Fast Motion Planning for High-DOF Robot Systems Using Hierarchical System Identification

Biao Jia* Zherong Pan* Dinesh Manocha

***Abstract*— We present an efficient algorithm for motion planning and controlling a robot system with a high number of degrees-of-freedom (DOF). These systems include high-DOF soft robots and articulated robots interacting with a deformable environment. We present a novel technique to accelerate the evaluations of the forward dynamics function by storing the results of costly computations in a hierarchical adaptive grid. Furthermore, we exploit the underactuated properties of the robot systems and build the grid in a low-dimensional space. Our approach approximates the forward dynamics function with guaranteed error bounds and can be used in optimization-based motion planning and reinforcement-learning-based feedback control. We highlight the performance on two high-DOF robot systems: a line-actuated elastic robot arm and an underwater swimming robot in water. Compared to prior techniques based on exact dynamics evaluation, we observe one to two orders of magnitude improvement in the performance.**

I. Introduction

High-DOF robot systems are increasingly used for different applications. These systems include soft robots with deformable joints [1], [2], which have a high-dimensional configuration space, and articulated robots interacting with highly deformable objects like cloths [3], [4] or deformable environments like fluids [5], [6]. In these cases, the number of degrees-of-freedom (DOF $\mathbb{C}$, $N = |\mathbb{C}|$) can be more than 1000. As we try to satisfy dynamics constraints, the repeated evaluation of forward dynamics of these robots becomes a major bottleneck. For example, an elastically soft robot can be modeled using the finite-element method (FEM) [7], which discretizes the robot into thousands of points. However, each forward dynamics evaluation reduces to factorizing a large, sparse matrix, the complexity of which is $\boldsymbol{o}(N^{1.5})$ [8]. An articulated robot swimming in water can be modeled using the boundary element method (BEM) [5] by discretizing the fluid potential using thousands of patches on the robot's surface. In this case, each evaluation of the forward dynamics function involves inverting a large, dense matrix, the complexity of which is $\boldsymbol{O}(N^2\mathbf{log}(N))$ [9].

The high computational cost of forward dynamics becomes a major bottleneck for dynamics-constrained motion planning and feedback control algorithms. To compute a feasible motion plan or optimize a feedback controller, these algorithms typically evaluate the forward dynamics function hundreds of times per iteration. For example, a sampling-based planner [10] evaluates the feasibility of a sample using a forward dynamics simulator. An optimization-based planner [11] requires the Jacobian of the forward dynamics function to improve the motion plan during each iteration. Finally, a reinforcement learning algorithm [12] must perform a large number of forward dynamics evaluations to compute the policy gradient and improve a feedback controller.

Several methods have been proposed to reduce the number of forward dynamics evaluations. For sampling-based planners, the number of samples can be reduced by learning a prior sampling distribution centered on highly successful regions [13]. For optimization-based planners, the number of gradient evaluations can be reduced by using high-order convergent optimizers [14]. Moreover, many sampling-efficient algorithms [15] have been proposed to optimize feedback controllers. However, the number of forward dynamics evaluations is still on the level of thousands [14] or even millions [15].

Another method for improving the sampling efficiency is system identification [16], [17] that approximates the exact forward dynamics model with a surrogate model. A good surrogate model should accurately approximate the exact model while being computationally efficient [18]. These methods are mostly learning-based and require a training dataset. However, it is unclear whether the learned surrogate dynamics model is accurate enough for a given planning task. Indeed, [19] noticed that the learned dataset could not cover the subset of a configuration space required to accomplish the planning or control task.

Main Results: In this paper, we present a method of system identification for high-DOF robot systems. Our key observation is that, although the configuration space is high-dimensional, these robot systems are highly underactuated, with only a few controlled DOFs. The number of controlled DOFs typically corresponds to the number of actuators in the system and applications tend to use a small number of actuators for lower cost [20], [21]. As a result, the state of the remaining DOFs can be formulated as a function of the few controlled DOFs, leading to a function $\mathbf{f} : \mathbb{C}_c \to \mathbb{C}$, where $\mathbb{C}_c$ is the space of the controlled DOFs. Since $\mathbb{C}_c$ is low-dimensional, sampling in $\mathbb{C}_c$ does not suffer from a curse-of-dimensionality. Therefore, our method accelerates the evaluations of $\mathbf{f}$ by precomputing and storing $\mathbf{f}$ on the vertices of a hierarchical grid. The hierarchical grid is a high-dimensional extension of the octree in 3D, where each parent node has $2^{|\mathbb{C}_c|}$ children. This hierarchical data structure has two desirable features. First, the error due to our approximate forward dynamics function can be bounded. Second, we construct the grid in an on-demand manner, where new sample points are inserted only when a motion planner requires more samples. As a result, the sampled dataset covers exactly the part of the configuration space required by the given motion planning task and the construction of

* indicates joint first author

Biao Jia* is with the Department of Computer Science, University of Maryland at College Park. E-mail: biao@cs.umd.edu

Zherong* is with the Department of Computer Science, University of North Carolina at Chapel Hill. E-mail:zherong@cs.unc.edu

Dinesh Manocha is with the Department of Computer Science and Electrical & Computer Engineering, University of Maryland at College Park. E-mail:dm@cs.umd.edu

the hierarchical grid is efficient.

We have evaluated the performance of our method on two benchmarks: a 1575-dimensional line-actuated soft robot arm and a 1415-dimensional underwater swimming robot. Our use of a hierarchical grid reduces the number of forward dynamics evaluations by one to two orders of magnitude and a plan can be computed within 2 hours on a desktop machine. We show that the error of our system identification method can be bounded and the algorithm converges to the exact solution of the dynamics constrained motion planning problem as the error bound tends to zero.

II. Related Work

In this section, we give a brief overview of prior work on high-DOF robot systems, motion planning and control with dynamics constraints, and system identification.

High-DOF Robot Systems are used in various applications such as soft robots [22]. A popular method for numerically modeling these soft robots is the finite-element method (FEM) [7], [23], [24]. Another example is a low-DOF articulated robot swimming in high-DOF fluid environments [5], where the boundary element method (BEM) [9] is used to model robot-fluid interactions. A third example is a robot arm manipulating a piece of cloth [3], [4], [25], where the state of the cloth is also discretized using FEM in [25]. Both FEM and BEM induce a forward dynamics function, the evaluation of which involves matrix factorization and inversion, where the matrix is of size $\boldsymbol{O}(N \times N)$. As a result, the complexity of each evaluation is $\boldsymbol{o}(N^{1.5})$ using FEM [8] and $\boldsymbol{O}(N^2\mathbf{log}(N))$ using BEM [9]. Prior work [26], [27] compromises accuracy for speed by using iterative linearization and fast matrix solvers. Instead, our method uses accurate FEM or BEM solvers but stores the solver results in a hierarchical grid for speedup.

Dynamics-Constrained Motion Planning algorithms can be optimization-based or sampling-based methods. Optimization-based methods find locally optimal motion plans [14], [11], [28], [29], [30], [31] by iteratively minimizing an objective function under the dynamics constraints, where each iteration involves evaluating the forward dynamics function and its differentials. Sampling-based methods [10], [32] seek globally optimal motion plans, where the feasibility of each sampled motion plan is checked by calling the forward dynamics function. Differential dynamic programming [33] relies on forward dynamics evaluations to provide state and control differentials. Finally, reinforcement learning algorithms [12] requires a large number of forward dynamics evaluations to compute the policy gradient. Our method can be combined with all these methods.

System Identification has been widely used to approximate the forward dynamics function. Most system identification methods are data-driven and approximate the system dynamics using non-parametric models such as the Gaussian mixture model [34], Gaussian process [16], [35], neural networks [36], and nearest-neighbor computations [37]. Our method based on the hierarchical grid is also non-parametric. In most prior learning methods, training data are collected before using the identified system for motion planning. Recently, system identification has been combined with reinforcement learning [38], [39] for more efficient data-sampling of low-DOF dynamics systems. However, these methods do not guarantee the accuracy of the resulting approximation. In contrast, our method provides guaranteed accuracy.

III. Problem Formulation

In this section, we introduce the formulation of high-DOF robot systems and forward dynamics evaluations. Next, we formulate the problem of dynamics-constrained motion planning for high-DOF robots.

A. High-DOF Robot System Dynamics

A high-DOF robot can be formulated as a dynamics system, the configuration space of which is denoted as $\mathbb{C}$. Each $\mathbf{x} \in \mathbb{C}$ uniquely determines the kinematic state of the robot and the high-DOF environment with which it is interacting. To compute the dynamics state of the robot, we need $\mathbf{x}$ and its time derivative $\dot{\mathbf{x}}$. Given the dynamics state of the robot, its behavior is governed by the forward dynamics function:

$$\mathbf{g}(\mathbf{x}_i, \dot{\mathbf{x}}_i, \mathbf{u}_i) = (\mathbf{x}_{i+1}, \dot{\mathbf{x}}_{i+1}),$$

where the subscript denotes the timestep index, $\mathbf{x}_i$ is the kinematic state at time instance $i\Delta t$, and Δt is the timestep size. Finally, we denote $\mathbf{u}_i \in \mathbb{C}_c$ as the control input to the dynamics system (e.g., the joint torques for an articulated robot). In this work, we assume that the robot system is highly underactuated so that $|\mathbf{u}| \ll |\mathbf{x}|$. This assumption holds because the number of actuators in a robot is kept small to reduce manufacturing cost. For example, [2] proposed a soft robot octopus where each limb is controlled by only two air pumps. The forward dynamics function $\mathbf{g}$ is a result of discretizing the Euler-Lagrangian equation governing the dynamics of the robot. In this work, we consider two robot systems: an elastically soft robot arm and an articulated robot swimming in water.

B. Elastically Soft Robot

According to [7], [26], [27], the elastically soft robot is governed by the following partial differential equation (PDE):

$$\mathbf{M}\frac{\partial^2 \mathbf{x}}{\partial \mathbf{t}^2} = \mathbf{p}(\mathbf{x}) + \mathbf{c}(\mathbf{x}, \mathbf{u}), \tag{1}$$

where $\mathbf{p}(\mathbf{x})$ corresponds to the internal and external forces, $\mathbf{M}$ is the mass matrix, and $\mathbf{c}(\mathbf{x}, \mathbf{u})$ is the control force. This system is discretized by representing the soft robot as a tetrahedral mesh with $\mathbf{x}$ representing the vertex positions, as illustrated in Figure 1. Then the governing PDE (Equation 1) is discretized using an implicit-Euler time integrator as follows:

$$\mathbf{M}\frac{\mathbf{x}_{i+1} - 2\mathbf{x}_i + \mathbf{x}_{i-1}}{\Delta t^2} = \mathbf{p}(\mathbf{x}_{i+1}) + \mathbf{c}(\mathbf{x}_{i+1}, \mathbf{u}_i). \tag{2}$$

This function $\mathbf{g}$ is costly to evaluate because solving for $\mathbf{x}_{i+1}$ involves factorizing a large sparse matrix resulting from FEM discretization.

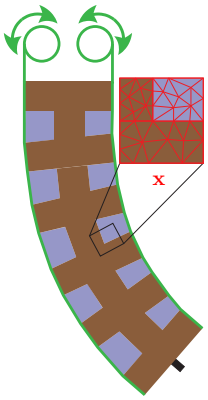

Fig. 1: A 2D soft robot arm modeled using two materials (a stiffer material shown in brown and a softer material shown in blue), making it easy to deform. It is discretized by a tetrahedral mesh with thousands of vertices (red). However, the robot is controlled by two lines (green) attached to the left and right edges of the robot, so that $|\mathbf{u}| = 2$. The control command is the pulling force on each line (green circles).

C. Underwater Swimming Robot System

Our second example, the articulated robot has a low-dimensional configuration space by itself. The configuration $\mathbf{x}$ consists of joint parameters. This robot is interacting with a fluid, so the combined fluid/robot configuration space is high-dimensional. According to [6], [5], the fluid's state can be simplified as a potential flow represented by the potential ϕ. This ϕ is discretized by sampling on each of the P vertices of the robot's surface mesh, as shown in Figure 2. The kinematic state of the coupled system is $(\mathbf{x}, \phi) \in \mathbb{C}$ and $N = |\mathbf{x}| + P$. However, ϕ can be computed from $\mathbf{x}$ and $\dot{\mathbf{x}}$ using the BEM method, denoted as $\phi(\mathbf{x}, \dot{\mathbf{x}})$. The governing dynamics equation in this case is:

$$\mathbf{M}(\mathbf{x})\frac{\partial^2 \mathbf{x}}{\partial \mathbf{t}^2} = \mathbf{C}(\mathbf{x}, \dot{\mathbf{x}}) + \mathbf{J}(\mathbf{x})\mathbf{u} + \left[\frac{d}{dt}\frac{\partial}{\partial \dot{\mathbf{x}}} - \frac{\partial}{\partial \mathbf{x}}\right] \int \frac{1}{2}\phi(\mathbf{x}, \dot{\mathbf{x}})\frac{\partial \phi(\mathbf{x}, \dot{\mathbf{x}})}{\partial \mathbf{n}}, \tag{3}$$

where $\mathbf{M}$ is the generalized mass matrix, $\mathbf{C}$ is the centrifugal and Coriolis force, and $\mathbf{J}(\mathbf{x})$ is the Jacobian matrix. Finally, the last term in Equation 3 is included to account for the fluid pressure forces, where the integral is over the surface of the robot and $\mathbf{n}$ is the outward surface normal. Time discretization of Equation 3 is performed using an explicit-Euler integrator, as follows:

$$\mathbf{M}(\mathbf{x}_i)\frac{\mathbf{x}_{i+1} - 2\mathbf{x}_i + \mathbf{x}_{i-1}}{\Delta t^2} = \mathbf{C}(\mathbf{x}_i, \dot{\mathbf{x}}_i) + \mathbf{J}(\mathbf{x}_i)\mathbf{u}_i + \left[\frac{d}{dt}\frac{\partial}{\partial \dot{\mathbf{x}}_i} - \frac{\partial}{\partial \mathbf{x}_i}\right] \int \frac{1}{2}\phi(\mathbf{x}_i, \dot{\mathbf{x}}_i)\frac{\partial \phi(\mathbf{x}_i, \dot{\mathbf{x}}_i)}{\partial \mathbf{n}}. \tag{4}$$

This function $\mathbf{g}$ is costly to evaluate because computing $\phi(\mathbf{x}_i, \dot{\mathbf{x}}_i)$ involves inverting the large, dense matrix that results from the BEM discretization.

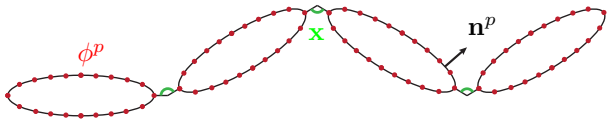

Fig. 2: An articulated swimming robot consists of 4 rigid ellipses connected by hinge joints. The configuration space of the robot is low-dimensional, consisting of joint parameters (green). The fluid state is high-dimensional and represented by a potential function ϕ discretized on the vertices of the robot's surface mesh (the pth component of ϕ^p in red). The kinetic energy is computed as a surface integral (the pth surface normal $\mathbf{n}^p$ in the black arrow).

D. Dynamics-Constrained Motion Planning and Control

We mainly focus on the specific problem of dynamics-constrained motion planning and feedback control. In the case of motion planning, we are given a reward function $\mathcal{R}(\mathbf{x}_i, \mathbf{u}_i)$ and our goal is to find a series of control commands $\mathbf{u}_1, \cdots, \mathbf{u}_{K-1}$ that maximizes the cumulative reward over a trajectory: $\mathbf{x}_1, \cdots, \mathbf{x}_K$, where K is the planning horizon. This maximization is performed under dynamics constraints, i.e. $\mathbf{g}$ must hold for every timestep:

$$\underset{\mathbf{u}_1, \cdots, \mathbf{u}_K \ 1}{\mathbf{argmax}} \sum_{i=1}^{K} \mathcal{R}(\mathbf{x}_i, \mathbf{u}_i) \quad \mathbf{s.t.}\ \mathbf{g}(\mathbf{x}_i, \dot{\mathbf{x}}_i, \mathbf{u}_i) = (\mathbf{x}_{i+1}, \dot{\mathbf{x}}_{i+1}). \tag{5}$$

In the case of feedback control, our goal is still to compute the control commands, but the commands are generated by a feedback controller $\pi(\mathbf{x}_i, \mathbf{w}) = \mathbf{u}_i$, where $\mathbf{w}$ is the optimizable parameters of π:

$$\underset{\mathbf{w}}{\mathbf{argmax}} \sum_{i=1}^{K} \mathcal{R}(\mathbf{x}_i, \mathbf{u}_i)\ \mathbf{s.t.}\ \mathbf{g}(\mathbf{x}_i, \dot{\mathbf{x}}_i, \pi(\mathbf{x}_i, \mathbf{w})) = (\mathbf{x}_{i+1}, \dot{\mathbf{x}}_{i+1}). \tag{6}$$

In both formulations, $\mathbf{g}$ must be evaluated tens of thousands of times to find the motion plan or controller parameters. In the next section, we propose a method to accelerate the evaluation of $\mathbf{g}$.

IV. Hierarchical System Identification

Our method is based on the observation that high-DOF robot systems are highly underactuated. As a result, we can identify a novel function $\mathbf{f}$ that maps from the low-dimensional control input $\mathbf{u}$ to the high-dimensional kinematic state $\mathbf{x}$. When the evaluation of $\mathbf{f}$ is involved in the evaluation of $\mathbf{g}$, it causes a bottleneck. We approximate $\mathbf{f}$, instead of $\mathbf{g}$, using our hierarchical system identification method. We first show how to identify this function for different robot systems and then describe our approach to constructing the hierarchical grid.

A. Function $\mathbf{f}_s$ for an Elastically Soft Robot

We identify function $\mathbf{f}_s$ for an elastically soft robot (subscript s for short). We first consider a quasistatic procedure in which all the dynamics behaviors are discarded and only the kinematic behaviors are considered. In this case, Equation 2 becomes:

$$0 = \mathbf{p}(\mathbf{x}_{i+1}) + \mathbf{c}(\mathbf{x}_{i+1}, \mathbf{u}_i). \tag{7}$$

Equation 7 defines our function $\mathbf{f}_s(\mathbf{u}_i) \triangleq \mathbf{x}_{i+1}$ implicitly, whose domain has dimension $|\mathbf{u}|$ and range has dimension N. We can also compute $\mathbf{f}_s$ explicitly using Newton's method as shown in [27]. This computation is costly due to the inversion of a large, sparse matrix $\partial \mathbf{p}(\mathbf{x}_{i+1})/\partial \mathbf{x}_{i+1}$.

Given $\mathbf{f}_s$ that only models kinematics, we can also compute the dynamics function. To do this, we reinterpret $\mathbf{f}_s$ as a shape embedding function such that for each $\mathbf{x}$ there exists a latent parameter $\boldsymbol{\alpha}$ and $\mathbf{f}_s(\boldsymbol{\alpha}) = \mathbf{x}$. Note that although we used the forward kinematic function to define $\mathbf{f}_s$, $\mathbf{f}_s$ does not have a physical meaning when used as a shape embedding function and the input $\boldsymbol{\alpha}$ is a dimensionless latent variable. This relationship can be plugged into Equation 1 to derive a projected dynamics system in the space of $\boldsymbol{\alpha}$ as:

$$\frac{\partial \mathbf{f}_s(\boldsymbol{\alpha}_{i+1})}{\partial \boldsymbol{\alpha}_{i+1}}^T \mathbf{M} \frac{\mathbf{f}_s(\boldsymbol{\alpha}_{i+1}) - 2\mathbf{f}_s(\boldsymbol{\alpha}_i) + \mathbf{f}_s(\boldsymbol{\alpha}_{i-1})}{\Delta t^2} = \frac{\partial \mathbf{f}_s(\boldsymbol{\alpha}_{i+1})}{\partial \boldsymbol{\alpha}_{i+1}}^T \left[\mathbf{p}(\mathbf{f}_s(\boldsymbol{\alpha}_{i+1})) + \mathbf{c}(\mathbf{f}_s(\boldsymbol{\alpha}_{i+1}), \mathbf{u}_i)\right], \tag{8}$$

where the left multiplication by $\partial \mathbf{f}_s(\boldsymbol{\alpha}_{i+1})/\partial \boldsymbol{\alpha}_{i+1}{}^T$ is due to Galerkin projection (see [40] for more details). This technique is similar to reduced order method [41] but we use a special shape space defined by the forward kinematic function. Using Equation 8, we can compute $\boldsymbol{\alpha}_{i+1}$ from $\boldsymbol{\alpha}_i, \boldsymbol{\alpha}_{i-1}$ via Newton's method and then recover $\mathbf{x}_{i+1}$ using $\mathbf{x}_{i+1} = \mathbf{f}_s(\boldsymbol{\alpha}_{i+1})$. Computing $\boldsymbol{\alpha}_{i+1}$ is very efficient because Equation 7 represents a low-dimensional dynamics system. In summary, the computational bottleneck of $\mathbf{g}$ lies in the computation of $\mathbf{f}_s$, which is a mapping from the low-dimensional variables $\boldsymbol{\alpha}$ to the high-dimensional variable $\mathbf{x}$.

B. Function $\mathbf{f}_u$ for an Underwater Swimming Robot

We present our $\mathbf{f}_u$ for the underwater swimming robot in this section (subscript u for short). The kinematic state $\mathbf{x}$ is low-dimensional and the fluid potential $\phi(\mathbf{x}, \dot{\mathbf{x}})$ is high dimensional. We interpret this case as an underactuation because the state of the high-dimensional fluid changes due to the low-dimensional state of the articulated robot. The fluid potential is computed by the boundary condition that fluids and an articulated robot should have the same normal velocities at every boundary point:

$$\left[\frac{\partial}{\partial \mathbf{n}^p}\right] \phi = \mathbf{n}^{pT} \mathbf{J}(\mathbf{x}) \dot{\mathbf{x}}, \tag{9}$$

where $\left[\frac{\partial}{\partial \mathbf{n}_i}\right]$ is a linear operator that is used to compute ϕ's directional derivative along the normal direction $\mathbf{n}^p$ at the pth surface sample (see Figure 2), which corresponds to the fluid's normal velocity. The right-hand side corresponds to the robot's normal velocity. Finally, we compute ϕ as:

$$\phi = \left[\frac{\partial}{\partial \mathbf{n}}\right]^{-1} \mathbf{n}^T \mathbf{J}(\mathbf{x}) \dot{\mathbf{x}},$$

where we assemble all the equations on all the P surface samples from Equation 9. Since there are a lot of surface sample points, $\left[\frac{\partial}{\partial \mathbf{n}}\right]$ is a large, dense $P \times P$ matrix and inverting it can be computationally cost. Therefore, we define:

$$\mathbf{f}_u(\mathbf{x}) \triangleq \left[\frac{\partial}{\partial \mathbf{n}}\right]^{-1} \mathbf{n}^T \mathbf{J}(\mathbf{x}), \tag{10}$$

which encodes the computationally costly part of the forward dynamics function $\mathbf{g}$. $\mathbf{f}_u$ has a domain of dimension $|\mathbf{x}|$ and a range of dimension $P \times |\mathbf{x}|$. Finally, $\mathbf{f}_u$ is a kinematics function like $\mathbf{f}_s$ because $\dot{\mathbf{x}}$ is excluded from $\mathbf{f}_u$. This choice reduces the dimension of the domain of $\mathbf{f}_u$.

C. Constructing the Hierarchical Grid

The evaluation of the forward dynamics function $\mathbf{g}$ requires the time-consuming evaluation of function $\mathbf{f}$ ($\mathbf{f}_s$ or $\mathbf{f}_u$). Moreover, certain motion planning algorithms require $\partial \mathbf{f}/\partial \mathbf{x}$ to solve Equation 5 or Equation 6. In this section, we develop an approach to approximate function $\mathbf{f}$ efficiently.

We accelerate $\mathbf{f}$ using a hierarchical grid-based structure, as shown in Figure 3 (a). Since the domain of $\mathbf{f}$ is low-dimensional, this formulation does not suffer from a-curse-of-dimensionality. To evaluate $\mathbf{f}(\mathbf{x})$ using a $|\mathbf{x}|$-dimensional grid with a grid size of Δx. We first identify the grid cell that contains $\mathbf{x}$. This grid cell has an interior:

$$\{\mathbf{y} | \forall i, \lfloor \mathbf{x}_i/\Delta x \rfloor = \lfloor \mathbf{y}_i/\Delta x \rfloor, \lceil \mathbf{x}_i/\Delta x \rceil = \lceil \mathbf{y}_i/\Delta x \rceil\}.$$

Each grid cell has $2^{|\mathbf{x}|}$ corner points $\mathbf{x}_c$ that satisfies $\lfloor \mathbf{x}_c/\Delta x \rfloor = \lceil \mathbf{x}_c/\Delta x \rceil$. For every corner point $\mathbf{x}_c$, we precompute $\mathbf{f}(\mathbf{x}_c)$ and $\partial \mathbf{f}/\partial \mathbf{x}_c$. Next, we can approximate $\mathbf{f}(\mathbf{x}), \partial \mathbf{f}/\partial \mathbf{x}$ at an arbitrary point inside the grid cell using a multivariate cubic spline interpolation [42]. Using a gird-based structure, we can improve the approximation accuracy by refining the grid and halving the grid size to $\Delta x/2$. After repeated refinements, a hierarchy of grids is constructed.

We first show how to build the grid at a fixed resolution. Evaluating $\mathbf{f}$ on every grid point is infeasible, but we do not know which grid points will be required before solving Equation 5. We therefore choose to build the grid on demand. When the motion planner requires the evaluation of $\mathbf{g}$ and $\partial \mathbf{g}/\partial \mathbf{x}, \dot{\mathbf{x}}$, the evaluation of $\mathbf{f}, \partial \mathbf{f}/\partial \mathbf{x}$ is also required. Next, we check each of the $2^{|\mathbf{x}|}$ corner points, $\mathbf{x}_c$. When $\mathbf{f}(\mathbf{x}_c)$ and $\partial \mathbf{f}/\partial \mathbf{x}_c$ have not been computed, we invoke the costly procedure of computing $\mathbf{f}$ exactly (Equation 7 and Equation 10) and then store the results in our database. After all the corner points have been evaluated, we perform multivariate spline interpolation.

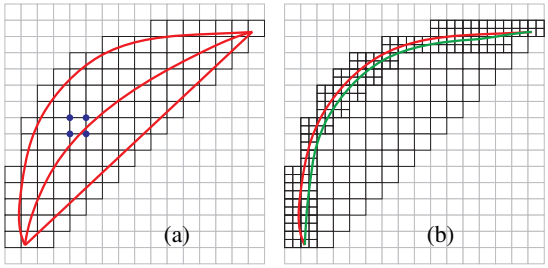

Fig. 3: (a): We check and precompute $\mathbf{f}$ on $2^2 = 4$ corner points (blue). The initial guess of a motion plan is the straight red line and the converged plan is the curved line. (b): During the next execution, we refine the grid using the last motion plan (red) as the initial guess. The next execution updates the red curve to the green curve. The two curves are close and the number of corner points on the fine grid is limited.

Our on-demand scheme only constructs the grid at a fixed resolution or grid size. Our method allows the user to define a threshold η and continually refines the grid for $R = \lceil \mathbf{log}(\Delta x/\eta) \rceil$ times until $\Delta x/2^R < \eta$. Therefore, for each evaluation of $\mathbf{f}$ and $\partial \mathbf{f}/\partial \mathbf{x}$, we need to compute the appropriate resolution. Almost all motion planning [14] and control [12] algorithms start from an initial motion plan or controller parameters and update iteratively until convergence. We want to use coarser grids when the algorithm is far from convergence and finer grids when it is close to converging. However, measuring the convergence of an algorithm is difficult and we do not have a unified solution for different motion planning algorithms. As a result, we choose to interleave motion planning or control algorithms with grid refinement. Specifically, we execute the motion planning or control algorithms R times. During the rth execution of the algorithm, we use the result of the $(r-1)$th execution as the initial guess and use a grid resolution of $\Delta x/2^r$, as shown in Algorithm 1. Note that the only difference between the rth execution and $(r-1)$th execution is that the accuracy of the approximation for $\mathbf{f}$ is improved. Therefore, the rth execution will only disturb the solution

slightly. This property will confine the solution space covered by the rth execution and limit the number of new evaluations on the fine grid, as shown in Figure 3 (b). Finally, we show that under mild assumptions, the solution for Equation 5 and Equation 6 found using an approximate $\mathbf{f}$ will converge to that of the original problem with the exact $\mathbf{f}$ as the number of refinements $\mathbf{R} \to \infty$:

Lemma IV.1. *Assuming the functions $\mathcal{R}, \mathbf{g}$ are sufficiently smooth, the solution space of $\mathbf{x}$ is bounded, and the forward kinematic function is non-singular, then there exists a small enough Δt such that solutions $\mathbf{u}$ of Algorithm 1 will converge to a local minimum of Equation 5 or Equation 6 as $R \to \infty$, as long as the local minimum is strict (the Hessian of $\mathcal{R}$ has full rank).*

The proof of Lemma IV.1 is straightforward and we provide it in our extended report downloadable from [43].

Algorithm 1 Motion planner with system identification

if Solve motion planning problem **then**
  Input: Initial guess $\mathbb{P}^0 \leftarrow \mathbf{u}_1, \ldots, \mathbf{u}_{K-1}$
else
  Input: Initial guess $\mathbb{P}^0 \leftarrow \mathbf{w}^0$
end if
Input: Threshold of accuracy, η
▷ Run multiple times of motion planning or control
for $r = 0, 1, \cdots, R = \lceil \mathbf{log}(\Delta x/\eta) \rceil$ **do**
  Set grid resolution to $\Delta x/2^r$ ▷ Refine the grid
  ▷ Use previous solution as initial guess
  if Solve motion planning problem **then**
    Solve Equation 5 from initial guess $\mathbb{P}^r$
    $\mathbb{P}^{r+1} \leftarrow \mathbf{u}_1^*, \ldots, \mathbf{u}_{K-1}^*$
  else
    Solve Equation 6 from initial guess $\mathbb{P}^r$
    $\mathbb{P}^{r+1} \leftarrow \mathbf{w}^*$
  end if
end for
Return $\mathbb{P}^R$

V. Implementation and Performance

We have evaluated our method on the 3D versions of the two robot systems described in Section III. The computational cost of each substep of our algorithm is summarized in Table I.

The 3D soft robot arm is controlled by four lines attached to four corners of the arm so that the control signal is 4-dimensional, $|\mathbf{u}| = 4$, and each evaluation of $\mathbf{f}_s$ requires $2^4 = 16$ grid corner point evaluations. To simulate its dynamics behavior, the soft arm is discretized using a tetrahedral mesh with 525 vertices so that $\mathbb{C}$ has $N = 3 \times 525 = 1575$ dimensions. To set up the hierarchical grid, we use an initial grid size of $\Delta x = 0.5$ and $\eta = 0.2$, so we will execute the planning algorithm for $R = 3$ times. In this example, we simulate a laser cutter attached to the top of the soft arm and the goal of our motion planning is to have the laser cut out a circle on the metal surface, as shown in Figure 5 (a). We use an optimization-based motion planner [14] that solves Equation 5. The computed motion plan is a trajectory discretized into $K = 200$ timesteps and the trajectory is initialized to zero control forces at every timestep. In this case, if we evaluate $\mathbf{f}_s(\mathbf{x})$ exactly each time, then 200 evaluations of $\mathbf{f}_s$ are needed in each iteration of the optimization. To measure the rate of acceleration achieved by our method, we plot the number of exact $\mathbf{f}_s$ evaluations on grid corner points against the number of iterations of trajectory optimization with and without hierarchical system identification in Figure 4 (a). Our method requires 22 times fewer evaluations and the total computational time is 20 times faster. The total number of evaluations of function $\mathbf{f}_s$ for the elastically soft arm is 216 with system identification and is 4800 without system identification. We can also add various reward functions to accomplish different planning tasks, such as obstacle avoidance shown in Figure 5 (b).

Example	N	$\lvert\mathbb{C}_c\rvert$	$\mathbf{f}$ (s)	$\mathbf{g}$ (s)	$\tilde{\mathbf{g}}$ (s)	+HSI (s)	-HSI (s)	Speedup	#Corner	Err
Deformation Arm Trajectory Optimization	1575	4	1.5	1.51	0.01	5.5	305	20	216	$7e-6$
Swimming Robot Trajectory Optimization	1415	3	0.9	0.902	0.02	3.1	183	190	732	$2e-5$
Swimming Robot Reinforcement Learning	1415	3	0.9	0.902	0.02	42	16424	1590	1973	$5e-5$

TABLE I: Summary of computational cost. From left to right: name of example, DOF of the robot system, dimension of $|\mathbb{C}_c|$, cost of evaluating $\mathbf{f}$, cost of evaluating $\mathbf{g}$, cost of evaluating $\mathbf{g}$ using system identification ($\tilde{\mathbf{g}}$), cost of each iteration of the planning algorithm with system identification, cost of each iteration without system identification (estimated), overall speedup, number of grid corner points evaluated, relative approximation error computed from: $\|\mathbf{g}(\mathbf{x}_i, \dot{\mathbf{x}}_i, \mathbf{u}_i) - \tilde{\mathbf{g}}(\mathbf{x}_i, \dot{\mathbf{x}}_i, \mathbf{u}_i)\| / \|\mathbf{g}(\mathbf{x}_i, \dot{\mathbf{x}}_i, \mathbf{u}_i)\|$.

For the 3D underwater robot swimmer, the robot has 3 hinge joints, so $\mathbf{x}$ is 3-dimensional and $2^3 = 8$ grid corner points are needed to evaluate $\mathbf{f}_u$. The fluid potential ϕ is discretized on the robot surface with 1412 vertices, so $\mathbb{C}$ of the robot system has $N = 3 + 1412 = 1415$ dimensions. To set up the hierarchical grid, we use an initial grid size of $\Delta x = 0.3$ and $\eta = 0.1$, so we will execute the planning algorithm for $R = 3$ times. Our goal is to have the robot move forward like a fish, as shown in Figure 5 (c). We use two algorithms to plan the motions for this robot. The first algorithm is an optimization-based planner [14], which solves Equation 5 from an initial motion plan with zero control forces. The resulting plot of the exact number of $\mathbf{f}_u$ evaluations on grid corner points is shown in Figure 4 (b). Our method requires 205 times fewer evaluations and the estimated total computational time is 190 times faster. We have also tested our method with reinforcement learning [44], which solves Equation 6 and optimizes a feedback swimming controller parameterized by a neural network. The neural network is fully connected with one hidden layer and SmoothReLU activation function, which is initialized using random weights. This algorithm is also iterative and, in each iteration, [44] calls the function $\mathbf{g}$ 16384 times. The resulting plot of the number of exact function $\mathbf{f}_u$ evaluations during reinforcement learning with and without hierarchical system identification is given in Figure 4 (c). Our method requires 1638 times fewer evaluations and the total computational time is 1590 times faster.

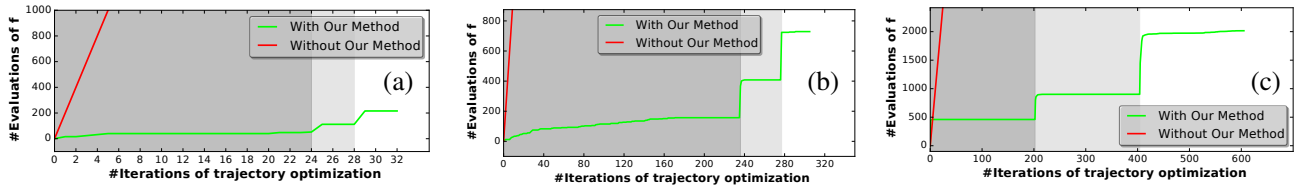

Fig. 4: Number of evaluations of $\mathbf{f}$ plotted against the number of planning iterations with (red) and without (green) our method. (a): Optimization-based motion planning for the deformation soft arm. (b): Optimization-based motion planning for the underwater robot swimmer. (c): Reinforcement learning for the underwater robot swimmer.

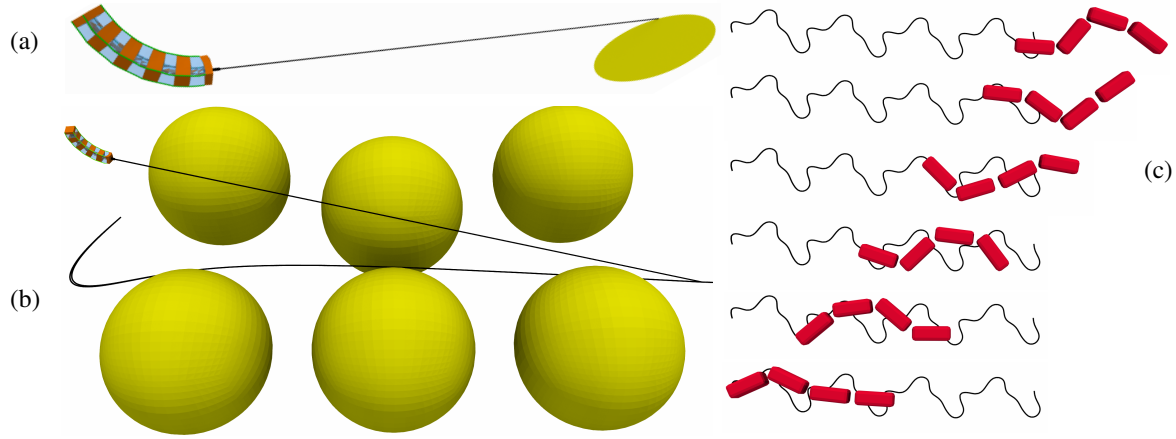

Fig. 5: (a): A frame of a 3D soft robot arm attached with a laser cutter carving out a circle (yellow) on a metal surface. The arm is controlled by four lines attached to the four corners (green). (b): 3D soft robot arm steering the laser beam to avoid obstacles (yellow). (c): Several frames of a 3D underwater swimming robot moving forward. The robot is controlled by the 3-dimensional joint torques. The black line is the locus of the center-of-mass.

A. *Comparisons*

Several prior works solve problems similar to those in our work. To control an elastically soft robot arm, [45] evaluates $\mathbf{g}$ and its differentials using finite difference in the space of control signals, $\mathbb{C}_c$. However, this method does not take dynamics into consideration and takes minutes to compute each motion plan in 2D workspaces. Other methods [46] only consider soft robots with a very coarse FEM discretization and do not scale to high-DOF cases. To control an underwater swimming robot, [47] achieves real-time performance in terms of evaluating the forward dynamics function, but they used a simplified fluid drag model; we use the more accurate potential flow model [5] for the fluid. Finally, the key difference between our method and previous system identification methods such as [34], [16], [35], [36], [37] is that we do not identify the entire forward dynamics function $\mathbf{g}$. Instead, we choose to identify a novel function $\mathbf{f}$ from $\mathbf{g}$ that encodes the computationally costly part of $\mathbf{g}$ and does not suffer from a-curse-of-dimensionality.

VI. Conclusion and Limitations

We present a hierarchical, grid-based data structure for performing system identification for high-DOF soft robots. Our key observation is that these robots are highly under-actuated. We identify a low-dimension to high-dimension mapping function $\mathbf{f}$ and store that function in our grid to accelerate the computation. Since the domain is low-dimensional, we can precompute $\mathbf{f}$ on a grid without suffering from a curse-of-dimensionality. The construction is performed in an on-demand manner and the entire hierarchy construction is interleaved with the motion planning or control algorithms. These techniques effectively reduce the number of grid corner points to be evaluated and thus reduce the total running time by one to two orders of magnitude.

One major limitation of the current method is that the function $\mathbf{f}$ cannot always be identified and there is no general method known to identify such a function for all types of robot systems. However, in our two examples, the domain of $\mathbf{f}$ has a dimension equal to the number of controlled DOFs and the function $\mathbf{f}$ itself does not account for dynamics (although we finally model dynamics when $\mathbf{f}$ is built into $\mathbf{g}$). These two observations indicate that the forward kinematic function is a good candidate of $\mathbf{f}$. Moreover, our method is only effective when $\mathbb{C}_c$ is very low-dimensional. Another issue is that we cannot guarantee that function $\mathbf{f}$ is a one-to-one mapping. Indeed, a single control input can lead to multiple quasistatic poses for a soft robot arm. In addition, our method does not scale well to a many-actuator system, where the space of control inputs is also high-dimensional. For future research, a direction is to extend our grid-based structure to handle functions with special properties such as one-to-many function mappings and discontinuous functions. Finally, to further reduce the number of grid corner points to be evaluated, we are interested in using a spatially varying grid resolution in which higher grid resolutions are used in regions where function $\mathbf{f}$ changes rapidly.

Acknowledgement

This research is supported in part by ARO grant W911NF16-1-0085, QNRF grant NPRP-5-995-2-415, and Intel.

REFERENCES

[1] J. Fras, M. Macias, Y. Noh, and K. Althoefer, "Fluidical bending actuator designed for soft octopus robot tentacle," in *2018 IEEE International Conference on Soft Robotics (RoboSoft)*. IEEE, 2018, pp. 253–257.

[2] J. Fras, Y. Noh, M. Maciaś, H. Wurdemann, and K. Althoefer, "Bio-inspired octopus robot based on novel soft fluidic actuator." IEEE, 2018.

[3] Z. M. Erickson, H. M. Clever, G. Turk, C. K. Liu, and C. C. Kemp, "Deep haptic model predictive control for robot-assisted dressing," *CoRR*, vol. abs/1709.09735, 2017.

[4] A. Clegg, W. Yu, Z. M. Erickson, C. K. Liu, and G. Turk, "Learning to navigate cloth using haptics," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2799–2805, 2017.

[5] E. Kanso, J. E. Marsden, C. W. Rowley, and J. B. Melli-Huber, "Locomotion of articulated bodies in a perfect fluid," *Journal of Nonlinear Science*, vol. 15, no. 4, pp. 255–289, Aug 2005.

[6] A. Munnier and B. Pinçon, "Locomotion of articulated bodies in an ideal fluid: 2d model with buoyancy, circulation and collisions," *Mathematical Models and Methods in Applied Sciences*, vol. 20, no. 10, pp. 1899–1940, 2010.

[7] Y.-c. Fung, P. Tong, and X. Chen, *Classical and computational solid mechanics*. World Scientific Publishing Company, 2017, vol. 2.

[8] A. George and E. Ng, "On the complexity of sparse qr and lu factorization of finite-element matrices," *SIAM Journal on Scientific and Statistical Computing*, vol. 9, no. 5, pp. 849–861, 1988.

[9] P. A. P. and N. J. A. L., "A spectral multipole method for efficient solution of large-scale boundary element models in elastostatics," *International Journal for Numerical Methods in Engineering*, vol. 38, no. 23, pp. 4009–4034.

[10] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[11] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of guidance, control, and dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[12] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, May 1992.

[13] J. Pan and D. Manocha, "Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing," *The International Journal of Robotics Research*, vol. 35, no. 12, pp. 1477–1496, 2016.

[14] 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," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[15] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[16] C. Williams, S. Klanke, S. Vijayakumar, and K. M. Chai, "Multi-task gaussian process learning of robot inverse dynamics," in *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. Curran Associates, Inc., 2009, pp. 265–272.

[17] S. Genc, "Parametric system identification using deep convolutional neural networks," in *2017 International Joint Conference on Neural Networks (IJCNN)*, May 2017, pp. 2112–2119.

[18] K. strm and P. Eykhoff, "System identificationa survey," *Automatica*, vol. 7, no. 2, pp. 123 – 162, 1971.

[19] S. Ross and J. A. Bagnell, "Agnostic system identification for model-based reinforcement learning," in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ser. ICML'12. USA: Omnipress, 2012, pp. 1905–1912.

[20] M. Skouras, B. Thomaszewski, S. Coros, B. Bickel, and M. Gross, "Computational design of actuated deformable characters," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 82:1–82:10, July 2013.

[21] X. Xiao, E. Cappo, W. Zhen, J. Dai, K. Sun, C. Gong, M. J. Travers, and H. Choset, "Locomotive reduction for snake robots," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 3735–3740.

[22] D. Rus and M. T. & Tolley, "Design, fabrication and control of soft robots." *Nature*, vol. 521, pp. 467–475, 2015.

[23] N. Cheney, R. MacCurdy, J. Clune, and H. Lipson, "Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding," in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '13. New York, NY, USA: ACM, 2013, pp. 167–174.

[24] Z. Pan and D. Manocha, "Realtime planning for high-dof deformable bodies using two-stage learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 1–8.

[25] B. Jia, Z. Hu, J. Pan, and D. Manocha, "Manipulating highly deformable materials using a visual feedback dictionary," in *ICRA*, 2018.

[26] C. Duriez, "Control of elastic soft robots based on real-time finite element method," in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 3982–3987.

[27] F. Largilliere, V. Verona, E. Coevoet, M. Sanz-Lopez, J. Dequidt, and C. Duriez, "Real-time Control of Soft-Robots using Asynchronous Finite Element Modeling," in *ICRA 2015*, SEATTLE, United States, May 2015, p. 6.

[28] C. Park, J. Pan, and D. Manocha, "Itomp: Incremental trajectory optimization for real-time replanning in dynamic environments," in *Proceedings of the Twenty-Second International Conference on International Conference on Automated Planning and Scheduling*, ser. ICAPS'12. AAAI Press, 2013, pp. 207–215.

[29] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 4569–4574.

[30] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *2009 IEEE International Conference on Robotics and Automation*, May 2009, pp. 489–494.

[31] C. Park, J. Pan, and D. Manocha, "Real-time optimization-based planning in dynamic environments using gpus," in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 4090–4097.

[32] D. J. Webb and J. van den Berg, "Kinodynamic rrt*: Optimal motion planning for systems with linear differential constraints," *CoRR*, vol. abs/1205.5088, 2012.

[33] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2012, pp. 4906–4913.

[34] G. Biagetti, P. Crippa, A. Curzi, and C. Turchetti, "Unsupervised identification of nonstationary dynamical systems using a gaussian mixture model based on em clustering of soms," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, May 2010, pp. 3509–3512.

[35] D. Nguyen-Tuong, M. Seeger, and J. Peters, "Model learning with local gaussian process regression," vol. 23, pp. 2015–2034, 10 2009.

[36] S. R. Chu, R. Shoureshi, and M. Tenorio, "Neural networks for system identification," *IEEE Control Systems Magazine*, vol. 10, no. 3, pp. 31–35, April 1990.

[37] W. Greblicki and M. Pawlak, "Hammerstein system identification with the nearest neighbor algorithm," *IEEE Transactions on Information Theory*, vol. 63, no. 8, pp. 4746–4757, Aug 2017.

[38] W. Yu, J. Tan, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," *arXiv preprint arXiv:1702.02453*, 2017.

[39] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Advances in Neural Information Processing Systems*, 2014, pp. 1071–1079.

[40] K. Carlberg, C. Bou-Mosleh, and C. Farhat, "Efficient non-linear model reduction via a least-squares petrovgalerkin projection and compressive tensor approximations," *International Journal for Numerical Methods in Engineering*, vol. 86, no. 2, pp. 155–181.

[41] J. Chenevier, D. Gonzlez, J. V. Aguado, F. Chinesta, and E. Cueto, "Reduced-order modeling of soft robots," *PLOS ONE*, vol. 13, no. 2, pp. 1–15, 02 2018.

[42] C. C. Lalescu, "Two hierarchies of spline interpolations. practical algorithms for multivariate higher order splines," *arXiv preprint arXiv:0905.3564*, 2009.

[43] B. Jia, Z. Pan, and D. Manocha, "Fast motion planning for high-dof robot systems using hierarchical system identification," *arXiv preprint arXiv:1809.08259*, 2018.

[44] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[45] G. Fang, C.-D. Matte, T.-H. Kwok, and C. C. Wang, "Geometry-based direct simulation for multi-material soft robots," in *ICRA*, 2018.
[46] R. Gayle, P. Segars, M. C. Lin, and D. Manocha, "Path planning for deformable robots in complex environments," in *In Robotics: Systems and Science*, 2005.
[47] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5026–5033.