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 or an articulated robot interacting with a deformable environment. Our approach considers dynamics constraints and we present a novel technique to accelerate the forward dynamics computation using a data-driven method. We precompute the forward dynamics function of the robot system on a hierarchical adaptive grid. Furthermore, we exploit the properties of underactuated robots systems and perform these computations in a lower dimensional space. We provide error bounds for approximate forward dynamics computation and use our approach for optimization-based motion planning and reinforcement-learning-based feedback control. We highlight the performance on two high-DOF robot systems: a high-DOF line-actuated elastic robot arm and an underwater swimming robot operating 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. Other scenarios correspond to articulated robots interacting with highly deformable objects like cloth [3], [4] or deformable environments like fluids [5], [6]. In these cases, the number of degrees-of-freedom (DOF \(N\)) can be much higher 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 \(O(N^{1.5})\) [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 points 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 \(O(N^2 \log(N))\) [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 required by the motion planning and control algorithms. For sampling-based planners, the number of samples can be reduced by learning a prior sampling distribution centered around 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], which can become a major bottleneck for high-DOF robot systems.

Another method for improving the sampling efficiency is system identification [16], [17]. These methods approximate 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 new efficient method for system identification of a high-DOF robot system. 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 \(f : C_c \rightarrow C\), where \(C_c\) is the space of the controlled DOFs. Since \(C_c\) is low-dimensional, sampling in \(C_c\) does not suffer from a curse-of-dimensionality. Therefore, our method accelerates the evaluations of \(f\) by precomputing and storing \(f\) 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 \(2^{|C_c|}\) 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 the hierarchical grid is efficient.

We have combined our dynamics evaluation algorithm with optimization-based motion planning and reinforcement-learning-based feedback control. We evaluate the performance of these algorithms on two benchmarks: a 1575-dimensional line-actuated elastic robot arm and a 1415-dimensional underwater swimming robot system. 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. This is due to the increasing use of soft robots [22]. A popular method for numerically modeling these soft robots is the finite-element method (FEM) [7]. FEM represents a soft robot using a general mesh with thousands of vertices or DOFs. The other set of applications includes a low-DOF articulated robot interacting with high-DOF passive objects, such as when a swimming robot interacts with fluids [5]. To numerically model the robot-fluid interaction, some methods represent the state of the fluid using the boundary element method (BEM) [9]. BEM represents the fluid state using a surface mesh that has hundreds of DOFs on a 2D manifold and tens of thousands of DOFs in 3D workspaces. Another example is a robot arm manipulating a piece of cloth [3], [4], [23], where the state of the cloth is also discretized using FEM in [23]; the cloth is also represented using a mesh with thousands of DOFs. Both FEM and BEM induce a forward dynamics function \( f \), the evaluation of which involves matrix factorization and inversion, where the matrix is of size \( O(N \times N) \). As a result, the complexity of evaluating \( f \) is \( o(N^{1.5}) \) using FEM [8] and \( O(N^2 \log(N)) \) using BEM [9].

Dynamics-Constrained Motion Planning algorithms can be optimization-based or sampling-based methods. Optimization-based methods are used to compute locally optimal motion plans [14], [11] by minimizing a set of state-dependent or control-dependent objective functions using the dynamics constraints. Such optimization is performed iteratively, where each iteration involves evaluating the forward dynamics function \( f \) and its differentials. On the other hand, sampling-based methods [10], [24] seek globally feasible or optimal motion plans. These methods repeatedly evaluate proposed motion plans by calling the forward dynamics function \( f \). Feedback control algorithms also include a large number of evaluations. Differential dynamic programming [25] relies on \( f \) evaluations to provide state and control differentials. These differentials are used to optimize a trajectory over a short horizon. Finally, reinforcement learning algorithms [12] optimize the feedback controller parameters by repeatedly computing the policy gradient, which requires a large number of evaluations of \( f \). Our method can be combined with all these methods.

System Identification has been widely used to approximate the forward dynamics function \( f \) when the evaluation of \( f \) and its differentials is costly. Most system identification methods are data-driven and approximate the system dynamics using non-parametric models such as the Gaussian mixture model [26], Gaussian process [16], [27], neural networks [28], and nearest-neighbor computation [29]. 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 [30], [31] 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 dynamics evaluation method can be easily combined with motion planning algorithms, it handles high-DOF systems, and it provides guaranteed accuracy.

III. PROBLEM FORMULATION

In this section, we introduce the formulation of high-DOF robot systems and 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 \( C \). Each \( x \in C \) 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 \( x \) and its time derivative \( \dot{x} \). Given the dynamics state of the robot, its behavior is governed by the forward dynamics function:

\[
g(x_i, \dot{x}_i, u_i) = (x_{i+1}, \dot{x}_{i+1}),
\]

where the subscript denotes the timestep index, \( x_i \) is the kinematic state at time instance \( i \Delta t \), and \( \Delta t \) is the timestep size. Finally, we denote \( u_i \in C_c \) 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 \( |u| \ll |x| \). 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 \( g \) 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], [32], [33], the elastically soft robot is governed by the following partial differential equation (PDE):

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

where $p(x)$ corresponds to the internal and external forces, $M$ is the mass matrix, and $c(x, u)$ is the control force. This system is discretized by representing the soft robot as a tetrahedra mesh with $x$ 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:

$$M \frac{x_{i+1} - 2x_i + x_{i-1}}{\Delta t^2} = p(x_{i+1}) + c(x_{i+1}, u_i).$$

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

C. Underwater Swimming Robot System

Our second example, the articulated robot swimmer, has a low-dimensional configuration space. The configuration $x$ 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 $\phi$. The kinematic state of the sampling on each of the vertices of the robot’s surface mesh (the $p$th component of $\phi$ in red). The kinetic energy is computed as a surface integral (the $p$th surface normal $n^p$ 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 $R(x, u_i)$ and our goal is to find a series of control commands $u_{i+1}, \cdots, u_{K-1}$ that maximizes the cumulative reward over a trajectory: $x_1, \cdots, x_K$, where $K$ is the planning horizon. This maximization is performed under dynamics, i.e. $g$ must hold for every timestep:

$$\arg\max_{u_{i+1}, \cdots, u_{K-1}} \sum_{i=1}^{K} R(x_i, u_i) \ 	ext{s.t.} \ g(x_i, \dot{x}_i, u_i) = (x_{i+1}, \dot{x}_{i+1}).$$

In the case of feedback control, our goal is still to compute the control commands, but the commands are generated by a feedback controller $\pi(x_i, w) = u_i$, where $w$ is the optimizer parameters of $\pi$:

$$\arg\max_{w} \sum_{i=1}^{K} R(x_i, u_i) \ 	ext{s.t.} \ g(x_i, \dot{x}_i, \pi(x_i, w)) = (x_{i+1}, \dot{x}_{i+1}).$$

In both formulations, $g$ 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 $g$.

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 $f$ that maps from the low-dimensional control input $u$ to the high-dimensional kinematic state $x$. When the evaluation of $f$ is involved in the evaluation of $g$, it causes a bottleneck. We approximate $f$, instead of $g$, 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 $f$ for an Elastically Soft Robot

We identify function $f$ for an elastically soft robot. 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:

$$0 = p(x_{i+1}) + c(x_{i+1}, u_i).$$

Equation 7 defines our function $f(u_i) \triangleq x_{i+1}$ implicitly. We can also compute $f$ explicitly using Newton’s method. This
is a mapping from the low-dimensional variables $\alpha$ to a high-dimensional dynamics system. In summary, the computation is costly due to the inversion of a large, sparse matrix $\partial p(x_{i+1})/\partial x_{i+1}$.

Given $f$ that defines the quasistatic function, we can also compute the dynamics function. We assume that function $f$ is a shape embedding function such that for each $x$ there exists a latent parameter $\alpha$ and $f(\alpha) = x$. Note that $\alpha$ is not the control input, but a latent space parameter without any physical meaning. This relationship can be plugged into Equation 1 to derive a projected dynamics system in the space of the control input as:

$$
\frac{\partial f(\alpha_{i+1})}{\partial \alpha_{i+1}}^T \begin{bmatrix} F(\alpha_{i+1}) - 2f(\alpha_{i}) + f(\alpha_{i-1}) \end{bmatrix} = f(\alpha_{i+1}) = \begin{bmatrix} \frac{\partial f(\alpha_{i+1})}{\partial \alpha_{i+1}}^T [p(f(\alpha_{i+1})) + c(f(\alpha_{i+1}), u_{i})] \end{bmatrix},
$$

where the left multiplication by $\partial f(\alpha_{i+1})/\partial \alpha_{i+1}^T$ is due to Galerkin projection (see [34] for more details). To time integrate Equation 8, we first compute $\alpha_{i+1}$ using $\alpha_{i}$ and then recover $x_{i+1} = f(\alpha_{i+1})$. Computing $\alpha_{i+1}$ is very efficient because Equation 7 represents a low-dimensional dynamics system. In summary, the computational bottleneck of $g$ lies in the computation of $f$, which is a mapping from the low-dimensional variables $u, \alpha$ to the high-dimensional variable $x$.

**B. Function $f$ for an Underwater Swimming Robot**

We present our $f$ for the underwater swimming robot in this section. The kinematic state $x$ is low-dimensional and the fluid potential $\phi(x, \dot{x})$ 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 point:

$$
\frac{\partial \phi}{\partial \vec{n}} = \vec{n}^T \dot{\mathbf{J}}(x) \dot{x},
$$

where $\frac{\partial \phi}{\partial \vec{n}}$ is a linear operator that is used to compute $\phi$’s directional derivative along the normal direction $\vec{n}$ at the $p$th 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 $\phi$ as:

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

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

$$
\mathbf{f}(x) \triangleq \frac{\partial \phi}{\partial \vec{n}}^{-1} \vec{n}^T \dot{\mathbf{J}}(x),
$$

which encodes the computationally costly part of the forward dynamics function $g$. Here we use a modified notation so that the range of $f$ is not $\mathbb{C}$ but $(x, f(x, \dot{x})) \in \mathbb{C}$. However, our method is still valid with this formulation. We only need $f$ to be a mapping from a low-dimensional space to a high-dimensional space.

**C. Constructing the Hierarchical Grid**

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

We accelerate $f$ using a hierarchical grid-based structure, as shown in Figure 3(a). Since the domain of $f$ is low-dimensional, this formulation does not suffer from a curse-of-dimensionality. To evaluate $f(x)$ using a grid with a grid size of $\Delta x$, we first identify the grid that contains $x$. This grid node has $2^{\lceil \log \Delta x \rceil}$ corner points, $x_c$, with coordinates:

$$
x_c = \left[ \frac{x}{\Delta x} \right] \Delta x, \left[ \frac{x}{\Delta x} \right] \Delta x.
$$

For every corner point $x_c$, we precompute $f(x_c)$ and $\partial f/\partial x_c$. Next, we can approximate $f(x)$, $\partial f/\partial x$ at an arbitrary point using a multivariate cubic spline interpolation [35]. One main point of using a grid-based structure is that 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.

**Algorithm 1** Motion planner using hierarchical system identification

1. if Solve motion planning problem then
2. Input: Initial guess $[p^0 \leftarrow u_1, \ldots, u_{K-1}]$
3. else
4. Input: Initial guess $[p^0 \leftarrow w^0]$
5. end if
6. Input: Threshold of accuracy, $\eta$
7. for $r = 0, 1, \ldots, R = \lceil \log(\Delta x/\eta) \rceil$ do
8. Refine the grid
9. Set grid resolution to $\Delta x/2^r$
10. Use previous solution as initial guess
11. if Solve motion planning problem then
12. Solve Equation 5 from initial guess $[p^r \leftarrow u_1, \ldots, u_{K-1}]$
13. else
14. Solve Equation 6 from initial guess $[p^r \leftarrow w^*]$
15. end if
16. $[p^{r+1} \leftarrow u_1, \ldots, u_{K-1}]$
17. $[p^{r+1} \leftarrow w^*]$
18. end if
19. end for
20. Return $[p^R]$

Our main step is the construction of the grid hierarchy. We first show how to build the grid at a fixed resolution. Evaluating $f$ on every grid point is infeasible, but we do not know which grid points will be required before solving Equation 5. Therefore, we choose to build the grid on demand. When the motion planner requires the evaluation of $g$ and $\partial g/\partial x$, the evaluation of $f$, $\partial f/\partial x$ is also required. Next, we check each of the $2^{\lceil \log \Delta x \rceil}$ corner points, $x_c$. When $f(x_c)$ and $\partial f/\partial x_c$ have not been computed, we invoke the costly procedure of computing $f$ exactly (Equation 7 and Equation 10) and 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 $f$ 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.](image)

Our on-demand scheme only constructs the grid at a fixed resolution or grid size. Our method allows the user to define a threshold $\eta$ and continually refines the grid for $R = \lceil \log(\Delta x/\eta) \rceil$ times until $\Delta x/2^R < \eta$. Therefore, for each evaluation of $f$ and $\partial f/\partial x$, 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 updates iteratively until convergence. We also 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 $r$th 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 $r$th execution and the $(r-1)$th execution is the accuracy of approximation for $f$ is improved. Therefore, the $r$th execution will only perturb the solution slightly. This property will confine the solution space covered by the $r$th 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 of Equation 5 and Equation 6 found using an approximate $f$ will converge to that of the original problem with the exact $f$ as the number of refinements $R \rightarrow \infty$:

**Lemma IV.1.** Assuming the functions $R$, $g$ are sufficiently smooth, the solution space of $x$ is bounded, and the forward kinematic function is non-singular, then there exists a small enough $\Delta t$ such that solutions $u$ of Algorithm 4 will converge to a local minimum of Equation 5 or Equation 6 as $R \rightarrow \infty$, as long as the local minimum is strict (the Hessian of $R$ has full rank).

The proof of Lemma IV.1 is straightforward and we provide it in our appendix for completeness.

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 1.

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, $|u| = 4$, and each evaluation of $f$ requires $2^4 = 16$ grid point evaluations. To simulate its dynamics behavior, the soft arm is discretized using a tetrahedra mesh with 525 vertices so that $C$ 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 4 (a). We use an optimization-based motion planner [14], which solves Equation 5. The computed motion plan is a trajectory discretized into $K = 200$ timesteps. In this case, if we evaluate $f(x)$ exactly each time, then 200 evaluations of $f$ are needed in each iteration of the optimization. To measure the rate of acceleration achieved by our method, we plot the number of exact $f$ evaluations on grid corner points against the number of iterations of trajectory optimization with and without hierarchical system identification in Figure 5 (a).

Our method requires 22 times fewer evaluations and the total computational time is 20 times faster. The total number of evaluations of function $f$ for the elastically soft arm is 216 with system identification and is 4800 without system identification. We can also added various reward functions to accomplish different planning tasks, such as obstacle avoidance shown in Figure 4 (b).

For the 3D underwater robot swimmer, the robot has 3 hinge joints, so $x$ is 3-dimensional and $2^3 = 8$ grid corner points are needed to evaluate $f$. The fluid potential $\phi$ is discretized on the robot surface with 1412 vertices, so $C$ 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 4 (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. The resulting plot of the number of exact $f$ evaluations on grid corner points is shown in Figure 5 (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 [36], which solves Equation 6 and optimizes a feedback swimming controller. This algorithm is also iterative and, in each iteration, [36] calls the function $g$ 16384 times. The resulting plot of the number of exact function $f$ evaluations during reinforcement learning with and without hierarchical system identification is given in Figure 5 (c). Our method requires 1638 times fewer evaluations and the total computational time is 1590 times faster.

A. Comparisons

Several prior works solve problems similar to those in our work. To control an elastically soft robot arm, [37] evaluates $g$ and its differentials using finite difference in the space
of control signals, $C_c$. However, this method does not take dynamics into consideration and takes minutes to compute each motion plan in 2D workspaces. Other methods [38] only consider soft robots with a very coarse FEM discretization and do not scale to high-DOF cases. To control an underwater swimming robot, [39] 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 [26], [16], [27], [28], [29] is that we do not identify the entire forward dynamics function $g$. Instead, we choose to identify a novel function $f$ from $g$ that encodes the computationally costly part of $g$ and does not suffer from a curse-of-dimensionality.

| Example               | $N$ | $C_c$ | Cost $f$ | Cost $g$ | HSI $f$ | HSI $g$ | Speedup | $\text{Err}$ |
|-----------------------|-----|-------|----------|----------|---------|---------|---------|-------------|
| Deformation Arm       | 575 | 4     | 1.5      | 1.51     | 0.01    | 5.5     | 305     | 20          | 216         | $7e - 6$   |
| Trajectory Optimization | 415 | 3     | 0.9      | 0.902    | 0.02    | 3.1     | 183     | 190         | 732         | $2e - 5$   |
| Swimming Robot         | 415 | 3     | 0.9      | 0.902    | 0.02    | 42      | 16424   | 1590        | 1973        | $5e - 5$   |
| Trajectory Learning    | 415 | 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 $|C_c|$, cost of evaluating $f$, cost of evaluating $g$, cost of evaluating $g$ using system identification ($\tilde{g}$), 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: $\|g(x_i, \dot{x}_i, u_i) - \tilde{g}(x_i, \dot{x}_i, u_i)\|/\|g(x_i, \dot{x}_i, u_i)\|$.

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 underactuated. We compute a low dimensional approximation of the dynamics function and use that to accelerate the computation. As a result, we can precompute $f$ 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 $f$ cannot always be identified and there is no general method known to identify such a function for all types of robot systems. Moreover, our method is only effective when $C_c$ is very low-dimensional. Another major issue is that we cannot guarantee that function $f$ is a one-to-one mapping. Indeed, a single control input can lead to multiple quasistatic poses for a soft robot arm. Therefore, a major direction of future research 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 $f$ changes rapidly.
We prove Lemma IV.1 for the elastically deformable soft arm, the forward dynamics function of which is Equation 8. The case with the underwater swimming robot is similar. Before our derivation, we note that Equation 8 involves the latent variable \( \alpha \), which complicates our derivation. We first transform the variables by plugging \( x = f(\alpha) \) into Equation 8. In this way, we eliminate \( \alpha \), and Equation 8 becomes:

\[
\argmax_{u_1, \cdots, u_K} \sum_{i=1}^{K} \tilde{R}(x_i, u_i)
\]

s.t. \( g(x_i, \alpha_i, u_i) = (\alpha_{i+1} - \alpha_{i+1}) \).

Next, we replace \( \alpha \) with \( x \) for notational consistency, giving:

\[
\argmax_{u_1, \cdots, u_K} \sum_{i=1}^{K} \tilde{R}(x_i, u_i)
\]

s.t. \( G_i = 0 \quad \forall 1 \leq i < K, \)

where we have:

\[
G_i \triangleq g(x_i, x_{i+1}, u_i, u_i) - (x_{i+1} - x_i + x_{i+1}) \Delta t,
\]

assuming finite difference approximation as is used in Equation 8. And we define \( \tilde{R}(x_i, u_i) \triangleq R(f(x_i), u_i) \). Our proof is based on Equation 11 and we can perform the same transformation for Equation 8. This transformation does not change the smoothness properties of various functions. Also note that, since we use \( f \) as a shape embedding function in case of elastically deformable soft arm, \( f \) is the forward kinematic function which is required to be non-singular. In the first part of the proof, we show that Equation 11 satisfies LICQ [40], so every local minimum satisfies the KKT condition.

Lemma VI.1. Assuming the functions \( \tilde{R}, G \) are sufficiently smooth, the solution space of \( x \) is bounded, and the forward kinematic function is non-singular, there then exists a small enough \( \Delta t \) such that LICQ holds.

Proof. LICQ requires the constraint Jacobian \( J \) to have full rank. Our constraint Jacobian \( J \) takes the following form:

\[
J = \begin{bmatrix}
\frac{\partial G_1}{\partial x_1} & \frac{\partial G_1}{\partial x_2} & \cdots & \frac{\partial G_1}{\partial x_K}
\end{bmatrix},
\]

where \( \frac{\partial G}{\partial x} \) is a square, block-lower-triangular matrix:

\[
\frac{\partial G}{\partial x} = \begin{pmatrix}
\frac{\partial G_1}{\partial x_1} & 0 & \cdots & 0 \\
\frac{\partial G_1}{\partial x_2} & \frac{\partial G_1}{\partial x_3} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & \frac{\partial G_1}{\partial x_K}
\end{pmatrix},
\]

where \( G_1 \) is the implicit form of \( g \). For the elastically deformable soft arm, this is Equation 8:

\[
\frac{\partial (f(x_{i+1}) + M(x_{i+1} - f(x_i) + f(x_{i+1})) - p(f(x_{i+1}) + e(f(x_{i+1}), u_{i+1}))}{\partial x_{i+1}}.
\]

As long as \( \frac{\partial G}{\partial x_{i+1}} \) has full rank, \( J \) has full rank and LICQ is satisfied. We have the following form of \( \frac{\partial G}{\partial x_{i+1}} \):

\[
\frac{\partial G_1}{\partial x_{i+1}} = A + B = \frac{\partial f(x_{i+1})}{\partial x_{i+1}} T [M(f(x_{i+1}) - f(x_i) + f(x_{i+1})) - p(f(x_{i+1}) + e(f(x_{i+1}), u_{i+1}))]
\]

\[
B = \frac{\partial f(x_{i+1})}{\partial x_{i+1}} T \frac{\partial f(x_{i+1})}{\partial x_{i+1}} T C \triangleq \frac{\partial f(x_{i+1})}{\partial x_{i+1}} T M \frac{\partial f(x_{i+1})}{\partial x_{i+1}} T \]

When \( \Delta t \) is sufficiently small, \( A \) is upper bounded because the \( \Delta t \)-dependent term, \( f(x_{i+1}) - f(x_{i+1} - 2f(x_i) + f(x_{i+1})) \), is bounded because \( x_i \) is bounded and \( f \) is smooth. \( B \) is bounded because it is independent of \( \Delta t \). Finally, we can choose \( \Delta t \) small enough so that \( \text{rank}(\frac{\partial G_1}{\partial x_{i+1}}) = \text{rank}(C) \). We also assume the forward kinematic function (function \( f \) in the case of elastically soft arm) is non-singular so that \( C \) has full rank. As a result, \( \frac{\partial G_1}{\partial x_{i+1}} \) has full rank for all \( i \) and LICQ holds.

Note that Lemma VI.1 holds for both exact function \( f \) and approximate \( f \) by hierarchical system identification. Our approximate \( f \) is derived using spline interpolation, which is sufficiently smooth. Given Lemma VI.1, the convergence of Algorithm 1 (Lemma IV.1) is obvious and the proof is an extension to Theorem 1.21 of [41] as follows:

Proof. When we let iteration number \( r \to \infty \) in Algorithm 1, we will solve for a sequence of motion plans \( [T_{1}, 2, \ldots, t] \), where \( P_r \) consists of \( [x_1, \cdots, x_K] \) and satisfies the KKT condition due to Lemma VI.1. Therefore we have, with a slight abuse of notations:

\[
\frac{\partial R}{\partial P} + \frac{\partial G(\Delta x, P)}{\partial P} \lambda = 0 \quad G(\Delta x, P) = 0, \tag{12}
\]

where \( \lambda \) is the Lagrange multipliers. Since \( x \) is bounded, \( P_r \) is bounded and the sequence \( P_r \) will have an accumulation point \( P_\infty \) in the compact domain. The remaining issue is to show that \( P_\infty \) satisfies the KKT condition and that \( P_\infty \) is the unique accumulation point.

\( P_\infty \) satisfies the KKT condition: Note that, when using a grid to approximate \( f \) with grid size \( \Delta x \), we are essentially defining a new KKT system by modifying \( G \):

\[
\frac{\partial R}{\partial P} + \frac{\partial G(\Delta x, P)}{\partial P} \lambda = 0 \quad G(\Delta x, P) = 0, \tag{13}
\]

where the reward \( R \) is not approximated by our grid so it is not a function of \( \Delta x \). By changing \( \Delta x \), we get a one-parameter set of sufficiently smooth KKT problems. By passing Equation 11 onto infinity, we have \( P_\infty \) satisfying the KKT condition of Equation 11:

\( P_\infty \) is unique: \( P_\infty \) is the strict local solution at \( \Delta x = 0 \) and there must be a neighborhood \( B(P_\infty, \delta_1) \) in which \( P_\infty \) is the global solution. By the Weierstrass theorem, there is a neighborhood \( |\Delta x| < \delta_2 \) such that Equation 11 has a global solution when \( P \in B(P_\infty, \delta_1) \) and \( |\Delta x| < \delta_2 \).

Now that \( P_\infty \) is an accumulation point, there must be a large enough iteration number \( r_1 \) in Algorithm 1 such that \( P_{r_1} \in B(P_\infty, \delta_1) \) and \( |\Delta x| < \delta_2 \). From this iteration onwards, every \( P_{r>r_1} \in B(P_\infty, \delta_1) \) is because Algorithm 1 uses \( P_r \) as the initial guess and \( P_r \) is in \( B(P_\infty, \delta_1) \). Therefore, \( P_r \) is the local minimum in the same basin area of \( P_r \), which is the global minimum in \( B(P_\infty, \delta_1) \). As a result, \( P_{r>r_1} \in B(P_\infty, \delta_1) \) by mathematical induction.

Finally, if the sequence \( P_{1, 2, \ldots} \) converges to an accumulation point \( P_\infty \), then \( P_\infty \) is the solution to the KKT system at \( \Delta x = 0 \) and \( P_\infty \in B(P_\infty, \delta_1) \). But there is only one global minimum for this problem in \( B(P_\infty, \delta_1) \), so that \( P_\infty = P_\infty \).
