Realtime Planning for High-DOF Deformable Bodies using Two-Stage Learning

Zherong Pan[1] and Dinesh Manocha[1]
http://gamma.cs.unc.edu/DefoControl

***Abstract*— We present a method for planning the motion of arbitrarily-shaped volumetric deformable bodies or robots through complex environments. Such robots have very high-dimensional configuration spaces and we compute trajectories that satisfy the dynamics constraints using a two-stage learning method. First, we train a multitask controller parameterized using dynamic movement primitives (DMP), which encodes various locomotion or movement skills. Next, we train a neural-network controller to select the DMP task to navigate the robot through environments while avoiding obstacles. By combining the finite element method (FEM), model reduction, and contact invariant optimization (CIO), the DMP controller's parameters can be optimized efficiently using a gradient-based method, while the neural-network's parameters are optimized using Deep Q-Learning (DQL). This two-stage learning algorithm also allows us to reuse the trained DMP controller for different navigation tasks, such as moving through different environmental types and to different goal positions. Our results show that the learned motion planner can navigate swimming and walking deformable robots with thousands of DOFs at realtime.**

I. Introduction

Motion planning for deformable bodies is an important problem in robotics, factory automation, physically-based modeling, and surgical simulation. This problem frequently arises in robotic design and control, haptic rendering, medical robotics, cable placement and layouts, and assembly planning. Many of these applications tend to use complex deformable models and can simulate their movements subject to external and internal forces.

There is extensive work on motion planning of rigid, articulated, and deformable models. Some widely used methods are based on sampling-based planning or optimization-based planning. Compared with articulated or rigid bodies, motion planning for deformable robots can be computationally more challenging due to the very high-dimensional configuration space of the robot. The most accurate methods for simulating deformable motion are based on FEM (finite-element meshing) [1]. However, even for a moderately complex object shape, we need to discretely represent the deformable model using thousands of vertices, i.e. tens of thousands of DOFs. To simulate deformable models using a high-dimensional representation, a conventional FEM algorithm has a cubic complexity in the number of vertices and an accurate simulation can take hours on a desktop PC [2]. For example, in many assembly planning applications, the underlying objects may undergo large deformations, which adds to the computational complexity. Prior planning algorithms for deformable robots [3], [4], [5], [6], [7] have been designed for simpler deformable models. These models either correspond to simple geometric shapes such as a linear rod or may undergo small, local deformations. These techniques may not be practical in terms of modeling large deformations of general volumetric deformable robots with thousands of vertices. Other methods have been designed for deformable body registration [8], which can be used to manipulate passive deformable bodies [9].

A driving application for our work is the control and planning of arbitrary deformable bodies with internal motors [10]. This problem has rich applications in biology for modeling boneless animals [11], in computer graphics for generating animations [12], and in robotic design and control [13]. Compared with motion planning using external forces such as is demonstrated in [14], our problem is more challenging because the control forces correspond to only internal forces and the deformable robot must move around by interacting with the environment. Prior techniques used for planning and control are based on differential dynamic programming [15], reinforcement learning [16], and sampling-based methods [17]. However, these methods have been limited to articulated models, and are not practical for very high-DOF deformable models.

Main Result: We present a novel planning algorithm for high-DOF deformable bodies using a two-stage learning algorithm, where we learn a low-level control policy in the first stage and a high-level motion planner in the second stage. In the first stage, we train a multitask controller parameterized using DMP functions [18], [19]. A DMP function encodes a deformable body's movement or locomotion gaits using very few parameters, making it possible to directly optimize these parameters (Section IV-A). We use subspace FEM approximation and contact invariant optimization for efficient DMP-based controller optimization. In the second stage, we train a neural-network motion planner using deep Q-Learning (DQL), which selects the DMP task to navigate the deformable body to a given target position while avoiding obstacles. This training typically requires sampling millions of simulated deformable body states. To accelerate this computation, we use a simplified dynamic model, which is identified from DMP-controlled simulations and perform DQL on the simplified model (Section IV-B).

We evaluate the performance of our planning on 3 types of deformable models and 2 kinds of locomotion: an underwater swimming fish (6354-DOF), a virtual T-shaped walker (9363-DOF), and a 4-legged deformable walker/swimmer (3162-DOF) (Section V). These are two widely used locomotion patterns used to move a robot around. In both cases, our two-stage navigation planner is trained within a day of computation on a desktop machine and can be executed at realtime in our simulated environment. We also show extensions to various environments, obstacle shapes, and input features such as laser range scanner data.

[1] Department of Computer Science, the University of North Carolina at Chapel Hill {zherong,dm}@cs.unc.edu

II. Related Work

We briefly review related work on deformable model simulation, planning and manipulation of deformation objects, trajectory optimization, and reinforcement learning.

Deformable Model Simulation: The most widely used method for deformable simulation is FEM [1]. It has been shown [2] that the FEM method is capable of simulating very large, nonlinear, and complex deformations. The high computational cost makes it difficult to develop efficient control algorithms for these models. For example, in [12], it is only possible to control FEM methods over a very short horizon. In this paper, we use a subspace FEM model [20], [21], [22] to represent complex deformable robots. These methods provide two orders of magnitude speedup over conventional FEM methods. We combine this subspace model with efficient planning and learning techniques to provide a complete framework for deformable body control and planning.

Deformable Body Planning and Manipulation: There is extensive literature on motion planning and manipulation of deformable models, as surveyed in [4]. Most prior planning algorithms make different assumptions about deformable models. These assumptions include a reduced configuration space without any dynamics properties [23] or passive deformable models with no actuation [24], [25], [5], [26], [6], [7]. Most such deformable models undergo deformations based only on external forces. Other methods [14] consider deformable bodies undergoing motion due to internal and external forces, but they are not very efficient for very high-DOF robots. Other planning algorithms focus on different kinds of deformable models such as fluids [27] or ropes [28].

Trajectory Optimization: There is extensive work on trajectory and controller optimization. Those techniques have also been used for motion planning of deformable models [29], [30], [31], [32], [33], [34]. Our work extends these trajectory optimization methods during the first stage of our learning method.

Learning-based Motion Planning: Learning techniques are increasingly used to design efficient motion planning algorithms. In terms of deformable models, many learning-based algorithms [35], [36] have been designed for passive deformable models and they use imitation learning, which requires human demonstrations. In our method, we use a neural-network trained using DQL [37] as the motion planner.

III. Problem Formulation

Figure 1 illustrates our overall idea of motion planning for deformable bodies. The input to our method is an arbitrary volume mesh representing the shape of a deformable body. The kinematic configuration of the body is represented by the position of all the vertices, denoted as x, whose dimension $|x|$ is proportional to the number of vertices. In practice, $|x|$ is several thousand for a moderately complex deformable model. In this configuration space, we discretize the body's potential and kinetic energy using the FEM method [16], and denote the resulting dynamic system as $x_{i+1} = f(x_i, u_i, \Delta t)$, where f is the FEM time integrator, Δt is the timestep size, and u_i is the control input at timestep i. f takes both internal potential forces and external forces into consideration. In our approach, we consider two kinds of external forces: fluid drag force for underwater swimming bodies and frictional contact force for walking bodies. The actual formulation of u_i is application dependent. We assume that u_i corresponds to the target shape of deformation, which means that we achieve control by adding additional energy, $\|x_{i+1} - u_i\|^2$, to the dynamic system f.

The function f is computationally costly to evaluate because the cost is superlinear in $|x|$. This is the main challenge for performing in the planner and controller optimizations, where many f evaluations are required. To overcome this complexity, we use a subspace FEM approximation [20], [21], [22]. These methods are based on the assumption that, in the high-dimensional configuration space, most salient elastic deformations lie on a low-dimensional manifold. Moreover, if we apply a special nonlinear coordinate transformation on x, this manifold is mapped to a linear space [22]. We call this the transformed rotation-strain (RS) space and denote the transformation function as $\text{RS}(\bar{x}) = x$, where $\bar{x}$ is a low-dimensional coordinate, and $|\bar{x}| < 20$ in most cases. The RS transformation can be considered as a nonlinear dimensionality reduction computation [38], which is used for elastic deformations. Pan et al. [22] showed that we can construct a low-dimensional counterpart of f in RS space, denoted as follows:

$$\text{RS}(\bar{x}_{i+1}) = f(\text{RS}(\bar{x}_i), \bar{u}_i, \Delta t),$$

which can be rewritten as :

$$\bar{x}_{i+1} = \bar{f}(\bar{x}_i, \bar{u}_i, \Delta t).$$

This $\bar{f}$ is also an FEM dynamics model, where the configuration space is constrained to the low-dimensional manifold of salient elastic deformations. Due to the reduced dimension, function $\bar{f}$ can be evaluated efficiently (e.g., more than 30Hz on a desktop machine), which greatly accelerates controller optimization.

After constructing the transformation $\text{RS}(\bullet)$ and the reduced dynamic simulator $\bar{f}$, the goal of our planning algorithm is to optimize a controller $\bar{u} = \pi(\bar{x})$ to navigate the deformation body to a given goal position in the world coordinate system, i.e. minimizing $\|\text{COM}(\bar{x}) - T\|^2$ while avoiding collisions with the obstacles. In this case, COM refers to the deformable body's center of mass.

IV. Motion Planning with Two Stage Learning

In this section, we introduce our two-stage training procedure for the controller $\pi(\bar{x})$, as illustrated in Figure 2. In the first stage, we train a low-level controller that encodes a deformable robot's movement gaits, such as swimming or walking in different directions. The controller is parameterized using a multitask DMP function [18], [19]. In the second stage, we learn a high-level motion planner parameterized using neural-network and deep Q-Learning [37]. The goal of our neural-network planner is to select the DMP task for navigation. Compared with end-to-end methods such as

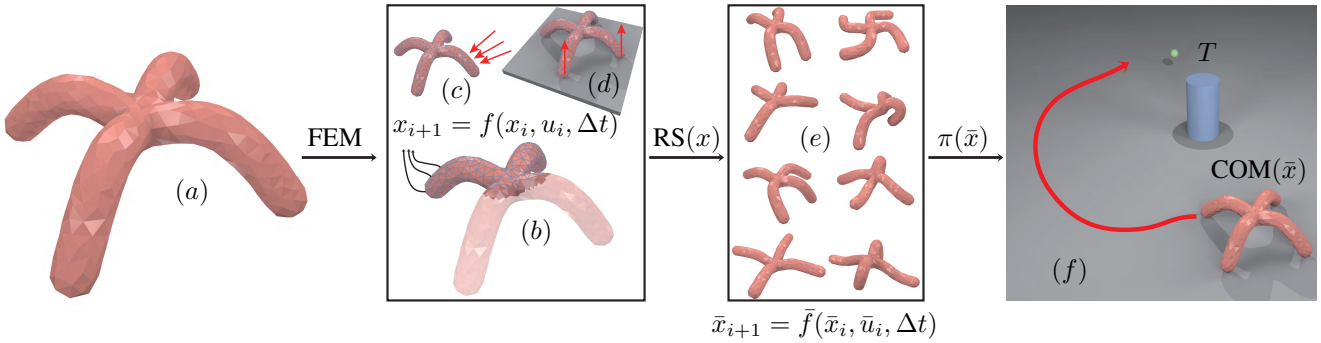

Fig. 1: Our overall idea of motion planning for a deformable body: Given an arbitrarily-shaped deformable body (a), we use the FEM method to discretize the governing dynamics equation, which is denoted as $x_{i+1} = f(x_i, u_i, \Delta t)$ (b). This formulation also takes into account various external forces. These forces include fluid drag force for swimming (c) and frictional contact force for walking (d). Function f is costly to evaluate, so we apply the RS transformation [22] to derive a low-dimensional dynamic system that captures most of the deformations on a low-dimensional manifold. Some examples of the deformations on this manifold are given in (e). We highlight the path computed using our planner to navigate the deformable body to reach a target (green) while avoiding obstacles (blue) (f), where our method can handle different obstacle shapes and input environmental features.

[39], two-stage training has two major advantages. First, since DMP is a smooth function, we can use a gradient-based method to optimize the DMP controller, which is computationally more efficient than using a sampling-based reinforcement learning algorithm, e.g., [16]. In addition, we can reuse the trained DMP controller to accomplish multiple navigation tasks and adapt to different environments and obstacle shapes by only retraining the neural-network planner.

A. Optimizing Multitask DMP Controller

In this section, we present our method to optimize the DMP controller [18], [19], which encodes each deformable body's locomotion skills. DMP is a widely used open-loop controller parameterization. Here we use the following rhythmic multitask DMP representation:

$$\bar{u}_i^j = \text{DMP}^j(i\Delta t) \triangleq \sum_k^D \alpha_k^j \mathbf{exp}(\beta_k^j \mathbf{cos}(\tau i \Delta t - \mu_k)),$$

where $\alpha_k^j, \beta_k^j, \tau, \mu_k$ are the DMP parameters to be optimized, denoted as $w_{\text{dmp}} \triangleq (\alpha_k^j\ \beta_k^j\ \tau\ \mu_k)$. Here $\alpha_k^j, \beta_k^j, \mu_k$ are vector-valued and represent activation magnitude, activation time, and phase shift, respectively. D is the number of DMP elements. For a multitask controller, we use different α, β for different tasks, while keeping the period τ and the phase shift μ the same for all tasks. We use superscript j to denote the task id, e.g. the direction to walk or swim. This leads to synchronized rhythmic movements of the same deformable body performing different tasks [19].

The multitask DMP controllers are trained using gradient-based trajectory optimization. Specifically, for a K_1-task DMP controller, we optimize K_1 trajectories, each with N_1 timesteps, solving the following optimization problem:

$$\underset{\bar{x}_i^j, \bar{u}_i^j, w_{\text{dmp}}}{\mathbf{argmin}} \quad E_{\text{phys}} + E_{\text{obj}} + E_{\text{dmp}} + E_{\text{cio}}, \tag{1}$$

where we simultaneously find the trajectory $\bar{x}_i^j$, control input $\bar{u}_i^j$ for task j, and DMP parameters w_{dmp}, which minimizes four objective terms. $E_{\text{phys}} \triangleq \sum_j^{K_1} \sum_i^{N_1} \|\bar{x}_{i+1}^j - \bar{f}(\bar{x}_i^j, \bar{u}_i^j, \Delta t)\|^2$ penalizes the violations of the dynamics equation. E_{obj}^j is the high-level objective function that formulates the goal of the controller. For example, to have a deformable body move to T^j, we have $E_{\text{obj}} \triangleq \sum_j^{K_1} \|\text{COM}(\bar{x}_{N_1}^j) - T^j\|^2$. $E_{\text{dmp}} \triangleq \sum_j^{K_1} \sum_i^{N_1} \|\bar{u}_i^j - \text{DMP}^j(i\Delta t)\|^2$ ensures that $\bar{u}_i^j$ is consistent with the output of the DMP controller. Finally, for a walking deformable body, the frictional contact forces are optimized using CIO cost term, E_{cio}, as detailed in [29]. In practice, we use an alternating direction method to minimize the above objective function. First, we fix w_{dmp} and optimize $\bar{x}_i, \bar{u}_i$ using Newton method. Next, we fix $\bar{x}_i, \bar{u}_i$ and optimize w_{dmp} using Quasi-Newton method.

In the basic formulation, we use open-loop controllers to control swimming deformable bodies. However, for contact-rich locomotion such as walking, an open-loop controller is not robust enough to force noises and model discrepancy during online control phase. In this case, we use an iterative LQG algorithm [15] during the online control, using $E_{\text{obj}}^j + E_{\text{dmp}}$ as the cost function, over a control horizon of $5\Delta t$. In this way, we extend the DMP controller to a feedback controller that follows the DMP output as much as possible.

B. Optimizing Neural-Network Planner

After the multitask DMP is trained, we use a neural-network to select the DMP task for motion planning to navigate the deformable body to a target position while avoiding obstacles. This is a standard reinforcement learning problem with discrete action space and continuous state space. DQL [16] has been proven to be very effective to solve this kind of Markov Decision Process.

As illustrated in Figure 2 (c), we use a fully connected neural-network with 3 hidden layers, each having 64 neurons and a BNLL activation function [40]. The input to our neural-network is the current RS-space state $\bar{x}$ of the deformable body and the environmental information, i.e. the relative position of obstacles with respect to COM$(\bar{x})$. The output is the expected return after applying each DMP task, from which we pick the DMP task (i.e. superscript j) leading to the highest return and run the selected DMP-controlled simulation.

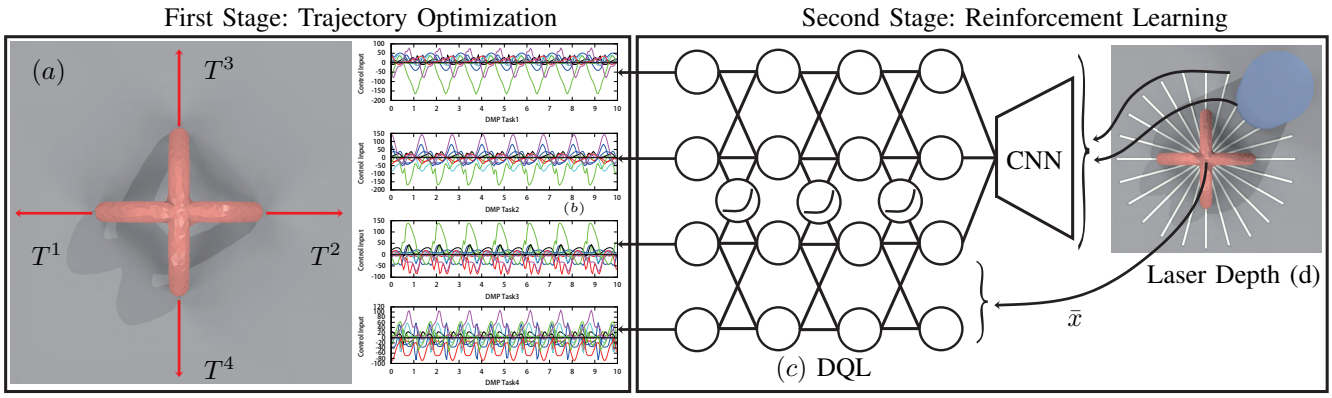

Fig. 2: An illustration of the structure of our control policy ($\pi(\bar{x})$ in Figure 1) and its two-stage training procedure. For the 4-legged deformable walker, our first stage simultaneously optimizes 4 walking trajectories in different directions (a), giving us a 4-task DMP controller (b). Our second stage uses the DQL to train a 3-layer, fully connected neural-network (c) to select the DMP task for navigation. The inputs to our neural-network are the current state of the deformable body, $\bar{x}$, and the feature encoding the current environment information (d). In our benchmarks, the environment information is represented based on the distance to the obstacles along a set of fixed directions shown in white in (d), preprocessed by a convolutional neural-network (CNN).

In our benchmarks, we use a more general setting and assume that each deformable body is a robot equipped with a 360° laser scanner. The depth readback of each laser beam can be used as the environmental feature (Figure 2 (e)). This feature is preprocessed by a CNN block. Our CNN has 3 convolutional layers: 16 filters of kernel size 8, followed by 16 filters of kernel size 4, followed by another 16 filters of kernel size 4, and finally followed by a fully connected layer with 32 output features. This CNN is pre-trained using supervised learning to predict the relative obstacle position with respective to COM($\bar{x}$).

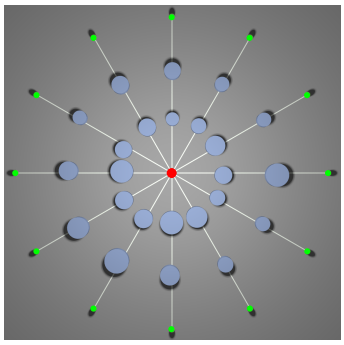

Fig. 3: The DQL learning task for the 4-legged walker (red in the middle). We sample $K_2 = 12$ evenly distributed distant targets (green), and the white line indicates the relative position against the deformable body. On the way to each target, we put 2 cylindrical obstacles (blue circles) with random sizes and center offsets.

Our DQL training uses a standard algorithm [16]. For each deformable body and environmental setting, we randomly generate K_2 distant navigation targets, T^j, which are distributed evenly around the body. For each target, we put two obstacles along the straight line between the initial COM($\bar{x}$) and T^j. During each DQL iteration, we sample K_2 trajectories with fixed length N_2 for all the T^j. This setting is illustrated in Figure 3. At each timestep i, our reward function is defined as:

$$R(\bar{x}_i, \bar{x}_{i+1}, \bar{u}_i, T^j) = \|\text{COM}(\bar{x}_i) - T^j\|^2 - \|\text{COM}(\bar{x}_{i+1}) - T^j\|^2 - 1000\text{coll}(\bar{x}_{i+1}), \quad (2)$$

where coll($\bar{x}$) is an indicator of whether the deformable body in state $\bar{x}$ is in collision with any obstacles.

However, running the algorithm described in [16] under this setting is still computationally impractical, as a typical DQL training requires millions of state samples. In our case, each state sample requires a time integration using $\bar{f}$ and a collision check to determine coll($\bar{x}$). We use two techniques to accelerate training. First, we exploit the fact that the goal of the neural-network is to select the DMP task and the DMP output is rhythmic. As a result, we can query the neural-network at a frequency of $\lceil \frac{2\pi}{\tau \Delta t} \rceil$, i.e., only between consecutive DMP control cycles. Second, it is intuitive to expect that each DMP control cycle does not change a body's deformable state. It only moves the deformable body rigidly. Therefore, we propose avoiding evaluating the actual $\bar{f}$ and run DQL using a cheap surrogate simulator $\bar{f}_{fit}$. We assume that running one DMP control cycle essentially applies a rigid affine transformation to the deformable body. This rigid

Algorithm 1 Two-Stage Deformable Body Planner Learning

Input: An arbitrarily-shaped deformable body mesh
▷ Building subspace FEM model [22] (Section III)
Build RS transformation function $\bar{x} = \text{RS}(x)$
Build subspace FEM simulator $\bar{x}_{i+1} = \bar{f}(\bar{x}_i, \bar{u}_i, \Delta t)$
▷ First Stage: train DMP controller (Section IV-A)
Pick K_1 targets: $T^{1,\cdots,K_1}$ and solve Equation 1
▷ Fit simplified dynamic system (Section IV-B)
Initialize tuple set $S^j = \emptyset \quad \forall j = 1, \cdots, K_1$
Initialize deformable state $\bar{x}$
while $\exists j, \quad |S^j| < 1000$ **do**
    pick random task id j
    $\bar{x}_{old} \leftarrow \bar{x}$
    for $i = 1, \cdots, \lceil \frac{2\pi}{\tau \Delta t} \rceil$ **do**
        $\bar{x} \leftarrow \bar{f}(\bar{x}, \text{DMP}^j(i\Delta t), \Delta t)$
    end for
    $T^j \leftarrow S^j \cup \{(\bar{x}_{\text{old}}, \bar{x})\}$
end while
for $j = 1, \cdots, K_1$ **do**
    Fit rigid transformation R^j, T^j from S^j using [41]
    define $\bar{f}_{\text{fit}}(\bar{x}, \text{DMP}^j)$ as rigid transformation R^j, T^j
end for
▷ Second Stage: Train motion planner (Section IV-B)
Pick K_2 targets: $T^{1,\cdots,K_2}$ around initial deformable body
for $j = 1, \cdots, K_2$ **do**
    Put 2 obstacles on the way to T^j
end for
Run [16] on $\bar{f}_{\text{fit}}(\bar{x}, \text{DMP}^j)$ to maximize cumulative reward Equation 2 for all K_2 trajectories

transformation is parameterized by a 3×3 orthogonal rotation matrix, R, and a 3×1 translation, T. To find R and T that best represent the state changes caused by DMP control cycles, we run random DMP-controlled FEM simulations by randomly jumping from one DMP task to another and collecting tuples $(\bar{x}^j_{before}, \bar{x}^j_{after})$ of states before and after running one DMP control cycle of task j. After we have collected 1000 tuples for each j, we use an optimal rigid transformation algorithm [41] to compute the globally optimal R and T. This defines our surrogate simulator $\bar{f}_{\text{fit}}$, which is just a rigid transformation. DQL training with $\bar{f}_{fit}$ becomes much faster. We summarize our method in Algorithm 1.

V. Implementation and Performance

In this section, we describe our implementation and highlight the performance on complex deformable models.

Example	#vertices	$\lvert\bar{x}\rvert$	cost f	cost $\bar{f}$	K_1	N_1	D	cost DMP	cost DQL	cost CNN
Fish Swimming	2118	11	2.5s	0.07s	3	200	5	1.7h	2.5h	0.7h
T-shaped Walking	3121	16	2.2s	0.13s	4	200	5	3.2h	3.3h	1.2h
4-legged Walking	1054	16	2.1s	0.11s	4	200	5	2.5h	3.3h	1.51h
4-legged Swimming	1054	16	5.2s	0.17s	5	200	5	5.2h	11.0h	9.12h

TABLE I: Computational cost of each step of our learning method. From left to right: name of example, number of vertices in volume mesh, dimension of identified manifold in RS space, cost of evaluating f, cost of evaluating $\bar{f}$, number of DMP tasks, trajectory length for DMP controller optimization, number of DMP elements, cost of solving Equation 1 (in hours), cost of DQL training (in hours), and cost of CNN feature learning (in hours). Note that the simulation cost in subspace is related with number of bases, $|\bar{x}|$, instead of #vertices.

Computational Cost: We validate our method using 2 deformable body shapes performing 3 kinds of locomotion tasks: a fish swimming, a 4-legged body walking, and a 4-legged body swimming. Table I summarizes the parameters of these deformable bodies and the training setup. The original FEM model has 6354-DOF for the fish, 9363-DOF for the T-shaped body, and 3162-DOF for the 4-legged body. After the RS transformation, the low-dimensional configuration spaces have 11-DOF for the fish and 16-DOF for the 4-legged body and T-shaped body. However, simulating these relatively small dynamic systems is still quite computationally costly. Each evaluation of $\bar{f}$ takes 0.1 seconds on an 8-core CPU. Even using our reduced dynamics, the DQL learning still takes more hours than DMP controller optimization.

Multitask DMP Controller Performance: To optimize the multitask DMP controller, we have to manually design E_{obj} for all $1 \leq j \leq K_1$. For our 4 examples illustrated in Figure 4, we design objectives so that different navigation skills are needed by the neural-network planner. For the swimming fish, we choose $K_1 = 3$ and set $T^{1,2,3}$ at the west, north-west, and south-west corners, respectively, so that the fish must turn in different directions while swimming forward. For the 4-legged walking body and T-shaped walking body, we choose $K_1 = 4$ and set $T^{1,2,3,4}$ at the east, west, south, and north corners, respectively. Finally, for the 4-legged swimming body, we choose $K_1 = 5$ and set T^5 at the east corner. However, we set all other targets to be zero (i.e., $T^{1,2,3,4} = \mathbf{0}$) so that the 4-legged body does not move. In addition, we require that the orientation of the 4-legged swimmer changes to different directions by using an additional objective energy term $E_{\text{obj}} = \|\mathrm{R}^j(\bar{x}_N)y - y^j\|^2$, where $y = \begin{pmatrix}0 & 1 & 0\end{pmatrix}$ is the +y direction in the deformable body's local coordinate of reference, and y^j is the target orientation set to $y^{1,2,3,4} = \begin{pmatrix}\pm1 & 0 & \pm1\end{pmatrix}$. Unfortunately, the reward is non-zero only for task 1, and zero otherwise, so that DQL suffers from the well-known "distal reward problem". Note that in all these examples, the deformable body is a non-holonomic robot because they can only navigate along a set of discrete directions dictated by the DMP controller, insteading of all possible directions. As a result, our neural-net is a non-holonomic motion planner.

Example	K_2	N_2	memSize	$\lceil\frac{2\pi}{\tau\Delta t}\rceil$	#$\bar{f}$	#$\bar{f}_{\text{fit}}$
Fish Swimming	10	$200\lceil\frac{2\pi}{\tau\Delta t}\rceil$	20k	11	22k	2k
T-shaped Walking	10	$400\lceil\frac{2\pi}{\tau\Delta t}\rceil$	40k	23	92k	4k
4-legged Walking	10	$400\lceil\frac{2\pi}{\tau\Delta t}\rceil$	40k	23	92k	4k
4-legged Swimming	30	$800\lceil\frac{2\pi}{\tau\Delta t}\rceil$	240k	9	216k	24k

TABLE II: Parameters of DQL training in the 4 examples. From left to right, name of example, number of trajectories, trajectory length measured by number of DMP control cycles, equivalent memory bank size ($K_2 \times N_2 \times 10$), number of timesteps in each DMP control cycle, expected number of calls to $\bar{f}$ in each DQL iteration using accurate FEM dynamics ($K_2 \times N_2$), number of call to $\bar{f}_{\text{fit}}$ in each DQL iteration using fitted dynamics ($K_2 \times N_2/\lceil\frac{2\pi}{\tau\Delta t}\rceil$).

Neural-Network Planner Performance: For each DQL training, we run 10^4 iterations of [16]. In each iteration, we sample K_2 trajectories of length N_2 and then update neural-network parameters by running RMPProp iterations with memory bank size $(K_2 \times N_2 \times 10/\lceil\frac{2\pi}{\tau\Delta t}\rceil)$ and batch size 32, i.e., the state samples of the most recent 10 DQL iterations are stored in the memory bank. The parameters of DQL training for the three examples are summarized in Table II. We pick our targets very carefully. Each T^j is located 20 meters away from the initial COM$(\bar{x}^j_0)$, where the average body size is 1 meter. In addition, we pick our targets so that they evenly cover all possible moving directions. The fish swimming and the 4-legged/T-shaped walking examples are essentially 2D motion planning problems where we can use evenly distributed targets, as illustrated in Figure 3. The 4-legged swimming is a 3D motion planning problem, for which we use more targets ($K_2 = 30$) to cover all possible 3D directions. These directions are selected using spherical Poisson disk sampling. Moreover, since the 4-legged body cannot move during orientation adjustment, we use a longer trajectory size ($N_2 = 800\lceil\frac{2\pi}{\tau\Delta t}\rceil$). This is why DQL training for 4-legged swimming is more costly in Table I. The convergence history of DQL is plotted in Figure 5.

From the last two columns of Table II, we can see that using $\bar{f}_{\text{fit}}$ instead of $\bar{f}$ leads to at least two orders of magnitude speedup, because the number of samples is reduced by an order of magnitude and each evaluation of $\bar{f}_{\text{fit}}$ is more than 10 times faster than $\bar{f}$. However, it is still unclear

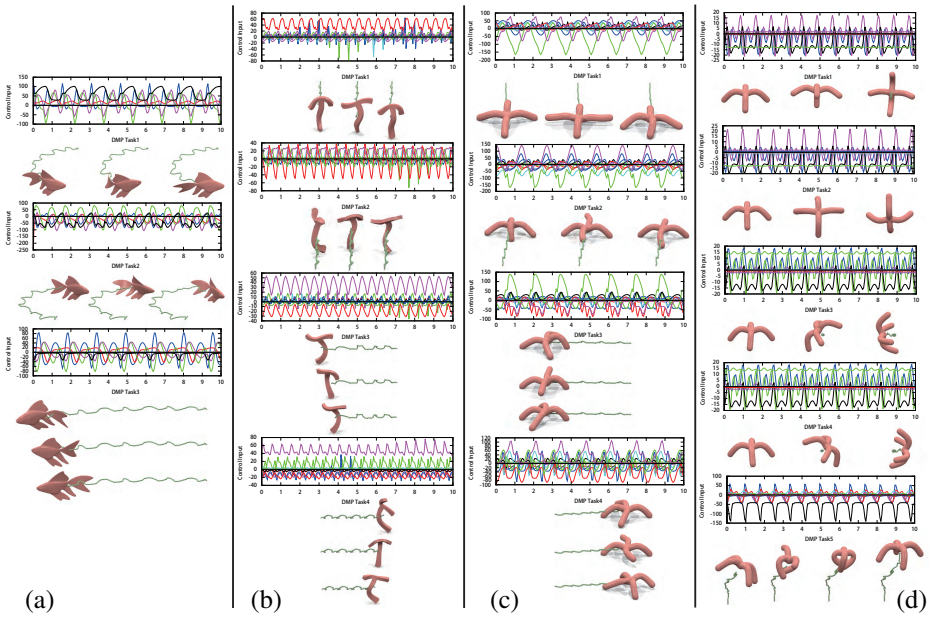

Fig. 4: For each of the K_1 tasks in our 4 examples, we plot the multitask DMP control commands on the deformable body (each control command has $|\bar{x}|$ DOFs shown as different curves in the plot). We also show the controlled trajectory of the center of mass (green curve). (a): a fish controlled by a 3-task DMP to swim left, right, and forward (from top to bottom). (b): a T-shaped walker controlled by a 4-task DMP to walk forward, back, left, and right. (c): a 4-legged walker controlled by a 4-task DMP to walk forward, back, left, and right. (d): a 4-legged swimmer controlled by a 5-task DMP to turn to the front, back, left, and right, and to swim forward. Its swimming gaits are similar to those of a real-life jellyfish. In this case, only the last task changes COM($\bar{x}$), leading to a "distal reward problem" in reinforcement learning. From these examples, we can see that both the control command and controlled trajectory are periodic. This motivates us to use proxy dynamics fitted from each cycle of the DMP controller to accelerate DQL training.

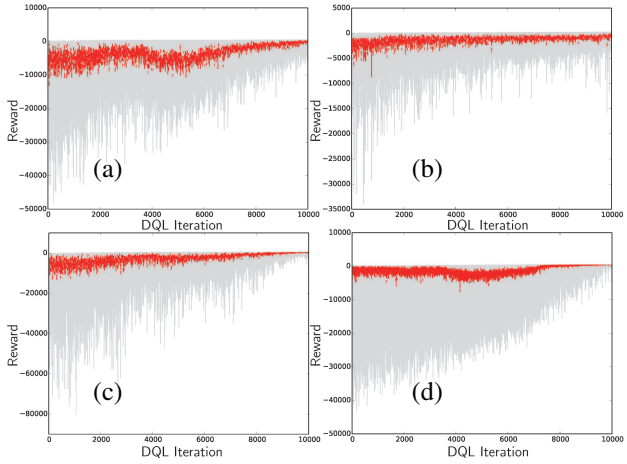

Fig. 5: The reward (Equation 2) of the K_2 trajectories plotted against DQL iteration numbers for fish swimming (a), 4-legged walking (b), T-shaped walking (c), and 4-legged swimming (d) examples. (Gray: minimum, maximum reward. Red: mean reward.) During DQL training, most of the negative rewards (gray region) are due to collisions and most efforts of DQL are spent on learning to avoid collisions (convergence of gray region). At the final stage of training where collisions rarely happen, most efforts of DQL are spent on moving towards goals, leading to a gain in positive rewards (convergence of the red region).

how well $\bar{f}_{\text{fit}}$ approximates one cycle of DMP controlled trajectory. To measure this approximation quality, we notice that $\bar{f}_{\text{fit}}$ is just an affine transformation fitted from sampled data S^j in Algorithm 1. Therefore, we summarize the mean and standard deviation of S^j in Table III. We can see that either the standard deviations are very small compared with mean value or they are both small. This means that an affine transformation is a very good approximation of the controlled trajectory. Therefore, we only take the mean value and run DQL on deterministic dynamics $\bar{f}_{\text{fit}}$. When the standard deviations are large, one can consider more sophisticated methods such as stochastic $\bar{f}_{\text{fit}}$ using GMM, such as in [16].

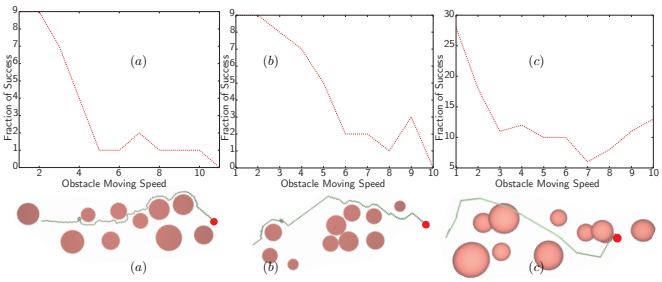

Fig. 6: Top: The number of successful planning trajectories under different dynamic obstacle moving speeds (in m/s). Bottom: The trajectory (green) generated using our neural-network planner to reach the target (red dot) in realtime. Note that the first two examples are 2D so that the obstacles are cylindrical while the last example is 3D so that obstacles are spherical ((a): fish swimming with $K_2 = 10$, (b): 4-legged walking with $K_2 = 10$, and (c): 4-legged swimming with $K_2 = 30$).

Applications and Extensions: After two-stage training, we test our planner using accurate FEM dynamics $\bar{f}$. Due to the subspace FEM approximations and fast neural-network evaluation, the online performance of our method is always in realtime. We first test the planner on the K_2 trajectories used during DQL training, and all 4 policies achieve a 100% rate of success, i.e. reaching each T^j without touching obstacles. Next, we run the tests using the novel target positions. In addition, we add 10 obstacles on the way to each T^j, instead of the 2 obstacles in DQL training. Figure 6 illustrates the resulting trajectory of the 4 examples.

Since our training only takes static obstacles, our planner has limited capability to handle dynamic obstacles. To measure this ability, we pick a random direction for each obstacle to move in each of the K_2 trajectories. Next, we generate the K_2 trajectories for each example using different obstacle moving speeds. In the top figure in Figure 6, we plotted the number of successful trajectories under different speeds.

An advantage of two-stage training is that we can retrain the neural-network for new navigation tasks with different

Example	$AVG(R^j)$	$SD(R^j)$	$AVG(t^j)$	$SD(t^j)$
Fish Swimming	0.833/0.969/0.131	0.085/0.048/0.044	1.409/1.424/1.484	0.096/0.087/0.058
T-shaped Walking	0.009/0.012/0.001/0.02	0.0/0.0/0.03/0.001	0.672/0.643/0.522/0.541	0.007/0.012/0.031/0.016
4-legged Walking	0.083/0.132/0.071/0.129	0.104/0.107/0.107/0.105	1.963/0.67/0.93/1.64	0.269/0.127/0.165/0.305
4-legged Swimming	0.184/0.206/0.229/0.211/0.971	0.129/0.121/0.118/0.122/0.274	0.09/0.087/0.089/0.089/0.426	0.076/0.073/0.05/0.066/0.055

TABLE III: The original RS dynamics model, $\bar{f}$, is still too costly for DQL training, so we use fitted dynamics, the affine transformation $\bar{f}_{fit}$. We measure how well $\bar{f}_{fit}$ approximates $\bar{f}$ by measuring the mean (AVG) and standard deviation (SD) in training data S^j (global rotation R^j and translation t^j of the deformable body for task j over one DMP control cycle). We can see that the standard deviations are all very small, so $\bar{f}_{fit}$ is a very good approximation and we can run DQL on $\bar{f}_{fit}$ as a deterministic dynamic model without introducing any state uncertainty, unlike in [16].

environment features and obstacle shapes. In Figure 7, we show such an example where the fish is swimming through a tunnel. The fish starts at one end and the T is at the other end. We again use depth data as the environmental feature, but the CNN is not pre-trained in this case. It is combined with the 3-layer neural-network to perform end-to-end DQL training. This result verifies that we can handle multiple motion planning tasks by just replacing the motion planner, without changing the set of movement gaits. A similar idea is presented in [42] for articulated robots.

Analysis and Comparisons: We notice that there are many concurrent works on controller optimization [30], reinforcement learning [43], and reinforcement learning for motion planning [44]. These methods mainly deal with the robustness and versatility of learning algorithms for articulated robots. Instead, our focus is on the efficiency of handling very high-dimensional configuration spaces corresponding to deformable robots, and the design of efficient algorithms for planning controller optimization. As a result, we decompose the pipeline of optimization into two learning stages, so that each stage can be accomplished in several hours on a desktop machine. If we do not use $\bar{f}_{fit}$ and directly run DQL on $\bar{f}$, then the training takes more than 48 hours for all 4 examples. Finally, the two-stage algorithm provides us with additional benefit that the control becomes modular and we can replace only the motion planner to extend our method to different environments and obstacle shapes, without retraining the DMP controller.

VI. Limitations and Conclusion

We present a learning-based algorithm for planning high-DOF deformable bodies. The computationally costly FEM dynamics is first reduced using subspace FEM approximations. Then, a two-stage learning is used to parameterize the control policy. This allows efficient learning algorithms to be used in each stage, and efficient retraining for new planning tasks. We have shown that the method can plan deformable bodies with thousands of DOFs, in less than a day of training on a desktop machine.

Our method still has several limitations. During first stage, our method uses DMP as our underlying controller so that we inherit some of its limitations. DMP is an open-loop controller and we could use a general feedback controller, such as neural-networks, to provide better robustness. And the number of DMP tasks K_1 and the type of these tasks must be manually selected. Moreover, our fast DQL training relies on the fact that we use rhythmic DMP controller. Although locomotion gaits used for navigation are mostly rhythmic, it is worth extending our method to non-rhythmic cases so that the method can be used for more general tasks than navigation.

During second stage, we currently use static obstacles and simple obstacle shapes for training the planner. Although our method is much faster than a conventional sampling-based motion planner such as RRT used in [34], our planner makes its decision based only on local environmental informations. Without global information, our deformable body sometimes get stuck by moving back and forth without approaching the target. This artifact is more obvious when we introduce more obstacles. It is worth exploring ways to take global information into consideration, as is done in [45]. A finally limitation with our planner is that it selects navigation direction only in-between DMP-cycles. Therefore, it cannot avoid obstacles within one DMP cycle.

Promising future work includes using a single unified deep neural-network for control and motion planning as in [44]. However, we expect such training to be much more time consuming. Although the subspace FEM model [22] provides much better simulation performance, subspace FEM simulation is still slower than articulated body simulation [46]. So that further acceleration and algorithm optimization for deformation body simulation is very useful tool for more complex controller optimization problems.

Acknowledgement

This work is supported in part by ARO grants W911NF16-1-0085 and W911NF-17-1-0181, and Intel.

References

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

[2] G. Irving, C. Schroeder, and R. Fedkiw, "Volume conserving finite element simulations of deformable models," in *ACM SIGGRAPH 2007 Papers*, ser. SIGGRAPH '07. New York, NY, USA: ACM, 2007.

[3] F. Lamiraux and L. E. Kavraki, "Planning paths for elastic objects under manipulation constraints," *The International Journal of Robotics Research*, vol. 20, no. 3, pp. 188–208, 2001.

[4] P. Jimnez, "Survey on model-based manipulation planning of deformable objects," *Robotics and Computer-Integrated Manufacturing*, vol. 28, no. 2, pp. 154 – 163, 2012.

[5] S. Patil, J. van den Berg, and R. Alterovitz, "Motion planning under uncertainty in highly deformable environments," *Robotics science and systems: online proceedings*, 2011.

[6] S. Rodriguez, J.-M. Lien, and N. M. Amato, "Planning motion in completely deformable environments," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE, 2006, pp. 2466–2471.

[7] M. Moll and L. E. Kavraki, "Path planning for deformable linear objects," *IEEE Transactions on Robotics*, vol. 22, no. 4, pp. 625–636, Aug 2006.

[8] A. X. Lee, H. Lu, A. Gupta, S. Levine, and P. Abbeel, "Learning force-based manipulation of deformable objects from multiple demonstrations," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 177–184.

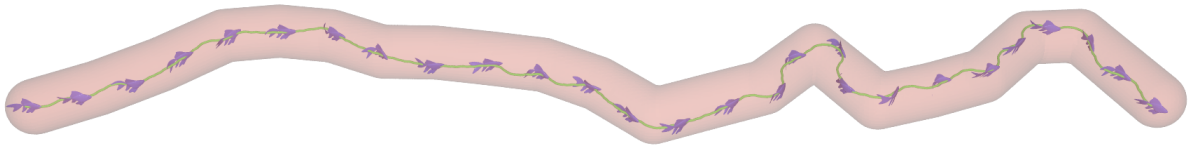

Fig. 7: Retraining DQL for a new navigation task, using an old multitask DMP controller. We drive the fish to swim through the tunnel (red tube). The planned trajectory is shown as the green curve and the intermediary fish states are illustrated in blue.

[9] D. K. Pai, K. v. d. Doel, D. L. James, J. Lang, J. E. Lloyd, J. L. Richmond, and S. H. Yau, "Scanning physical interaction behavior of 3d objects," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001, pp. 87–96.

[10] D. Rus and M. T. Tolley, "Design, fabrication and control of soft robots." *Nature*, vol. 521 7553, pp. 467–75, 2015.

[11] J. Kajtar and J. Monaghan, "On the swimming of fish like bodies near free and fixed boundaries," *European Journal of Mechanics-B/Fluids*, vol. 33, pp. 1–13, 2012.

[12] J. Tan, G. Turk, and C. K. Liu, "Soft body locomotion," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, p. 26, 2012.

[13] Y. Sugiyama and S. Hirai, "Crawling and jumping by a deformable robot," *The International journal of robotics research*, vol. 25, no. 5-6, pp. 603–620, 2006.

[14] 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.

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

[16] W. Han, S. Levine, and P. Abbeel, "Learning compound multi-step controllers under unknown dynamics," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 6435–6442.

[17] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[18] S. Schaal, "Dynamic movement primitives-a framework for motor control in humans and humanoid robotics," in *Adaptive motion of animals and machines*. Springer, 2006, pp. 261–280.

[19] E. Rückert and A. d'Avella, "Learned parametrized dynamic movement primitives with shared synergies for controlling robotic and musculoskeletal systems," *Frontiers in computational neuroscience*, vol. 7, p. 138, 2013.

[20] J. Barbič and D. L. James, "Real-time subspace integration for st. venant-kirchhoff deformable models," *ACM Transactions on Graphics / SIGGRAPH*, vol. 24, no. 3, pp. 982–990, Aug. 2005.

[21] S. S. An, T. Kim, and D. L. James, "Optimizing cubature for efficient integration of subspace deformations," in *ACM transactions on graphics (TOG)*, vol. 27, no. 5. ACM, 2008, p. 165.

[22] Z. Pan, H. Bao, and J. Huang, "Subspace dynamic simulation using rotation-strain coordinates," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 6, p. 242, 2015.

[23] A. Mahoney, J. Bross, and D. Johnson, "Deformable robot motion planning in a reduced-dimension configuration space," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 5133–5138.

[24] R. Alterovitz and K. Goldberg, *Motion Planning in Medicine: Optimization and Simulation Algorithms for Image-Guided Procedures*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[25] E. Yoshida, K. Ayusawa, I. G. Ramirez-Alpizar, K. Harada, C. Duriez, and A. Kheddar, "Simulation-based optimal motion planning for deformable object," in *2015 IEEE International Workshop on Advanced Robotics and its Social Impacts (ARSO)*, June 2015, pp. 1–6.

[26] B. Frank, C. Stachniss, N. Abdo, and W. Burgard, "Efficient motion planning for manipulation robots in environments with deformable objects," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2011, pp. 2180–2185.

[27] Z. Pan and D. Manocha, "Feedback motion planning for liquid pouring using supervised learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, 2017, pp. 1252–1259.

[28] M. Saha and P. Isto, "Manipulation planning for deformable linear objects," *Trans. Rob.*, vol. 23, no. 6, pp. 1141–1150, Dec. 2007.

[29] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, p. 43, 2012.

[30] I. Mordatch, K. Lowrey, G. Andrew, Z. Popovic, and E. V. Todorov, "Interactive control of diverse complex characters with neural networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 3132–3140.

[31] 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.

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

[33] N. Ratliff, M. Zucker, J. A. D. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, Pittsburgh, PA, May 2009.

[34] Z. Pan and D. Manocha, "Active animations of reduced deformable models with environment interactions," *arXiv preprint arXiv:1708.08188*, 2017.

[35] A. X. Lee, H. Lu, A. Gupta, S. Levine, and P. Abbeel, "Learning force-based manipulation of deformable objects from multiple demonstrations," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 177–184, 2015.

[36] A. X. Lee, A. Gupta, H. Lu, S. Levine, and P. Abbeel, "Learning from multiple demonstrations using trajectory-aware non-rigid registration with applications to deformable object manipulation," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2015, pp. 5265–5272.

[37] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[38] J. A. Lee and M. Verleysen, *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.

[39] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[40] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudk, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 315–323.

[41] O. Sorkine, "Least-squares rigid motion using svd," *Technical Notes*, vol. 120, p. 3, 2009.

[42] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, "Learning modular neural network policies for multi-task and multi-robot transfer," *CoRR*, vol. abs/1609.07088, 2016.

[43] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016, pp. 1329–1338.

[44] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," *CoRR*, vol. abs/1610.00696, 2016.

[45] A. Tamar, Y. WU, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 2154–2162.

[46] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 4397–4404.