Trajectory Prediction with Compressed 3D Environment Representation using Tensor Train Decomposition

Lara Brudermüller*, Teguh Lembono*, Suhan Shetty, and Sylvain Calinon

***Abstract*— Trajectory optimization for motion planning and optimal control is a popular approach in robotics. Algorithms typically require good initialization in order to find the optimal trajectories. To provide such initialization, many approaches rely on the concept of *memory of motion*, where a function approximator is trained on a database of robot trajectories to predict good initial trajectories for novel situations, and hence speeding up the subsequent trajectory optimization process. To be able to generalize well to a new environment, an expressive environment descriptor is necessary. We propose to encode the environment by discretized signed distance functions (SDF) which are then compressed using a tensor train (TT) decomposition approach. In order to show the expressiveness of this low-rank TT-SDF representation, three function approximators are compared: k-nearest neighbors, a neural network, and a mixture density network. We demonstrate the proposed method with motion planning examples on two different systems (point mass and quadcopter). Our experiments demonstrate that the TT-SDF encoding can provide compact environment descriptors in order to predict good initial trajectories for warm-starting an optimal control solver.**

I. Introduction

Motion planning is one of the core problems in robotics, which often includes a myriad of constraints (e.g. joint limit, stability, and obstacle avoidance) besides the task of finding the optimal path between the start and goal configurations. Moreover, when applied to robots with many degrees of freedom (DoFs), motion planning approaches also need to handle high-dimensional problems. Sampling-based motion planning algorithms such as probabilistic roadmap (PRM) and rapidly-exploring random tree (RRT) [1], [2] are very successful in robotics due to their applicability in highly complex and high-dimensional environments. They typically employ a two-stage process of first finding a feasible path and then optimizing it. More recently, instead of first finding solely feasible paths, newer approaches, referenced as *trajectory optimization (TO)* methods, directly construct trajectories by formulating the problem as a single optimization problem which formalizes the physical constraints, the goal specification, as well as notions of quality and efficiency [3], [4]. Trajectory optimization methods have been shown to generate optimal motions for high-dimensional robots in complex environments. They are also used to solve optimal control problems that involve nonlinear dynamics. However,

This work was supported by the European Commission's Horizon 2020 Programme (MEMMO project, http://www.memmo-project.eu/, grant 780684).

The first two authors have equal contribution. All authors are with the Idiap Research Institute, Martigny, Switzerland (e-mail: firstname.lastname@idiap.ch). L. Brudermüller is now with the University of Oxford (e-mail: larab@robots.ox.ac.uk).

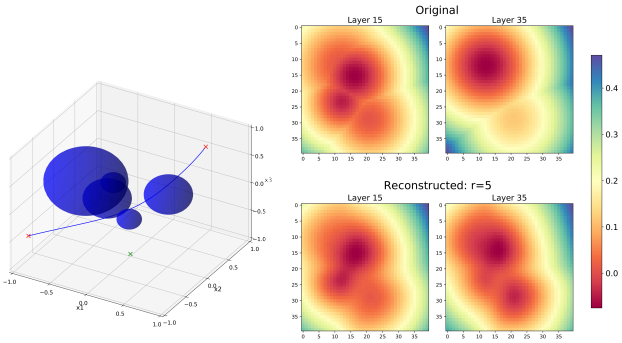

Fig. 1. *Left:* Example task scenario in an environment with five sphere obstacles of different radii and center positions. *Right:* Corresponding discretized SDF representation (uncompressed on the top, compressed on the bottom) at the selected z-slice plane. The colormap corresponds to the SDF-values at each voxel.

unlike sampling-based approaches, trajectory optimization approaches are highly sensitive to the initialization. They can get stuck in bad local optima when initialized far away from the optimal solution, especially for highly nonlinear problems. To address this issue, several approaches relying on the concept of *memory of motion* have been proposed [5], [6], [7]. A memory of motion is a dataset of precomputed motions that correspond to various tasks. By learning from such memory of motion, we can predict the initial guess of the optimal trajectories for the new task. Therefore, a function approximator is trained to learn the mapping $f : \boldsymbol{x} \rightarrow \boldsymbol{y}$ that maps each task $\boldsymbol{x}$ to the optimal robot trajectory $\boldsymbol{y}$. This problem has also been coined as *trajectory prediction* [8]. The prediction can then be used to initialize, or *warm start*, the trajectory optimizer. Lembono *et al.* propose an ensemble of function approximation methods to learn $f(\boldsymbol{x})$ [7], which is typically characterized by nonlinear and multi-modal mappings (i.e., there might be multiple robot paths to avoid the obstacle(s) and to reach the desired position in a scene). This paper extends the above work by focusing on finding an efficient representation of the environment (comprising the set of obstacles), such that the learning algorithm can generalize to a new environment. We propose to use *tensor train (TT) decomposition* to find the compact representation of the given 3D environment encoded into a discretized signed distance function representation.

In the context of environment representation, *signed distance functions (SDFs)*, also referred to as *signed distance fields*, have increasingly gained attention in robotics and computer graphics to represent 2D and 3D volumes [9], [10], 3D surface reconstruction from sensor data [11] and more

recently, also in optimization-based motion planners along with collision detection [3]. In [12], Oleynikova *et al.* discuss the advantages of using SDFs for online motion planning and online mapping. We propose to extend this approach by using the SDF representation as the environment descriptor for the memory of motion.

An SDF, as well as its truncated version (TSDF), are defined in 3D space and must therefore be stored in an appropriate 3D basis. An example for this discretized representation is shown in Fig. I. Most commonly, a 3D piecewise-constant (voxel) basis is used. However, *voxelized* SDFs as non-parametric volumetric representations of a continuous 3D mesh incorporate a high amount of redundancy that requires large memory storage, constraining their applicability in large-scale environments.[1] Previous work addressed this issue by means of different compression strategies [13], [14], [15] in various applications including real-time mapping and surface reconstruction. However, these compressed SDF representations have been barely explored in the context of motion planning and collision avoidance.

In [15], Boyko *et al.* applies low-rank *tensor train (TT) decomposition* for compression and operations on 3D objects and scenes represented by volumetric 3D SDFs. While their approach provides very efficient compression of the high-resolution SDF maps, the authors show that this representation is still powerful enough in terms of reconstruction, making it also useful for domains other than surface reconstruction, such as motion planning. The compression approach is also learning-less and mathematically simple, depending on only one single hyperparameter (*decomposition rank*). Building on this work, our work evaluates the use of such compressed representation as environment descriptor for a memory of motion.

A. Contributions

This paper extends the *TT-SDF* representation proposed in [15] to derive a compressed and expressive feature representation of the environment. This is used to predict initial trajectories for warm-starting trajectory optimization algorithms in a given environment. By additionally *orthogonalizing* the factors derived in the TT, a technique mathematically defined in the work of [16], the compressed features are more unique and expressive. The approach is demonstrated and tested on the motion planning of two increasingly complex systems (double-integrator point-mass system and quadcopter). Moreover, three different function approximators are applied to learn the task of trajectory prediction, taking the task descriptor as well as the TT-SDF environment descriptor as an input. We show that the proposed TT-SDF allows us to generalize the trajectory prediction to new environments involving multiple obstacles.

B. Overview

The paper is organized as follows. Section II covers the necessary mathematical and conceptual background on tensor decomposition methods and optimal control for a better understanding of the remainder of this work. Section III then explains how to use TT-decomposition to obtain a good representation of the environment, which is then used to predict an initial guess for warm starting the trajectory optimizer. Section IV evaluates the approach on two different systems surrounded by multiple obstacles, followed by the final conclusion in Section V.

[1]For instance, a space of $20 \times 20 \times 4$ m^3 discretized into voxels of 2 cm size takes up to 800 MB at 32 bits per voxel [13].

II. Background

A. Tensor decomposition

A tensor is a multidimensional array. It is a generalization of vectors and matrices to higher dimensions. The order of a tensor is the number of modes in the array. For instance, a vector is a first order tensor and a matrix is a second order tensor. We denote the $\boldsymbol{k}$-th elements of a d-th order tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{K_1 \times \cdots \times K_d}$, with indices $\boldsymbol{k} = (k_1, \ldots, k_d)$ and $k_i \in \{1, \ldots, K_i\}$, by $\boldsymbol{\mathcal{G}}_{\boldsymbol{k}}$. Here, $K_i \in \mathbb{Z}^+$ represents the size of i-th mode, with $i \in \{1, \ldots, d\}$.

Tensor factorization (or decomposition) techniques generalize matrix decomposition techniques (e.g. matrix singular value decomposition) to multidimensional arrays in order to compactly represent a multidimensional array. These techniques use a set of lower-order tensors (called factors) to represent the given tensor. The original tensor is reconstructed by applying algebraic operations on these factors. The type of factors and the algebraic operations that are used vary depending on which tensor decomposition technique is used. The accuracy of the representation is controlled by the *rank* of the tensor decomposition. If the tensor has some low-rank structure (e.g., arising from smoothness or parallel-proportionality), the corresponding representation of the tensor will typically be low-dimensional.

Some of the popular tensor decomposition techniques are *canonical polyadic* (CP), *Tucker*, and *tensor train* (TT) (cf. [17] for a survey). The CP decomposition scales well to high-order tensors. However, the problem of finding the best CP decomposition for a given tensor is not always a well-posed optimization problem as the space of all CP tensors with a fixed rank does not need to be a closed space. The Tucker decomposition overcomes this issue but suffers from the curse of dimensionality [18]. Tensor Train (TT) decomposition [19] is a recent popular decomposition technique that also scales to high-order tensors with fast and robust algorithms for finding the TT decomposition. Moreover, for any given tensor, we can always find a TT decomposition (1) [19]. In the remaining sections, based on our previous work exploiting TT for ergodic control [20], we will use TT as the tensor decomposition technique.

Let $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{K_1 \times \cdots \times K_d}$ be a d-th order tensor in TT format. Its TT format is defined by a tuple of d third-order tensors $(\boldsymbol{\mathcal{G}}^1, \ldots, \boldsymbol{\mathcal{G}}^d)$. Here, $\boldsymbol{\mathcal{G}}^i \in \mathbb{R}^{r_{i-1} \times r_i \times K_i}$, $i \in \{2, \ldots, d-1\}$, $\boldsymbol{\mathcal{G}}^1 \in \mathbb{R}^{1 \times r_1 \times K_1}$ and $\boldsymbol{\mathcal{G}}^d \in \mathbb{R}^{r_{d-1} \times 1 \times K_d}$. As shown in Fig. 2, the $\boldsymbol{k}$-th elements, with $\boldsymbol{k} \in \mathcal{K} = \{(k_1, \ldots, k_d) \colon k_i \in \{1, \ldots, K_i\}, i \in \{1, \ldots, d\}\}$, is given by

$$\boldsymbol{\mathcal{G}}_{\boldsymbol{k}} = \boldsymbol{\mathcal{G}}^1_{:,:,k_1} \boldsymbol{\mathcal{G}}^2_{:,:,k_2} \cdots \boldsymbol{\mathcal{G}}^d_{:,:,k_d}, \tag{1}$$

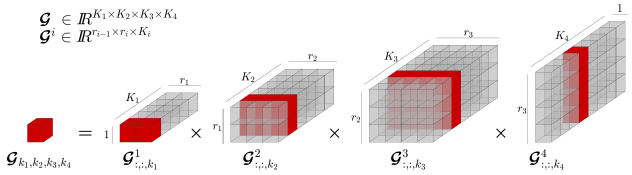

Fig. 2. The figure shows an example for a 4th order tensor $\mathcal{G} \in \mathbb{R}^{5\times6\times7\times8}$ of rank $\boldsymbol{r} = (2, 3, 4)$. We can evaluate an element of a tensor in TT format by multiplying the slices (matrices represented in red color) of the factors (3D arrays)).

where $\mathcal{G}^i_{:,:,k_i} \in \mathbb{R}^{r_{i-1}\times r_i}$ represents the k_i-th frontal slice (a matrix) of the third order tensor $\mathcal{G}^i$. The *TT-rank* of the tensor in TT representation is then defined as the tuple $\boldsymbol{r} = (r_1, r_2, \ldots, r_{d-1})$ and the *maximal TT rank* is defined as $r = \max(r_1, \ldots, r_{d-1})$. Let $K = \max(K_1, \ldots, K_d)$, then the TT representation uses $\mathcal{O}(Kdr^2)$ parameters to represent the tensor $\mathcal{G}$ which has $\mathcal{O}(K^d)$ elements. Hence, the TT representation is very efficient if the maximal rank r is low, which is often the case in engineering applications.

In this work we use the TT decomposition to compress the SDF tensor. However, it should be noted that the tensor decomposition factors of a fixed rank for a given tensor is not unique and that the factors possess several degrees of freedom to represent the same tensor. While it does not pose a problem for reconstruction [15], a unique envionment representation is crucial if we want to use it as environment descriptor for trajectory prediction. We therefore use a canonical representation, namely right-orthogonalized representation [16], and denote the resulting representation as TT-SDF.

B. Optimal control

In the broader context of this paper, we want to tackle the problem of finding the solution of an optimal control problem (OCP), i.e. finding the optimal control inputs and state trajectories $(\boldsymbol{x}^*, \boldsymbol{u}^*)$ that minimize a given cost function with regard to the system dynamics. A general discrete OCP consists of a cost function of the form

$$C(\boldsymbol{x}, \boldsymbol{u}) = \sum_{t=1}^{T} c_t(\boldsymbol{x}_t, \boldsymbol{u}_t) \quad \text{s.t. } \boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t). \tag{2}$$

This basic formulation may also include additional equality and inequality constraints, depending on the problem setting. Despite the term *control* in its name, optimal control formulation is also often used to plan the motion of a robotics system while considering its dynamics. In robotics, given the typically high complexity of the system including many DOFs, the problem is most commonly solved using numerical optimization methods such as the *iterative linear quadratic regulator (iLQR)* algorithm [21]. It is often chosen due to its computational efficiency. It iteratively approximates the cost function and dynamics to be quadratic and linear, respectively, and consequently solves an LQR subproblem at each step. It demonstrated good performance even for high-dimensional systems such as quadruped or humanoid robots. However, it often requires good initial guesses in order to obtain satisfactory performance [6].

C. Signed Distance Function

An SDF is an *implicit* representation of a surface, embedding geometry into a scalar field whose value represents the distance to the *nearest* surface of the embedded geometry. The sign of the SDF values provides information about the field location with respect to the nearest surface, assuming positive values outside the geometry, i.e. in free space, and negative values inside. The object surfaces themselves are given by the *level-set* or *iso-surface* $S = \{\boldsymbol{p} : f(\boldsymbol{p}) = 0\}$ where $f(\boldsymbol{p})$ denotes the SDF evaluated at a point p.

III. Method

A. Setup

Fig. 3 provides an overview of the entire motion planning pipeline developed in this work for learning, optimization and execution. Our main objective is, given a new task and an environment (comprising the obstacles), we want to provide good initial trajectories for warm starting the OCP solver. To do that, we first build a database of optimal trajectories offline by solving many motion planning problems with various tasks in different environments using the OCP solver. We use the proposed TT-SDF representation as the environment descriptor, and train a function approximator on the dataset. Given a new task and environment, the function approximator then predicts the initial guess of the trajectories, which is used to warm start the OCP solver. The following sections describe how the dataset is created, the formulation of the learning problem, and the function approximators considered in this work.

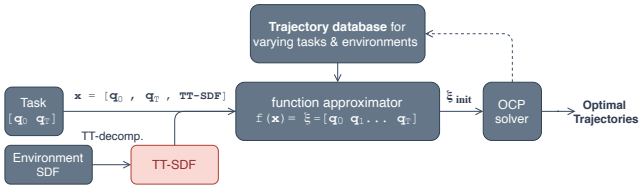

Fig. 3. Overview of the motion planning system used for learning, optimization and execution.

B. Creation of trajectory database

To learn the mapping $f(\boldsymbol{x}) = \boldsymbol{y}$ where $\boldsymbol{x}$ is the task and $\boldsymbol{y}$ is the corresponding trajectory, we first build a dataset $D = \{\boldsymbol{x}_i, \boldsymbol{y}(\boldsymbol{x}_i)\} = (\boldsymbol{X}, \boldsymbol{Y})$. Here, $\boldsymbol{x}_i$ consists of the motion planning task (the desired initial and goal location) as well as the compressed environmental representation, i.e. our TT-SDF encoding, and $\boldsymbol{y}(\boldsymbol{x}_i)$ denotes the corresponding trajectory. We sample N tasks, and compute the output $\boldsymbol{y}(\boldsymbol{x}_i)$ using the iLQR solver offline.

C. Construction and decomposition of input and output

A typical scenario in motion planning is a workspace filled with obstacles. A given input for the function approximator should therefore not only include the task setting itself, i.e.

start and goal robot posture, but also some notion of the robot surrounding, i.e. obstacles in the scene. The generalization ability of a learnt approximator crucially depends on the environment representation. Therefore, this work constructs the input vector $\boldsymbol{x}$ by including the TT-SDF representation in the form of parameters $\boldsymbol{\theta}$, i.e. $\boldsymbol{x} = [\boldsymbol{q}_0, \boldsymbol{q}_T, \boldsymbol{\theta}]^\top$, where $(\boldsymbol{q}_0, \boldsymbol{q}_T)$ are the initial and goal locations.

To represent the environment, we first construct a voxel grid with a fixed boundary, comprising of n_v^3 voxels. At each voxel location $\boldsymbol{q} = (q_1, q_2, q_3) \in \mathbb{R}^3$, we compute the SDF function $f : \mathbb{R}^3 \to \mathbb{R}$ that maps $\boldsymbol{q}$ to the signed distance between $\boldsymbol{q}$ and the closest point on the surface of the obstacle. This gives us a 3-rd order SDF tensor with n_v^3 elements. We then use TT decomposition to compress the SDF tensor to low rank TT factors in the canonical representation, which are then flattened to a single vector that we denote as TT-SDF. Parameters influencing the dimensionality of the resulting TT-SDF are most importantly the *decomposition rank* of the TT, as well as the *voxel resolution* which will be elaborated in Section IV.

To compute the SDF at each point, we are using the SDF library[2] for standard object primitives such as spheres, boxes, or capsules. For more general objects, we can compute the SDF from the mesh data[3].

The output vector $\boldsymbol{y} \in \mathbb{R}^{DT}$ (the path trajectories) contains T data points of dimensionality D. In a dataset of N samples, DT might be much larger than N, which will lead to a poor performance of the learning algorithm due to the curse of dimensionality. Therefore, $\boldsymbol{y}$ is compressed using standard *principal component analysis* (PCA).

D. Function Approximators

We consider three function approximators for the trajectory prediction, i.e. to learn the mapping $f : \boldsymbol{x} \to \boldsymbol{y}$.

1) *k-Nearest neighbor (k-NN):* Given a task $\boldsymbol{x}^*$, the algorithm finds the nearest sample in the database according to a chosen distance metric (Euclidean in this paper). It is simple to implement and typically performs very well in datasets with high variability. However, it suffers from the curse of dimensionality, since the required number of data grow exponentially with the increasing input dimension. Therefore, this approximator only serves as a baseline comparison.
2) *Artificial neural network (NN):* Neural networks are very popular, given their flexibility and ease of use for learning from the dataset. All networks are implemented with two hidden densely connected layers of 256 units per layer and a ReLu activation function. The learning rate, as well as the number of neurons has been manually tuned. Moreover, some of the experiments included a dropout layer before the output layer. Finally, the networks are trained using the mean squared error as regression loss function.
3) *Mixture density network (MDN):* As discussed in [7], multi-modality poses a big challenge to a standard function approximator such as NN. Mixture Density Networks (MDN) [22] solve this issues 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 multi-modal functions more precisely. The network is trained using the negative log-likelihood.

IV. Experimental Evaluation

In this section, the results and insights obtained from the experiments with a point-mass system and a quadcopter are reported[4]. The point mass state space is 6-dimensional, including position and velocity, whereas the control input is 3-dimensional, i.e. acceleration command for each dimension. The quadcopter has a 12-dimensional state space (position, orientation, and the corresponding velocity) and a 4-dimensional control input, i.e. the torque at each propeller. The workspace boundary for both systems is set to be $[-1, 1]$ m for each of the xyz axes. The expressiveness and the usability of the proposed environment descriptor for trajectory prediction were evaluated using eight different trajectory datasets, as well as three different function approximators f.

A. Trajectory databases

When creating the trajectory datasets, different scenarios were considered. The major dimensions along which the datasets differ are the following:

1) *Fixed or varying init/goal* determines whether the start and goal position of each sample are fixed or vary in a given range. In the fixed case, $\boldsymbol{q}_0 = [-1, -1, -1]$ and $\boldsymbol{q}_T = [1, 1, 1]$. In the varying case, $\boldsymbol{q}_0$ and $\boldsymbol{q}_T$ are sampled around the two opposing workspace corners.
2) *Include bias:* The presence of multi modality in the data poses a challenge in learning the trajectory prediction. To separate this effect from the TT-SDF evaluation, in some datasets we initialize iLQR by a trajectory that goes from $\boldsymbol{q}_0$ to $\boldsymbol{q}_T$ through an additional way point ($\boldsymbol{q}_{T/2} = [0, 0, -1]$), to bias the trajectories to pass through the lower area of the environment so that the multi modality is reduced. We denote datasets which do not include way points by the symbol ✗.
3) *Number of obstacles:* determines whether an environment includes one or multiple obstacles. The obstacle positions and sizes are varied in a given range.
4) *Type of obstacles:* determines whether the obstacles are spheres only, or various shapes (in this paper we consider spheres, boxes, and capsules).
5) *Dynamical system:* we consider two systems, a point mass and a quadcopter. The point mass dynamics are formulated as a double integrator system, i.e. linear dynamics controlled with acceleration command, while the quadcopter has nonlinear and underactuated dynamics, resulting in a more complex system.

[2] https://github.com/fogleman/sdf

[3] https://pypi.org/project/mesh-to-sdf/

[4] The implementation codes are available at https://github.com/teguhSL/tt_sdf

TABLE I

OVERVIEW OF DATASETS AND CORRESPONDING RESULTS FOR TT-RANK 3 WITH VOXEL RESOLUTION 0.05

	Dataset	**Point Mass**						**Quadcopter**	
		i	ii	iii	iv	v	vi	vii	viii
	Start/goal	fixed	var.	var.	fixed	var	var.	var	var.
	Way-point	$[0, 0, -1]$	$[0, 0, -1]$	✗	✗	$[0, 0, -1]$	✗	✗	✗
	Num. obstacles	$\{1\}$	$\{1\}$	$\{1\}$	$\{3, 4, 5\}$	$\{3, 4, 5\}$	$\{3, 4, 5\}$	$\{1\}$	$\{3\}$
	Type of obstacles	Sphere	Sphere	Sphere	Sphere	Sphere	Sphere	Various	Various
Model	**Metric**								
k-NN	$\text{MSE}_{\text{total}}(\text{m}^2)$	0.0290	0.1037	0.1127	0.0358	0.1005	0.0931	0.0304	0.0446
	$\text{MSE}_{\text{goal}}(\text{m}^2)$	0.0000	0.0753	0.0793	0.0000	0.0658	0.0644	0.0166	0.0208
	Collision free (%)	70.00	84.67	75.86	36.67	46.39	36.39	62.62	36.25
NN	$\text{MSE}_{\text{total}}(\text{m}^2)$	0.0079	0.0151	0.0209	0.0194	0.0420	0.0367	0.0104	0.0172
	$\text{MSE}_{\text{goal}}(\text{m}^2)$	0.0000	0.0017	0.0029	0.0000	0.0139	0.0112	0.0005	0.0020
	Collision free (%)	87.33	77.0	58.62	35.33	33.68	34.69	52.10	24.47
MDN	$\text{MSE}_{\text{total}}(\text{m}^2)$	0.0133	0.0289	0.0459	0.0271	0.0741	0.0701	0.0353	0.0413
	$\text{MSE}_{\text{goal}}(\text{m}^2)$	0.0000	0.0055	0.0176	0.0000	0.0328	0.0292	0.0159	0.0188
	Collision free (%)	92.00	90.58	92.34	65.67	82.82	80.61	75.70	67.37

TABLE II

EVALUATING THE EFFECT OF TT-RANK r ON THE TRAJECTORY PREDICTION FOR DATASET II

TT rank	2	3	5	10	raw
dim (flat)	320	600	1400	4800	64000
$\text{MSE}_{\text{total}}(\text{m}^2)$	**0.0133**	0.0151	0.0158	0.0195	0.0546
$\text{MSE}_{\text{goal}}(\text{m}^2)$	**0.0011**	0.0017	0.0029	0.0028	0.0417
collision free (%)	75.9	**77.0**	77.0	75.1	31.0

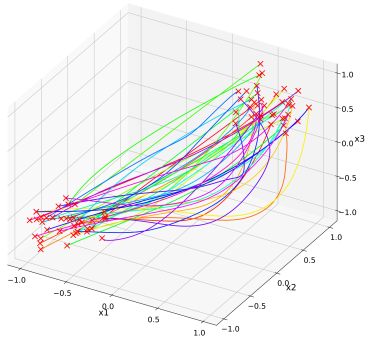

Fig. 4. An extract of the trajectories derived for environments containing 3 to 5 randomly sampled sphere obstacles of different radii ℓ at different positions $\boldsymbol{p}$, as shown in the legend (this corresponds to dataset 6).

For each scenario, a set of 1000 samples were sampled in the task space as well as the obstacle space and the corresponding trajectories were computed using iLQR. 70% of the dataset is used for training the function approximator while the remaning part is used for the evaluation/testing. An example of a test environment is shown in Fig. I, picturing a trajectory sample for the multiple-obstacle case with a way-point, shown in green. Moreover, Fig. 4 shows an excerpt of the trajectories generated for the case of three to five obstacles in a scene with varying task settings and no predefined way-point, corresponding to dataset number vi in Table I. Note that the obstacles are not shown in this figure.

B. Analysis

The performance of the learnt function approximators is evaluated in terms of standard regression metrics, such as the *mean squared error (MSE)*, but also with regard to *collision avoidance* and their warm-starting performance. We consider two MSE: $\text{MSE}_{\text{total}}$ and MSE_{goal}. The first refers to the MSE of the whole predicted trajectory as compared to the ground truth, while the later refers to the MSE with respect to only the last time step. The obstacle avoidance performance is evaluated by running collision tests for each point in the generated trajectory. As soon as one point in the trajectory collides with the obstacle, the entire trajectory is considered to be in collision. On a test set, we evaluate how many tasks that the algorithm is able to predict collision-free trajectories, and calculate the percentage. Lastly, the method is evaluated in terms of its effectiveness in warm-starting the OCP solver, by looking at the corresponding OCP cost values at different iLQR iterations.

For the MDN, the evaluation is done a bit differently from the methods. The output of MDN is not a single prediction but a mixture of Gaussians, each of which corresponds to a different mode. To evaluate its performance, we first sample K trajectories from the mixture of Gaussians. We then compute the iLQR cost of each trajectory, and select the one with the lowest cost to be the final prediction of MDN. Since iLQR cost computation is usually much faster than one iLQR iteration, this sampling step does not affect the computation time significantly. By doing this, the MDN performance is boosted significantly (as compared to only sampling one trajectory). We choose $K = 10$.

a) Decomposition rank: The expressiveness of the environment descriptor, i.e. the TT-SDF depends on the degree of compression and how much information is lost versus how much redundancy is removed by it. In the TT decomposition this depends on the decomposition rank r. The smaller the rank, the higher the compression. Therefore, we first evaluate the TT-SDF representation on a range of compression ranks, i.e. $r \in \{2, 3, 5, 10\}$. Analyzing the representations themselves outside the regression context, Fig. I shows an example scenario (*left*) and the corresponding SDF representation, compared to the TT-SDF using $r = 5$ (*right*). The discretized SDFs are shown for two different layers of the 3D grid structure sliced along the z-axis. Comparing the two, we can see that a low-rank compression ($r = 5$) is still able

TABLE III

WARM-STARTING PERFORMANCE FOR ILQR SOLVER WITH NN AND MDN AS COMPARED TO STANDARD INITIALIZATION.

Dataset	**Point Mass**						**Quadcopter**					
	iii (single obst.)			vi (mult. obst.)			vii (single obst.)			viii (mult. obst.)		
iLQR iteration	5	10	50	5	10	50	5	10	50	5	10	50
Cold start	215.679	19.559	1.299	101.874	25.918	1.421	6.297	3.148	1.964	5.296	2.510	2.078
NN	6.332	**1.342**	**1.297**	11.512	**1.422**	**1.148**	2.105	1.951	**1.936**	2.134	**2.028**	2.007
MDN	**1.514**	1.376	1.311	**3.455**	1.644	1.152	**2.039**	**1.951**	1.956	**2.107**	2.086	**2.005**

to maintain the obstacle information to a sufficient degree. A more thorough comparison of how the TT-SDF performs within the neural network across different ranks is shown in Table II. The number of parameters associated with the TT-SDF representations at each rank is given in the first row. The evaluation shown here has been performed for NN on dataset ii with a voxel resolution of 0.05, i.e., each voxel is a cube with side of 5cm which results in 40 voxels across each dimension for an environment of $[2\times2\times2]\text{m}^3$. Note, that the choice of voxel resolution also impacts the dimensionality of the feature space. In general, it needs careful consideration to which extent the resolution is increased, since smaller voxel cubes do not necessarily capture more information, but rather increase dimensionality along with redundancy.

In our experiments, a 0.05 resolution show the best results in combination with the TT decomposition. It can be seen in Table II that TT-SDF representation clearly outperforms its uncompressed counterpart, especially when $r = 3$. This shows that a low rank can capture sufficient information of the environment while filtering out redundancies, enabling efficient learning. Note that by using $r = 3$, we reduce the environment parameters by a factor of $64000/320 = 200$ times, while improving the performance. Moreover, we have also compared the canonicalized TT-representation against the CP decomposition. Besides the significantly longer computation time of the feature representation, CP decomposition also seem to be less expressive for the trajectory prediction as compared to TT. Due to constraints in terms of paper length, we do not present these results.

b) Predictive performance of approximators: We then use TT-SDF as part of the input for the trajectory prediction. The TT rank r has been set to 3 with a voxel resolution of 0.05. Table I concatenates all results in terms of MSE and collision avoidance on eight different datasets using k-nearest neighbors (k-NN) with $k = 1$, a neural network (NN), and a mixture density network (MDN).

When looking at k-NN baseline results, it can be seen that it performs very well in terms of collision avoidance across most datasets with single obstacle. This is due to the formulation of the problem itself: the distance metric naturally puts more importance on TT-SDF features which outnumber the start and goal position. Therefore, the latter are almost ignored in the prediction process, which is also highlighted by the fact that the MSE in the goal position is usually worse as compared to the other predictors. For the fixed start and goal position and one obstacle scenario, i.e., dataset 1, NN even outperforms the nearest neighbor algorithm. In the NN setting, the introduction of a way-point in the dataset generation seems to alleviate the problems with the multi-modal property of the task-setting. This is also emphasized by the fact that the MDN clearly outperforms the other two approximators in terms of collision avoidance. Especially, for the more complex environments with multiple obstacles and without way-point, predicting at the highest 92% of the trajectories without any collisions for dataset iii. Its performance for both MSE metrics is lower than NN. In a practical consideration, however, predicting a trajectory which is feasible and collision-free would be more important than minimizing the MSE. Additionally, $\text{MSE}_{\text{total}}$ is not a good metric when the data is multi-modal, because the method may predict a good trajectory that has a low cost, but corresponds to the different mode from the current ground truth. In that case, the MSE would be very bad, although the predicted trajectory is actually close to optimal.

c) Warm-start performance: Finally, we compare the warm starting performance on the test set of dataset iii and vi for the point-mass system, and dataset vii and viii for the quadcopter. The corresponding OCP costs are evaluated after the iLQR iterations 5, 10 and 50, to observe the evolution of the costs. The costs after initializing with the predictions of the NN and MDN are compared against a standard initialization (straight line trajectory for the point mass system, and stationary trajectory for the quadcopter).

There is one additional detail about using the prediction output to perform the warm start that needs to be explained here. Despite the good performance of NN and MDN, the predicted trajectories generally do not start exactly from the desired initial state. When using this prediction output directly to warm start the OCP solver, it often results in a high cost due to this discrepancy. To overcome this issue, we perform an additional processing stage on the prediction. Specifically, we use Linear Quadratic Tracking (LQT) to generate trajectories that start from the desired initial state and end at the desired final state, while tracking the predicted trajectories from the function approximators. We can consider this as an additional smoothing step to ensure that the warm start begins at the correct desired initial state. In practice, this additional step is very important for the warm start to be successful, while the additional computation time is significantly less than one iLQR iteration. Finally, we note that we only predict the state trajectories, but the OCP solver requires also the control input as the initialization. For the point mass system, the control input can be easily calculated from the state trajectories, but for the quadcopter, we compute the control sequence from the state trajectories using the quasi-static assumption. As the resulting initializa-

tion is not dynamically consistent, we use the iLQR solver *Crocoddyl* [23] that accepts infeasible initialization.

The result overview is shown in Table III. As expected, the costs in the MDN and NN settings already start at a lower level as opposed to the standard initialization. As the number of iteration increases, the difference between the methods reduces, as most of them converge to the same local optima. In practice, using NN and MDN to warm start the OCP solver enable us to stop the iteration at around 5-10 iterations, since the cost is already sufficiently low (it corresponds to a feasible solution that is collision free). Comparing the results on the quadcopter and the point mass gives us interesting insights. In the quadcopter case, we observe that NN performs similarly to MDN. Interestingly, while quadcopter is a more complex system in terms of the dynamics, the corresponding trajectories have less multi-modality than the point mass. This indeed makes sense, because the complex dynamics provide more constraints to the motion planning, resulting in less variability, while simpler dynamics enable the solution to be more diverse.

V. Conclusion

This work presented an approach to guide an optimal control solver by initializing the search (i.e., *warm starting*) via trajectories that are predicted by function approximators that take the task and environment descriptors as inputs. The major contribution of this work was to demonstrate the expressiveness of describing the environment by means of a discretized SDF representation which is further compressed by a canonicalized version of the tensor train decomposition (TT-SDF). This compressed representation of the environment is able to provide meaningful and expressive features to the learning algorithms mapping from a given task and variable environment to a set of optimal trajectories. The method has been evaluated on two systems, a point mass and a quadcopter. In both cases, the predicted warm start significantly decreased the initial costs of an OCP solver, enabling faster convergence. The latter plays a specifically important role in the case of dynamic obstacles requiring fast re-planning of the trajectory throughout the task execution. This work opens up a meaningful direction for future research in using SDF functions in combination with tensor decomposition in the field of motion planning. For instance, further experiments could apply this method to a robot manipulator, a task setting which adds more constraints to the OCP formulation, impeding the search for an optimal trajectory. Moreover, the rank parameter of the TT-decomposition, while still being a hyperparameter, could be estimated in a separate learning algorithm. Additionally, different ranks along different dimensions could be considered.

References

[1] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[2] J. J. Kuffner and S. M. La Valle, "RRT-connect: an efficient approach to single-query path planning," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, vol. 2, 2000, pp. 995–1001.

[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, 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, 6 2014.

[5] N. Mansard, A. Delprete, 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.

[6] E. Dantec, R. Budhiraja, A. Roig, T. Lembono, G. Saurel, O. Stasse, P. Fernbach, S. Tonneau, S. Vijayakumar, S. Calinon, M. Taix, and N. Mansard, "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.

[7] 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, 2020.

[8] N. Jetchev and M. Toussaint, "Fast motion planning from experience: Trajectory prediction for speeding up movement generation," *Autonomous Robots*, vol. 34, no. 1-2, pp. 111–127, 2013.

[9] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones, "Adaptively sampled distance fields: A general representation of shape for computer graphics," in *Proc. ACM Intl Conf. on Computer graphics and Interactive Techniques (SIGGRAPH)*, 2000, pp. 249–254.

[10] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "Deepsdf: Learning continuous signed distance functions for shape representation," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 165–174.

[11] R. Chabra, J. E. Lenssen, E. Ilg, T. Schmidt, J. Straub, S. Lovegrove, and R. Newcombe, "Deep Local Shapes: Learning Local SDF Priors for Detailed 3D Reconstruction," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2020, pp. 608–625.

[12] H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. Nieto, and R. Siegwart, "Signed Distance Fields: A Natural Representation for Both Mapping and Planning," *Proc. Robotics: Science and Systems (RSS)*, 2016.

[13] D. R. Canelhas, E. Schaffernicht, T. Stoyanov, A. J. Lilienthal, and A. J. Davison, "Compressed voxel-based mapping using unsupervised learning," *Robotics*, vol. 6, no. 3, 6 2017.

[14] D. Tang, S. Singh, P. A. Chou, C. Häne, M. Dou, S. Fanello, J. Taylor, P. Davidson, O. G. Guleryuz, Y. Zhang, S. Izadi, A. Tagliasacchi, S. Bouaziz, and C. Keskin, "Deep implicit volume compression," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 1290–1300.

[15] A. I. Boyko, M. P. Matrosov, I. V. Oseledets, D. Tsetserukou, and G. Ferrer, "TT-TSDF: Memory-Efficient TSDF with Low-Rank Tensor Train Decomposition," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2020, pp. 10 116–10 121.

[16] T. Prosen, "Note on a canonical form of matrix product states," *Journal of Physics A: Mathematical and General*, vol. 39, no. 22, pp. 357–360, 2006.

[17] 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.

[18] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[19] I. V. Oseledets, "Tensor-train decomposition," in *SIAM Journal on Scientific Computing*, vol. 33, no. 5, 2011, pp. 2295–2317.

[20] S. Shetty, J. Silvério, and S. Calinon, "Ergodic Exploration using Tensor Train: Applications in Insertion Tasks," *IEEE Trans. on Robotics (in press, arXiv preprint:2101.04428)*, 2021.

[21] W. Li and E. Todorov, "Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems," in *ICINCO (1)*, 2004, pp. 222–229.

[22] C. M. Bishop, "Mixture Density Networks," Aston University, Tech. Rep., 1994.

[23] 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.