. 65–93.

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

[49] Y. Tassa, T. Erez, and W. D. Smart, "Receding horizon differential dynamic programming," in *Advances in Neural Information Processing Systems (NIPS)*, 2008, pp. 1465–1472.

[50] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2012, pp. 4906–4913.

[51] J. Marti-Saumell, J. Solà, C. Mastalli, and A. Santamaria-Navarro, "Squash-box feasibility driven differential dynamic programming," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2020, pp. 7637–7644.

[52] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2014, pp. 1168–1175.

[53] T. A. Howell, B. E. Jackson, and Z. Manchester, "Altro: A fast solver for constrained trajectory optimization," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2019, pp. 7674–7679.

[54] J. Quiñonero-Candela and C. E. Rasmussen, "A unifying view of sparse approximate gaussian process regression," *Journal of Machine Learning Research*, vol. 6, no. Dec, pp. 1939–1959, 2005.

[55] J. Hensman, N. Fusi, and N. D. Lawrence, "Gaussian processes for big data," in *Conference on Uncertainty in Artificial Intellegence*, 2013, pp. 282–290.

[56] S. Calinon, "A tutorial on task-parameterized movement learning and retrieval," *Intelligent Service Robotics*, vol. 9, no. 1, pp. 1–29, 2016.

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

[58] Z. Ghahramani and M. I. Jordan, "Supervised learning from incomplete data via an EM approach," in *Advances in Neural Information Processing Systems (NIPS)*, 1994.

[59] E. Pignat and S. Calinon, "Bayesian Gaussian mixture model for robotic policy imitation," *IEEE Robotics and Automation Letters (RA-L)*, pp. 1–7, 2019.

[60] J. Viereck, J. Kozolinsky, A. Herzog, and L. Righetti, "Learning a structured neural network policy for a hopping task," *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 4, pp. 4092–4099, 2018.

[61] C. M. Bishop, "Mixture density networks," Aston University, Tech. Rep., 1994.

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

[63] M. J. Wainwright, M. I. Jordan, *et al.*, "Graphical models, exponential families, and variational inference," *Foundations and Trends® in Machine Learning*, vol. 1, no. 1–2, pp. 1–305, 2008.

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

[65] M. K. Titsias and F. Ruiz, "Unbiased implicit variational inference," in *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 167–176.

[66] T. Osa, "Multimodal trajectory optimization for motion planning," *Intl Journal of Robotics Research*, vol. 39, no. 8, pp. 983–1001, 2020.

[67] I. Goodfellow, "NIPS 2016 tutorial: Generative adversarial networks," *arXiv preprint arXiv:1701.00160*, 2016.

[68] L. Grasedyck, D. Kressner, and C. Tobler, "A literature survey of low-rank tensor approximation techniques," *GAMM-Mitteilungen*, vol. 36, no. 1, pp. 53–78, 2013.

[69] C. Liu and C. G. Atkeson, "Standing balance control using a trajectory library," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2009.

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

[71] D. Forte, A. Gams, J. Morimoto, and A. Ude, "On-line motion synthesis and adaptation using a trajectory database," *Robotics and Autonomous Systems*, vol. 60, no. 10, pp. 1327–1339, 2012.

[72] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning rhythmic movements by demonstration using nonlinear oscillators," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2002, pp. 958–963.

[73] P. Pastor, L. Righetti, M. Kalakrishnan, and S. Schaal, "Online movement adaptation based on previous sensor experiences," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2011.

[74] P. Pastor, M. Kalakrishnan, L. Righetti, and S. Schaal, "Towards associative skill memories," in *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, 2012, pp. 309–315.

[75] D. Kappler, P. Pastor, M. Kalakrishnan, M. Wüthrich, and S. Schaal, "Data-driven online decision making for autonomous manipulation." in *Proc. Robotics: Science and Systems (R:SS)*, 2015.

[76] P. Leven and S. Hutchinson, "A framework for real-time path planning in changing environments," *Intl Journal of Robotics Research*, vol. 21, no. 12, pp. 999–1030, 2002.

[77] S. Murray, W. Floyd-Jones, Y. Qi, D. J. Sorin, and G. D. Konidaris, "Robot motion planning on a chip." in *Proc. Robotics: Science and Systems (R:SS)*, vol. 6, 2016.

[78] Y. Yang and O. Brock, "Elastic roadmaps—motion generation for autonomous mobile manipulation," *Autonomous Robots*, vol. 28, no. 1, pp. 113–130, 2010.

[79] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *Intl Journal of Robotics Research*, vol. 21, no. 12, pp. 1031–1052, 2002.

[80] P. Lehner, A. Sieverling, and O. Brock, "Incremental, sensor-based motion generation for mobile manipulators in unknown, dynamic environments," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2015, pp. 4761–4767.

[81] S. R. Martin, S. E. Wright, and J. W. Sheppard, "Offline and online evolutionary bi-directional rrt algorithms for efficient re-planning in dynamic environments," in

Proc. IEEE Intl Conf. on Automation Science and Engineering (CASE), 2007, pp. 1131–1136.

[82] M. Phillips, B. J. Cohen, S. Chitta, and M. Likhachev, "E-graphs: Bootstrapping planning with experience graphs." in *Proc. Robotics: Science and Systems (R:SS)*, 2012.

[83] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2012.

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

[85] A. Werner, D. Trautmann, D. Lee, and R. Lampariello, "Generalization of optimal motion trajectories for bipedal walking," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2015, pp. 1571–1577.

[86] R. Lampariello, D. Nguyen-Tuong, C. Castellini, G. Hirzinger, and J. Peters, "Trajectory planning for optimal robot catching in real-time," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2011, pp. 3719–3726.

[87] C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint arXiv:1606.05908*, 2016.

[88] F. Suárez-Ruiz, T. S. Lembono, and Q.-C. Pham, "RoboTSP–a fast solution to the robotic task sequencing problem," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2018, pp. 1611–1616.

[89] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[90] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[91] S. Choudhury, S. Arora, and S. Scherer, "The planner ensemble: Motion planning by executing diverse algorithms," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2015, pp. 2389–2395.

[92] Y. Yiming, W. Merkt, V. Ivan, L. Zhibin, and S. Vijayakumar, "HDRM: A resolution complete dynamic roadmap for real-time motion planning in complex scenes," *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 1, 2017.

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

[94] A. W. Winkler, C. Mastalli, I. Havoutis, M. Focchi, D. G. Caldwell, and C. Semini, "Planning and execution of dynamic whole-body locomotion for a hydraulic quadruped on challenging terrain," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2015, pp. 5148–5154.

[95] J. Carpentier and N. Mansard, "Multicontact locomotion of legged robots," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1441–1460, 2018.

[96] B. Ponton, A. Herzog, A. Del Prete, S. Schaal, and L. Righetti, "On time optimization of centroidal momentum dynamics," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2018, pp. 5776–5782.

[97] B. Aceituno-Cabezas, C. Mastalli, H. Dai, M. Focchi, A. Radulescu, D. G. Caldwell, J. Cappelletto, J. C. Grieco, G. Fernandez-Lopez, and C. Semini, "Simultaneous Contact, Gait and Motion Planning for Robust Multi-Legged Locomotion via Mixed-Integer Convex Optimization," *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 3, pp. 2531–2538, 2017.

[98] S. Fahmi, C. Mastalli, M. Focchi, and C. Semini, "Passive whole-body control for quadruped robots: Experimental validation over challenging terrain," *IEEE Robotics and Automation Letters (RA-L)*, vol. 4, no. 3, pp. 2553–2560, 2019.

[99] P.-B. Wieber, "Holonomy and nonholonomy in the dynamics of articulated motion," in *Fast motions in biomechanics and robotics*. Springer, 2006.

[100] M. Neunert, F. Farshidian, A. W. Winkler, and J. Buchli, "Trajectory optimization through contacts and automatic gait discovery for quadrupeds," *IEEE Robotics and Automation Letters (RA-L)*, vol. 2, 2017.

[101] S. Tonneau, A. Del Prete, J. Pettré, C. Park, D. Manocha, and N. Mansard, "An efficient acyclic contact planner for multiped robots," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 586–601, 2018.

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

[103] O. Stasse, T. Flayols, R. Budhiraja, K. Giraud-Esclasse, J. Carpentier, J. Mirabel, A. Del Prete, P. Souères, N. Mansard, F. Lamiraux, J.-P. Laumond, L. Marchionni, H. Tome, and F. Ferro, "Talos: A new humanoid research platform targeted for industrial applications," in *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, 2017, pp. 689–695.

[104] J. Carpentier, A. Del Prete, S. Tonneau, T. Flayols, F. Forget, A. Mifsud, K. Giraud, D. Atchuthan, P. Fernbach, R. Budhiraja, M. Geisert, J. Sola, O. Stasse, and

N. Mansard, "Multi-contact locomotion of legged robots in complex environments–the loco3d project," in *Workshop on Challenges in Dynamic Legged Locomotion*, 2017.

[105] P. Fernbach, S. Tonneau, A. Del Prete, and M. Taïx, "A kinodynamic steering-method for legged multi-contact locomotion," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2017, pp. 3701–3707.

[106] P. Fernbach, S. Tonneau, and M. Taïx, "CROC: Convex resolution of centroidal dynamics trajectories to provide a feasibility criterion for the multi contact planning problem," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2018, pp. 1–9.

[107] B. Settles, "Active learning literature survey," University of Wisconsin-Madison, Department of Computer Sciences, Tech. Rep. TR 1648, 2009.

[108] T. S. Lembono and S. Calinon, "Probabilistic iterative LQR for short time horizon MPC," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2021, pp. 579–585.

[109] D. R. Robinson, R. T. Mar, K. Estabridis, and G. Hewer, "An efficient algorithm for optimal trajectory generation for heterogeneous multi-agent systems in non-convex environments," *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 2, pp. 1215–1222, 2018.

[110] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa, "Biped walking pattern generation by using preview control of zero-moment point," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, vol. 2, 2003, pp. 1620–1626.

[111] S. Caron and A. Kheddar, "Multi-contact walking pattern generation based on model preview control of 3d com accelerations," in *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, 2016, pp. 550–557.

[112] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and trajectory optimization for legged systems through phase-based end-effector parameterization," *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 3, pp. 1560–1567, 2018.

[113] A. Paolillo, T. S. Lembono, and S. Calinon, "A memory of motion for visual predictive control tasks," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2020, pp. 9014–9020.

[114] H. J. Kappen, V. Gómez, and M. Opper, "Optimal control as a graphical model inference problem," *Machine learning*, vol. 87, no. 2, pp. 159–182, 2012.

[115] K. Rawlik, M. Toussaint, and S. Vijayakumar, "On stochastic optimal control and reinforcement learning by approximate inference," in *Twenty-third international joint conference on artificial intelligence*, 2013.

[116] S. Levine and V. Koltun, "Guided policy search," in *Proc. Intl Conf. on Machine Learning (ICML)*, 2013, pp. 1–9.

[117] S. Calinon, "Stochastic learning and control in multiple coordinate systems," in *Intl Workshop on Human-Friendly Robotics*, 2016.

[118] S. Calinon and D. Lee, "Learning control," in *Humanoid Robotics: a Reference*, P. Vadakkepat and A. Goswami, Eds. Springer, 2019, pp. 1261–1312.

[119] M. I. Ribeiro, "Kalman and extended kalman filters: Concept, derivation and properties," *Institute for Systems and Robotics*, vol. 43, p. 46, 2004.

[120] E. Todorov and M. I. Jordan, "A minimal intervention principle for coordinated movement," in *Advances in Neural Information Processing Systems (NIPS)*, 2002, pp. 27–34.

[121] S. Calinon, D. Bruno, and D. G. Caldwell, "A task-parameterized probabilistic model with minimal intervention control," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2014, pp. 3339–3344.

[122] T. A. Howell, B. E. Jackson, and Z. Manchester, "ALTRO: A fast solver for constrained trajectory optimization," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2019, pp. 7674–7679.

[123] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proc. IEEE Intl Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4681–4690.

[124] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text to image synthesis," in *Proc. Intl Conf. on Machine Learning*, vol. 48, 2016, pp. 1060–1069.

[125] J.-Y. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros, "Generative visual manipulation on the natural image manifold," in *European conference on computer vision*. Springer, 2016, pp. 597–613.

[126] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2016, pp. 512–519.

[127] P. Lehner and A. Albu-Schäffer, "Repetition sampling for efficiently planning similar constrained manipulation tasks," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2017, pp. 2851–2856.

[128] ——, "The repetition roadmap for repetitive constrained motion planning," *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 4, pp. 3884–3891, 2018.

[129] J. Carpentier, R. Budhiraja, and N. Mansard, "Learning feasibility constraints for multi-contact locomotion of legged robots," in *Proc. Robotics: Science and Systems (R:SS)*, 2017.

[130] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2018, pp. 7087–7094.

[131] D. Holden, J. Saito, T. Komura, and T. Joyce, "Learning motion manifolds with convolutional autoencoders," in *SIGGRAPH Asia 2015 Technical Briefs*, 2015, pp. 1–4.

[132] S. Tonneau, A. Del Prete, J. Pettré, C. Park, D. Manocha, and N. Mansard, "An efficient acyclic contact planner for multiped robots," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 586–601, 2018.

[133] H. Ren and P. Ben-Tzvi, "Learning inverse kinematics and dynamics of a robotic manipulator using generative adversarial networks," *Robotics and Autonomous Systems*, vol. 124, p. 103386, 2020.

[134] Z. Kingston, M. Moll, and L. E. Kavraki, "Sampling-based methods for motion planning with constraints," *Annual review of control, robotics, and autonomous systems*, vol. 1, pp. 159–185, 2018.

[135] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2009, pp. 625–632.

[136] M. Stilman, "Global manipulation planning in robot joint space with task constraints," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 576–584, 2010.

[137] F. Kanehiro, E. Yoshida, and K. Yokoi, "Efficient reaching motion planning and execution for exploration by humanoid robots," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2012, pp. 1911–1916.

[138] E. Pignat, H. Girgin, and S. Calinon, "Generative adversarial training of product of policies for robust and adaptive movement primitives," in *Proc. Conference on Robot Learning*, 2020.

[139] H. Dai and R. Tedrake, "Planning robust walking motion on uneven terrain via convex optimization," in *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, 2016, pp. 579–586.

[140] H. Kurniawati and D. Hsu, "Workspace-based connectivity oracle: An adaptive sampling strategy for prm planning," in *Algorithmic Foundation of Robotics VII*, 2008, pp. 35–51.

[141] M. Rickert, A. Sieverling, and O. Brock, "Balancing exploration and exploitation in sampling-based motion planning," *IEEE Transactions on Robotics*, vol. 30, no. 6, pp. 1305–1317, 2014.

[142] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proc. Intl Conf. on Machine Learning (ICML)*, 2017, pp. 214–223.

[143] M. Y. Liu and O. Tuzel, "Coupled generative adversarial networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 469–477.

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

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

[146] K. Sozykin, A. Chertkov, R. Schutski, A. Phan, A. Cichocki, and I. Oseledets, "Ttopt: A maximum volume quantized tensor train-based optimization and its application to reinforcement learning," *ArXiv*, vol. abs/2205.00293, 2022.

[147] T. Osa, "Motion planning by learning the solution manifold in trajectory optimization," *Intl Journal of Robotics Research*, vol. 41, no. 3, pp. 281–311, 2022.

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

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

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

[151] N. D. Sidiropoulos, L. D. Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Transactions on Signal Processing*, vol. 65, pp. 3551–3582, 2017.

[152] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

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

[154] A. Cichocki, D. P. Mandic, L. D. Lathauwer, G. Zhou, Q. Zhao, C. F. Caiafa, and A. H. Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Processing Magazine*, vol. 32, pp. 145–163, 2015.

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

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

[157] A. Gorodetsky, S. Karaman, and Y. Marzouk, "Efficient high-dimensional stochastic optimal motion control using tensor-train decomposition," in *Robotics: Science and Systems (RSS)*, July 2015.

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

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

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

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

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

[163] G. S. Novikov, M. Panov, and I. Oseledets, "Tensor-train density estimation," in *Uncertainty in Artificial Intelligence*, 2021, pp. 1321–1331.

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

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

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

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

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

[169] T. S. Lembono and S. Calinon, "Deliverable 2.3: Memory of motion report and software: final version," Idiap Research Institute, Tech. Rep., 2022, https://www.memmo-project.eu/results.

[170] J. Chemin, P. Fernbach, D. Song, G. Saurel, N. Mansard, and S. Tonneau, "Learning to steer a locomotion contact planner," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2021, pp. 4430–4437.

[171] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, pp. 1–13, 2017.

[172] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and trajectory optimization for legged systems through phase-based end-effector parameterization," *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 3, pp. 1560–1567, 2018.

[173] R. Strudel, R. Garcia, J. Carpentier, J.-P. Laumond, I. Laptev, and C. Schmid, "Learning obstacle representations for neural motion planning," *arXiv preprint arXiv:2008.11174*, 2020.

[174] M. Zhong, M. Johnson, Y. Tassa, T. Erez, and E. Todorov, "Value function approximation and model predictive control," in *2013 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*, 2013, pp. 100–107.

[175] A. Parag, S. Kleff, L. Saci, N. Mansard, and O. Stasse, "Value learning from trajectory optimization and sobolev descent: A step toward reinforcement learning with superlinear convergence properties," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2022.

[176] J. Wang, T. S. Lembono, S. Kim, S. Calinon, S. Vijayakumar, and S. Tonneau, "Learning to guide online multi-contact receding horizon planning," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2022.

[177] N. Jetchev and M. Toussaint, "Fast motion planning from experience: trajectory prediction for speeding up movement generation," *Autonomous Robots*, vol. 34, no. 1, pp. 111–127, 2013.

[178] A. Razmjoo, T. S. Lembono, and S. Calinon, "Optimal control combining emulation and imitation to acquire physical assistance skills," in *Proc. IEEE Intl Conf. on Advanced Robotics (ICAR)*, 2021, pp. 338–343.

[179] A. Duburcq, Y. Chevaleyre, N. Bredeche, and G. Boéris, "Online trajectory planning through combined trajectory optimization and function approximation: Application to the exoskeleton atalante," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2020, pp. 3756–3762.

[180] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, 2019.

[181] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science robotics*, vol. 5, no. 47, p. eabc5986, 2020.

[182] T. Li, H. Geyer, C. G. Atkeson, and A. Rai, "Using deep reinforcement learning to learn high-level policies on the atrias biped," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2019, pp. 263–269.

[183] G. Maeda, M. Ewerton, T. Osa, B. Busch, and J. Peters, "Active incremental learning of robot movement primitives," in *Proc. Conference on Robot Learning*, 2017, pp. 37–46.

[184] H. Girgin, E. Pignat, N. Jaquier, and S. Calinon, "Active improvement of control policies with bayesian gaussian mixture model," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2020, pp. 5395–5401.

[185] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2018, pp. 4238–4245.

[186] M. Ewerton, A. Martínez-González, and J.-M. Odobez, "An efficient image-to-image translation hourglass-based architecture for object pushing policy learning," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2021, pp. 3478–3484.

[187] S. Coros, M. Macklin, B. Thomaszewski, and N. Thürey, "Differentiable simulation," in *SIGGRAPH Asia 2021 Courses*, 2021, pp. 1–142.

[188] M. A. Z. Mora, M. P. Peychev, S. Ha, M. Vechev, and S. Coros, "Pods: Policy optimization via differentiable simulation," in *International Conference on Machine Learning*, 2021, pp. 7805–7817.

[189] M. E. Tipping and C. M. Bishop, "Mixtures of probabilistic principal component analyzers," *Neural computation*, vol. 11, no. 2, pp. 443–482, 1999.

[190] Z. Ghahramani, G. E. Hinton, *et al.*, "The EM algorithm for mixtures of factor analyzers," Technical Report CRG-TR-96-1, University of Toronto, Tech. Rep., 1996.

[191] A. L. Mitchell, M. Engelcke, O. P. Jones, D. Surovik, S. Gangapurwala, O. Melon, I. Havoutis, and I. Posner, "First steps: Latent-space control with semantic constraints for quadruped locomotion," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2020, pp. 5343–5350.

[192] D. Surovik, O. Melon, M. Geisert, M. Fallon, and I. Havoutis, "Learning an expert skill-space for replanning dynamic quadruped locomotion over obstacles," in *Proc. Conference on Robot Learning*, 2021.

[193] D. Holden, J. Saito, T. Komura, and T. Joyce, "Learning motion manifolds with convolutional autoencoders," in *SIGGRAPH Asia 2015 technical briefs*, 2015, pp. 1–4.

[194] T. S. Lembono, F. Suárez-Ruiz, and Q.-C. Pham, "SCALAR-simultaneous calibration of 2d laser and robot's kinematic parameters using three planar constraints," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2018, pp. 5570–5575.

[195] ——, "SCALAR: Simultaneous calibration of 2-d laser and robot kinematic parameters using planarity and distance constraints," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 4, pp. 1971–1979, 2019.

[196] H. Girgin, T. S. Lembono, R. Cirligeanu, and S. Calinon, "Optimization of robot configurations for motion planning in industrial riveting," in *Proc. IEEE Intl Conf. on Advanced Robotics (ICAR)*, 2021, pp. 247–252.

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

Acronyms

k-NN k-Nearest Neighbors. 33, 141

BGMR Bayesian Gaussian Mixture Regression. 5, 33

DDP Differential Dynamic Programming. 49

DoF Degree of Freedom. 4, 10

EE End Effector. 4, 10–12, 14

FDDP Feasibility-prone Differential Dynamic Programming. 18, 19, 46, 49

GAN Generative Adversarial Networks. 3, 6, 23, 25, 136, 138, 142

GMM Gaussian Mixture Model. 6, 23, 24, 137

GMR Gaussian Mixture Regression. 5, 50, 135, 141

GPR Gaussian Process Regression. 4, 5, 22, 33, 50, 135, 141

IK Inverse Kinematics. 4, 6, 10–12, 14

iLQR Iterative Linear Quadratic Regulator. 2, 5, 6, 16–19, 59, 64

KL Kullback-Leibler. 24

LQR Linear Quadratic Regulator. 16

MDN Mixture Density Networks. 22, 133, 138

MLP Multi-layer Perceptron. 4, 22, 138, 142

MPC Model Predictive Control. 3, 6, 16, 131, 134

OCP Optimal Control Problem. 15–17, 131

PCA Principal Component Analysis. 4

PDF Probability Density Function. i, 19, 59, 89, 107, 138, 141, 142

PRM Probabilistic Roadmap. 3, 12

QP Quadratic Programming. 15, 18

RBF Radial Basis Function. 4

RRT Rapidly-exploring Random Tree. 3, 12

TSID Task-Space Inverse Dynamics. 2, 15, 135

TT Tensor Train. 23

TTGO Tensor Train for Global Optimization. 91, 107

VAE Variational Auto Encoders. 3

TEGUH SANTOSO LEMBONO

Research Assistant

+41 76 390 92 37 | tarsle14@gmail.com

SUMMARY OF QUALIFICATIONS

Easy to collaborate with

Many collaborations with both internal group members and external researchers (LAAS, Edinburgh, A*Star, Oxford, MPI). Able to understand others and communicate clearly, both technical and non-technical.

Practical experience

Worked in many robotics projects involving various robots (Universal Robot, Denso, Franka Emika Panda, Anymal, Talos) that require a lot of system integration (integrating the computer vision, robot control, motion planning, motion capture). One published US patent on taping robot.

Research experience

Worked in robotics research for > 8 years and published > 15 research papers in prestigious conferences and journals (ICRA, IROS, RA-L, T-ASE). The current research topics focus on motion planning, optimal control, and machine learning.

Strong technical knowledge

Good understanding on both classical robotics theory (kinematics, dynamics, control, planning, estimation, optimization) and machine learning (learning from demonstration, supervised learning, Bayesian model, VAE, GAN).

LANGUAGE SKILL

Indonesian	English
French	German

COMPUTER SKILLS

ROS	Gazebo	Pybullet	OpenRave
Ubuntu	Tensorflow	Solid Work	ANSYS CFX
MATLAB	Python	C	C++

EDUCATION

PhD (Cand.), EPFL (École Polytechnique Fédérale de Lausanne), Switzerland — 2018-Now

Expected graduation date: July 2022.

M.Sc. Mechanical Engineering, National University of Singapore (NUS), Singapore — 2016

COURSES: Linear System, Computer Control System, Neural Network, Deep Learning for Computer Vision, Real Time System, Advanced Robotics, Computer Aided Product Design, Measure and Integration. **CGPA: 4.55/5**

B.Eng. Mechanical Engineering, Nanyang Technological University (NTU), Singapore — 2012

CORE COURSES: Dynamics and Control, Mechanism Design, Robotics, Microprocessors, Fluid Mechanics, Fluid Dynamics, Heat Transfer, Thermodynamics, Electrical & Electronics, Mechanics of Material, Engineering Design, Manufacturing.

CGPA: 4.8/5.0, First Class Honours

Completed Courses on Coursera (Certified)

UC SAN DIEGO: Data Structures and Algorithms Specialization (Data Structures, Algorithmic Toolbox, Algorithms on Strings, Algorithm on Graphs, Advanced Algorithms and Complexity).

UNIVERSITY OF AMSTERDAM: Basic Statistics and Inferential Statistics.

TOEFL Score: **113 out of 120** (TOEFL IBT).

GRE Score: **340 out of 340, and 4 score for AWA.**

WORK EXPERIENCE

July 2018 - Now — Research Assistant at Robot Learning and Interaction, Idiap Research Institute and PhD Candidate at EPFL (Dr. Sylvain Calinon)

- My PhD takes part in MEMMO Project (www.memmo-project.eu), where we aim to build a unifying controller for legged robots using optimal control and memory of motion. With precomputed dataset and machine learning algorithms, we aim to use optimal control in real time to control various legged robots (quadruped, biped, and exoskeleton). The project involves 10 partners from various renowned institutions.

July 2017 - July 2018 — Research Associate at Control Robotics Intelligence (CRI) Group, Nanyang Technological University (Assoc. Prof. Pham Quang Cuong)

- Worked with ROS and OpenRave to develop an automated robotic drilling solution which consists of an industrial robot (Denso), 3D camera, and laser scanner, and developed a new motion planning algorithm (RoboTSP) to optimize the drilling sequence.
- Developed SCALAR, a new calibration method using only a laser scanner attached on the robot arm and several calibration devices (a flat plate and a sharp tool tip) that can be easily manufactured.

July 2017 - July 2018 — Teaching Assistant at Nanyang Technological University

- Tutorial classes on “Introduction to Electrical Circuits” and “Control Theory”.

July 2016 - July 2017 — Research Assistant at Singapore University of Technology and Design (Assoc. Prof. Tan U-Xuan)

- Designed a state estimation algorithm for monocopter with Extended Kalman Filter.
- Designed a Human Machine Interface (HMI) to control multiple Unmanned Ground Vehicles (UGVs).
- Created a simulation program for UGV navigation (SLAM) and control using ROS, Gazebo, and Qt in C++.
- Designed experiments to evaluate the effectiveness of the HMI.

Nov 2013 - July 2016 — Project Officer at Robotics Research Center, NTU Singapore (Prof. Chen I-Ming)

- Implemented robot calibration algorithm as Rviz plugin.
- Integrated a robot taping system, which consists of 4 processes: 3D model generation, post processing, path planning, and execution on Universal Robot.
- Finalist in a robot waiter competition. Built a mobile robot to serve as a waiter from scratch. Designed the navigation system and integrated the whole software system.

July 2012 - Oct 2013 — Engineer at Excel Marco Singapore Pte. Ltd.

- Designed graphics for Human Machine Interface (HMI) to be used in FPSO
- Progammer for the topsides. Use Siemens PCS7 to create ICSS (Integrated Control and Safety System) Program
- Helped to conduct FAT with customer

Feb 2011 - June 2011 — Research Intern at Fraunhofer Institute of Production and Automation (IPA), Stuttgart, Germany

- Joined ROSETTA research project, a collaboration between seven European Institutes and universities.
- The research was about how to generate (ABB) robot programming codes automatically without the help of an experienced programmer

ACHIEVEMENTS AND HONORS

2021 — US Patent (Published) No. 10946526, Robot taping system and method of taping

2009 - 2012 — Dean’s List, Nanyang Technological University, Singapore

Top 5% student according to GPA

2009 - 2012 — President Research Scholarship (NTU)

Research funding for top 10% students

2007 — Bronze Medal in International Astronomy Olympiad 2007 in Ukraine