Reproducing Expert-Like Motion in Deformable Environments Using Active Learning and IOC

Calder Phillips-Grafflin and Dmitry Berenson

Abstract We propose a method that allows a motion planning algorithm to imitate the behavior of expert users in deformable environments. For instance, a surgeon inserting a probe knows intuitively which organs are more sensitive than others, but may not be able to mathematically represent a cost function that governs his or her motion. We hypothesize that the relative *sensitivities* of deformable objects are encoded in the expert's demonstrated motion and present a framework which is able to imitate an expert's behavior by learning a sensitivity-based cost function under which the expert's motion is optimal. Our framework consists of three stages: (1) Automatically generating demonstration tasks that prompt the user to provide informative demonstrations through an active learning process; (2) Recovering object sensitivity values using an Inverse Optimal Control technique; and (3) Reproducing the demonstrated behavior using an optimal motion planner. We have tested our framework with a set of 5DoF simulated and 3DoF physical test environments, and demonstrate that it recovers object parameters suitable for planning paths that imitate the behavior of expert demonstrations. Additionally, we show that our method is able to generalize to new tasks; e.g. when a new obstacle is introduced into the environment.

1 Introduction

Manipulation of deformable objects, and in deformable environments, is an important area of research, as deformable objects are common in domestic, industrial, and medical environments. Unlike manipulation in rigid environments, where collisions are forbidden, deformable environments allow, and often require, collisions between a robot and deformable objects. However, modeling deformable objects is a difficult problem; models must not only capture the geometry (undeformed and deformed) of objects (itself a very difficult problem), but should also capture the sensitivity of the object. This qualitative aspect is critical for deformable environments, as it allows a motion planner to distinguish between multiple objects with similar physical properties but with different qualitative characteristics. An important example of this occurs in surgical robotics; while multiple organs and tissues may have sim-

Worcester Polytechnic Institute, 100 Institute Rd, Worcester, MA 01609 USA
e-mail: cnphillipsgraffl@wpi.edu, dberenson@cs.wpi.edu. +This work is supported in part by the Office of Naval Research under Grant N00014-13-1-0735 and by the National Science Foundation under Grant IIS-1317462.

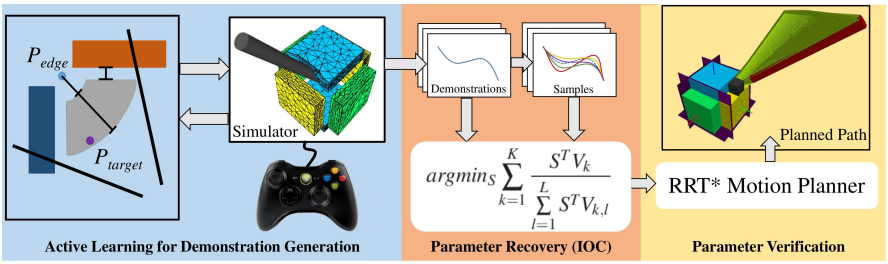


Fig. 1: Diagram of the three stages and main components of our framework.

ilar physical properties, some parts of the body are significantly more sensitive than others. Without accounting for sensitivity, motion planners can produce paths that could cause unnecessary injury.

The motion planning methods introduced in our previous work [16] use a voxel-based representation of deformable objects, in which each voxel has two parameters. The first parameter, *deformability*, captures physical properties of the rigidity of the material. The second parameter, *sensitivity*, captures the qualitative significance of deforming the object. Together, these parameters are used in a cost function that provides a cost of deformation that can be used in cost-aware motion planners.

While the deformability parameters are directly related to material properties, setting the sensitivity parameters is more difficult, as they capture a range of object characteristics. Setting them by hand is time-consuming and error-prone, as incorrect sensitivity values can produce unwanted planner behavior. More problematically, setting these parameters for practical environments requires both domain knowledge and the ability to mathematically represent that knowledge such that the planner will perform well. Instead, we propose a framework for automatically learning and validating these parameters from expert demonstrations. For example, a surgeon can demonstrate the optimal path for inserting a probe, and we can use this demonstration to find the sensitivity values of organs around the path.

Our framework consists of three parts: (1) Automatic generation of demonstration tasks that prompt the user to provide informative demonstrations through a novel active learning process; (2) Recovery of object sensitivity values using *Path Integral Inverse Reinforcement Learning* (PIIRL) Inverse Optimal Control (IOC) technique [10]; and (3) Reproduction of the demonstrated behavior using the RRT* asymptotically-optimal motion planner [12] with a key modification that allows us to check for punctures of deformable objects.

This approach offers two main advantages over existing similar techniques. First, by using sampling-based techniques for IOC that avoid the need to solve the forward problem and sampling-based asymptotically-optimal planners, our framework is applicable to higher-dimensional problems than approaches such as LEARCH [17], which are limited by the need to repeatedly compute optimal paths to recover the cost function. Second, our proposed method for automatically generating demonstration tasks for experts to perform reduces the number of demonstration tasks needed to capture the desired behavior and removes the need for domain knowledge

to generate these tasks by hand. Finally, to our knowledge, IOC has never before been applied to the problem of learning deformable object parameters.

In our experiments in simulated and physical test environments we show that, despite the limitations inherent in asymptotically-optimal sampling-based planning, the recovered sensitivity parameters allow motion planners to reliably reproduce behavior demonstrated by expert users. We also present experiments which show the generalization capabilities of our method.

2 Related Work

Extensive work on the modeling and representation of deformable objects has been done, primarily from the perspectives of computer graphics [6] and medicine [2]. In recent years, this work has been adopted by the robotics field to enable the manipulation of real-world deformable objects such as clothing, rope, food, and human tissue [8]. A wide range of simulation-based models for deformable objects are available, most of which are meshed models based on Mass-Spring (M-S) [6] and Finite-Element (FEM) [14, 7], but mesh-less models [13, 5, 16] have also been proposed. Our model replaces the need for physical simulation with a cost function based on the volume of intersection between voxelized deformable and rigid objects [16]. While this approach cannot capture moving objects, and can only approximate the true deformation, it is extremely efficient to compute in comparison to simulation-based methods, and thus ideal for use inside a motion planner. Notably, because our approach provides a cost function that accounts for both object deformation and sensitivity, it produces plans that minimize deformation and preferentially deform or avoid objects based on their sensitivity.

Inverse Optimal Control (IOC) is the problem of recovering the cost or reward function being optimized by a trajectory or policy. Introduced by Kalman [11] and applied to robotics by Ng et al. [15], several different formulations of the IOC problem and algorithms to address it have been proposed, covering both continuous and discrete state spaces [15]. Earlier approaches to the IOC problem, such as apprenticeship learning, require that the forward problem be solved in addition to computing optimal weights [1, 17]. More recent approaches, based on the maximum entropy principle, replace the need for solving the forward problem by using sample trajectories around the demonstration [19].

The IOC approach we use, *Path Integral Inverse Reinforcement Learning* (PIIRL) samples around the demonstration instead of solving the forward problem [10]. In the PIIRL formulation, a series of locally-optimal demonstration trajectories are gathered from the user(s). For each of these demonstrations, a set of sample trajectories around the demonstration is generated; note that these samples are assumed to be sub-optimal relative to their demonstration. For all demonstrations and all samples, user-specified features are evaluated, and the weights associated with these features are then recovered via a convex optimization problem that attempts to max-

imize the margin between the features of the demonstrations and the features of the samples.

3 Problem Statement

Let τ represent the path of a rigid object (i.e. the robot) through an environment composed of n deformable objects $E = O_1, O_2, ..., O_n$. Representing τ with a discrete sequence of configurations, we assume the cost of executing τ is a function of the form $C(\tau) = \sum_{k=1}^{|\tau|} \sum_{i=1}^{n} D_i S_i V_i(\tau_k)$, where $V_i(\tau_k)$ is the volume of deformation of O_i that results from placing the rigid object at the kth configuration of path τ, D_i is the deformability of O_i, and S_i is the sensitivity of O_i. We focus on learning the S_i parameters, so we assume $D_i = 1 \; \forall i$, though our methods work with any known D. Note that while sensitivity parameters S can be set per-voxel in our representation, we simplify the problem of recovering sensitivities by assuming that each object has uniform sensitivity.

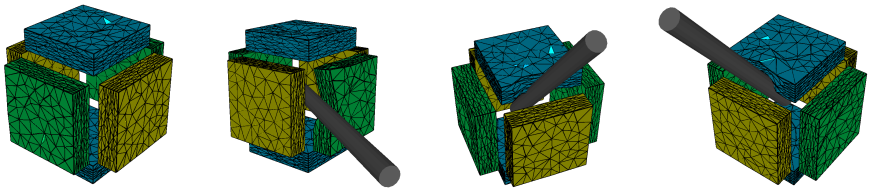
S represents the ground-truth sensitivities of the objects. We seek to generate a set of learned parameters $\hat{S}$ from a set of demonstrations, such that these $\hat{S}$ can be used in a motion planner to produce similar behavior to the demonstrations. Obtaining the true S from demonstration is not possible in general, as a demonstration can, at best, encode only the ratios between different elements of S and not their magnitudes. Thus it is not meaningful to compare S to $\hat{S}$ directly. A more informative comparison is in how well a planner imitates demonstrated behavior when planning with $\hat{S}$. Thus we evaluate our method in terms of the cost of the path produced by our framework. Therefore the quality of $\hat{S}$ relative to the ground truth is evaluated as $E(\hat{S}, S) = C_S(\tau_d) - C_S(\tau_{planned}(\hat{S}))$, where τ_d is a path demonstrated for a given task, $\tau_{planned}(\hat{S})$ is a path planned for the same task using the sensitivities $\hat{S}$, and the cost function $C_S(\cdot)$ is evaluated using the ground-truth sensitivities S.

4 Methods

We have developed a framework for recovering sensitivity parameters for deformable objects, as illustrated in Figure 1. Below we describe each of the four components in detail.

4.1 Capturing Demonstrations

Like all IOC problems, our approach requires demonstrations. In our case, demonstrations are captured in a simulation environment using a physics simulator to simulate deformable objects. Our demonstration task consists of inserting a cylindrical probe between deformable objects to reach target points distributed across the environment, as illustrated in Figure 2. The user attempts to minimize contact with more sensitive objects (shown in yellow and green) compared to less sensitive objects (shown in blue). We record the demonstration trajectory along with the features



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

Fig. 2: Example demonstration tasks for our 6-object test environment, shown with the probe reaching the target. (a) Low sensitivity objects L_1, L_2 (blue), medium sensitivity objects M_1, M_2 (green), and high sensitivity objects H_1, H_2 (yellow). (b,c,d) Goal configurations for three automatically generated tasks.

of that trajectory, which are the total amounts of deformation of each object. While outwardly simple, the problem of probe and needle insertion between deformable objects such as this is common in medical tasks [2] and a subject of previous research in robot motion [2, 13], however, none of this work has explored learning qualitative properties of deformable objects to determine higher-level behavior. In addition to capturing demonstrations, we use this simulation environment to compute feature vectors for demonstration and sample paths.

Each demonstration we capture can be parametrized as a *demonstration task* by a starting pose of the probe P_{start}, a target point P_{target} the user must touch with the probe tip, and a set of "collision planes" $Cplanes$, hyperplanes that constrain the motion of the probe. As shown in Figure 3, the hyperplanes approximate a funnel that guides the user towards the target point and restricts which objects the user can contact with the probe. These hyperplanes are added to constrain the user to producing demonstrations that capture the relative difference in sensitivity between the accessible objects. In our experience, without the hyperplanes users sometimes produce demonstrations that deform only the globally least-sensitive object(s) instead of capturing sensitivity relationships between neighboring objects.

While we attempt to capture optimal demonstrations, in practice users may provide slightly sub-optimal demonstrations. We attempt to correct for this using a local optimizer that optimizes each demonstration. This method generates a set of random sample trajectories around the demonstration trajectory and replaces the demonstration trajectory with any of the random samples with strictly dominating deformation (i.e. the random sample deforms all objects less than or equal to the demonstration).

4.2 Active Learning

We can capture demonstrations and compute features for demonstrations and samples needed for PIIRL; however, this leaves two problems to address: how to generate demonstration tasks for the user to complete, and how many demonstrations must be collected. Clearly, the accuracy of recovered sensitivity values depends on the quality of the demonstrations provided. For example, if an object has zero fea-

Algorithm 1 Demonstration task collection algorithm

procedure COLLECTDEMONSTRATIONS(A)
  $G \leftarrow \{\emptyset, \emptyset\}$
  $O_1 \leftarrow argmax_{o \in E}$ degree$(A(o))$
  $O_2 \leftarrow argmax_{o \in \text{neighbors}(A(O_1))}$degree $(A(o))$
  $G \leftarrow G \cup$ COLLECTSINGLEDEMONSTRATION(O_1, O_2)
  while $\{o \in E | o \notin G_v, \text{degree}(A(o)) > 0\} \neq \emptyset$ **do**
    $O_1 \leftarrow argmax_{\{o \in G_v | \text{neighbors}(A(o)) \setminus G_v \neq \emptyset\}}$ depth$(G(o))$
    $O_2 \leftarrow argmax_{\{o \in \text{neighbors}(A(O_1)) \setminus G_v\}}$ degree$(A(o))$
    $G \leftarrow G \cup$ COLLECTSINGLEDEMONSTRATION(O_1, O_2)
    $G \leftarrow$ ENSURERANKING(G)
  return G
procedure ENSURERANKING(G)
  for $O_1 \in G_v$ **do**
    for $\{O_2 \in G_v |$ depth$(G(O_2)) \geq$ depth$(G(O_1))\}$ **do**
      if NODIRECTEDPATHEXISTS(O_1, O_2) **then**
        if DIRECTLYCOMPARABLE(O_1, O_2) **then**
          $G \leftarrow G \cup$ COLLECTSINGLEDEMONSTRATION(O_1, O_2)
          return ENSURERANKING(G)
  return G
procedure COLLECTSINGLEDEMONSTRATION(O_1, O_2)
  $P_{target}, P_{edge}, C_{planes} \leftarrow$ GENERATEDEMONSTRATIONTASK$(O_1, O_2, T_{clearance}, T_{range})$
  $D_v, D_e \leftarrow$ GETDEMONSTRATIONFROMUSER$(P_{target}, P_{edge}, C_{planes})$
  return (D_v, D_e)

ture values in both demonstrations and the trajectory samples around the demonstrations used by PIIRL, we cannot recover a meaningful sensitivity value for the object; e.g. if all demonstrations entered through the forward half of our cube environment, no features would be available for objects on the reverse. A different, but equally problematic, issue occurs when features have been collected for every object, but the demonstrations are "unconnected"; for example, in an environment $E = \{O_1, O_2, O_3, O_4\}$, if features have been collected for demonstrations between O_1, O_2 and O_3, O_4, but not for O_2, O_3, the optimizer cannot determine if O_1 and O_2 are more or less sensitive that O_3 and O_4. Thus, we need to ensure that sufficient demonstrations have been collected.

The conservative solution is to require a demonstration for every pair of adjacent objects, however, this can result in a large number of demonstrations. For our test environment shown in Figure 2, 12 demonstrations would be required to capture the relationship between every adjacent pair. We seek to reduce the number of demonstrations required.

Simply collecting demonstrations such that we observe a non-zero feature for each object is insufficient for accurate parameter recovery, rather, we must ensure that the demonstrations collected form a *ranking* of the objects in terms of sensitivity; i.e. that for objects $O_1, O_2 \in E$, $rank(O_1)$ is either less than, equal to, or greater than $rank(O_2)$ if the objects are comparable. Rankings are derived from demonstrations collected between adjacent objects; the preferentially-deformed object receives a lower ranking than the preferentially-avoided object. Rankings are

Algorithm 2 Demonstration task generation algorithm

procedure GENERATEDEMONSTRATIONTASK($O_1, O_2, T_{clearance}, T_{range}$)
  $P_{edge} \leftarrow$ GETEDGEPOINTBETWEENOBJECTS(O_1, O_2)
  $P_{target} \leftarrow \emptyset$
  while $P_{target} = \emptyset$ **do**
    $P_{sampled} \leftarrow$ SAMPLEINRANGE(P_{edge}, T_{range})
    if $clearance(P_{sampled}) < T_{clearance}$ **then**
      $P_{target} \leftarrow P_{sampled}$
  $Cplanes \leftarrow$ GENERATECOLLISIONPLANES(P_{edge}, P_{target})
  return $P_{target}, P_{edge}, Cplanes$

not comparable in certain cases, such as between objects on the opposing faces of our test environment, in which it impossible to perform a demonstration between the two objects, and they cannot be ranked via a combination of other demonstrations.

Demonstrations are collected using Algorithm 1, which takes A, the set of object adjacencies in E, and iteratively collects demonstrations until there are no more useful demonstrations to perform. This algorithm captures preference relationships between objects by building a directed graph G. The nodes in G represent objects in the environment and the directed edges point from the less-sensitive object to the more-sensitive one. Initially, G contains no nodes or edges, and each demonstration adds an edge and 0, 1, or 2 nodes. The key to the algorithm is determining which demonstration (and thus which edge) should be queried next.

The algorithm uses the structure of G at the current time as well as a heuristic to decide which demonstration to query next. If the ranking between all objects in G is known, then the algorithm selects a new object to add to G (via a demonstration involving that object and one already in G). After adding a new object, the algorithm queries demonstrations until the ranking of all objects in the graph is again established (this is done in the ENSURERANKING function). It then selects a new object to add, and so on, until no more objects can be added.

At each step where objects or edges are selected, we choose the object or edge based on connectivity heuristics. For new objects (i.e. those not already in G), we prefer those that are adjacent to as many other objects as possible. When picking objects already in G for a new edge, we prefer objects that have a higher "depth". Here depth(n) is the length of the longest directed path in G which ends at n. These heuristics bias the algorithm to create long chains of edges where possible, which is clearly beneficial for forming a ranked list; e.g. $rank(O_1) < rank(O_2) < rank(O_3) < rank(O_4)$ is a chain of three edges which gives a complete ranking of four objects.

Algorithm 1 is not guaranteed to produce the minimal set of demonstrations because it cannot foresee the results of future demonstrations. It frequently collects demonstrations early on that prove to be unnecessary in the final set of demonstrations. In pathological environments, Algorithm 1 may be forced to collect all possible demonstrations. However, in practice, we show that it reduces the number of demonstrations without significant impact on the recovered sensitivity parameters.

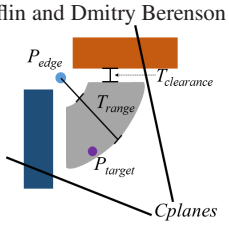


Fig. 3: Our automatic demonstration task generator.

For each demonstration requested by Algorithm 1, we generate a new task using Algorithm 2. This algorithm is given a pair of target objects O_1, O_2, a target clearance $T_{clearance}$, and a target depth range T_{range}. First, the algorithm selects an "edge point", P_{edge} by randomly selecting a point on the medial axis between the two target objects. Using the edge point, the algorithm randomly samples nearby points T_{range} away from the edge point to select one that is "inside" [1] the environment and also at least $T_{clearance}$ away from an object, which it returns as P_{target}, the target point. Finally, a set of "collision planes" are generated to restrict the user's demonstration to the desired area. The parameter T_{range} ensures that the user must insert the probe sufficiently to cause deformations. Similarly, the parameter $T_{clearance}$ controls how close to an object the target point can be, and can be used to ensure that the target point itself is not in contact with an object (see Figure 3).

4.3 Parameter Recovery

Our approach to motion planning for deformable objects, introduced in [16], uses a "cost of deformation" to enable any motion planner that accounts for cost to produce plans that minimize deformation. We can frame the problem of imitating demonstration behavior as the problem of inferring the sensitivity parameters used to produce the demonstration. Assuming that the demonstration is optimal, this is the well-established problem of Inverse Optimal Control (IOC).

Using the PIIRL formulation of IOC, the cost function consists of a series of features $V = V_1, V_2, ..., V_n$ (in our case these are the amounts of deformation of each of the n objects) with corresponding sensitivities $S = S_1, S_2, ..., S_n$, such that the total cost of a configuration $C = \sum_{i=1}^{n} V_i S_i$, where the V_i can be computed using our physics simulator, but the optimal set of sensitivities S^* is unknown.

To find the best estimate of the optimal set of sensitivities $\hat{S}$, PIIRL requires a set of sample paths around each demonstration. Because the demonstrations are assumed to be locally optimal, all samples around a demonstration will be sub-optimal w.r.t to the unknown cost function. For K demonstrations and L samples for each demonstration, the optimal weights are obtained using the following minimization problem (a similar form of the minimization problem used in [10]), where V_k are the feature values for demonstration k, and $V_{k,l}$ are the feature values for sample l of

[1] To determine which points are "inside" the environment, we compute a "local maxima map" using the Signed Distance Field (SDF) of the environment. For each point in the SDF, we follow the gradient away from obstacles and record the location the gradient becomes zero (i.e. the local distance maxima). Points "inside" the environment have corresponding local maxima inside the bounds of the SDF, while points "outside" have local maxima corresponding to the bounds of the SDF. Intuitively, "inside" points have finite-distance local maxima reachable via the gradient, while for "outside" points, the local maxima are undefined.

demonstration k:

$$\hat{S} = argmin_S \sum_{k=1}^{K} \frac{S^T V_k}{\sum_{l=1}^{L} S^T V_{k,l}} \tag{1}$$

This minimization finds the sensitivity values $\hat{S}$ that maximize the margin between the cost of the demonstrations and the costs of their samples. Note that in our problem, $S > 0$ and sample feature values $V_{k,l} \neq 0$, as all sensitivity values must be greater than zero and $V_{k,l} = 0$ implies $V_k = 0$ (since samples must be sub-optimal relative to their demonstrations). $V_k = 0$ implies that the demonstration k captures no information about any object and thus can be removed from the optimization so this condition will not occur. Since this minimization problem is convex, we can use standard convex optimization solvers to find optimal weights. Unlike previous work such as LEARCH [17], PIIRL does not rely on the specific configurations the demonstration path traverses; rather, only the corresponding feature values must be locally optimal in our cost function [9]. This makes it tractable to learn cost functions in high-dimensional spaces.

4.4 Recovered Parameter Verification

Once sensitivity values $\hat{S}$ have been recovered for each object in our test environments, we must verify that the recovered values allow our motion planner to imitate the behavior of the expert demonstrations. We attempt to perform each demonstration task using an optimal motion planner and comparing the planned path $\tau_{planned}(\hat{S})$ with the demonstration τ_d in terms of the true cost function $C_S(\cdot)$ using the ground truth sensitivity values S. In our previous work, we used the T-RRT and GradienT-RRT planners to efficiently produce paths in high-dimensional spaces [16]; however, since these planners have no optimality guarantees, they are unsuitable for parameter verification. Instead, we use the asymptotically-optimal RRT* planner [12] with our deformation cost function. While we could use deformations measured via a physics simulator to compute cost during planning, our voxel-based deformation cost function is significantly faster, more stable, and detects object punctures and separation. To accurately mimic the demonstration tasks, the RRT* planner is provided with the same task-space target point to reach with the probe tip, rather than a goal configuration of the probe. Feasible configurations touching this target point can be sampled, and RRT* attempts to connect the tree to these goal states. As RRT* runs, it improves the path by reducing the deformation cost of the path and by sampling and connecting to new, lower-cost goal states. Note that while RRT* is *asymptotically* optimal, for finite time it will not return *the optimal* path, so we expect paths reproduced with RRT* may be slightly higher-cost than their corresponding expert demonstrations, but should exhibit the same preferential deformation demonstrated by the expert.

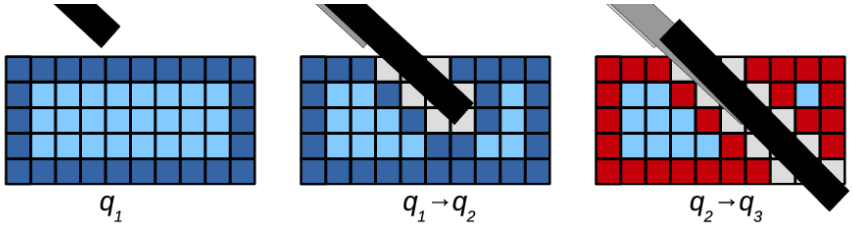


Fig. 4: Illustration of puncture checking for an extension from configuration q_1 to q_3. As the surfaces are no longer connected (red), puncture has occurred and the $q_2 \rightarrow q_3$ motion is invalid.

In addition to integrating our existing cost function with RRT*, we have significantly improved the quality of planned paths by adding puncture detection to prevent paths from puncturing or cutting deformable objects. Puncture and cut detection is essential to planner performance; without it, planners can produce low-cost paths that pass directly though deformable objects. To prevent punctures and cuts, we check every extension of the tree in RRT* for puncture using an incremental variant of the algorithm introduced by Chen et. al. [3] for computing topological invariants on voxel grids. The original algorithm extracts the surface vertices from the voxel grid, and computes the connectivity of each surface vertex. Each surface vertex can be connected to between one and six neighboring surface vertices; let $M1$ be the total number of surface vertices with one connected neighbor, $M2$ the total with two neighbors, and so on. From these totals, Chen et. al. prove that the number of holes in the voxel grid is $n_{holes} = 1 + ((M5 + (2 * M6) - M3)/8)$.

Thus, checking for punctures can be implemented by removing the swept volume of the path of the probe from the voxel-based model of deformable objects used for motion planning, and then computing the number of holes to ensure that no new holes have been created by the path. Additionally, to prevent objects from being completely cut apart by the path, the overall connectivity of the surface voxels corresponding to each object are computed; if the surface vertices for an object form multiple disconnected groups, then the object has been cut apart by the path.

To efficiently perform these checks during the planning process, we incrementally check for punctures with each extension and rewiring step of RRT* (see Figure 4). For testing a new edge from configuration q_1 to configuration q_2 the process is as follows: (1) retrieve the stored object surfaces corresponding to q_1, (2) update the object surfaces with the swept volume from q_1 to q_2, (3) compute the number of holes in each object surface (check for puncture), (4) compute the connectivity of each object surface (check for cuts), and (5) if no holes or cuts are encountered, store the updated surfaces corresponding to q_2. For every such check, we are effectively checking the entire path from the start configuration q_{start} to q_2 for punctures and cuts.

5 Results

We present results of testing our framework in a 3D simulated environment (5DoF probe insertion task) and in a physical planar environment (3DoF rigid object navigation task) using an industrial robot. We use the Bullet physics simulator [4] to provide an environment for capturing demonstrations and computing features, and the Open Motion Planning Library (OMPL) [18] to provide the RRT* planner used to verify the recovered sensitivity values. We show that our methods accurately recover sensitivity values that allow planners to imitate expert demonstrations. We also report on how the algorithm generalizes to a new task, where an obstacle is introduced into the environment, and report on the use of active learning for reducing the number of demonstrations required. Ideally, we would compare the performance of our framework with existing approaches such as LEARCH [17], however, these approaches require computing the true optimal path to perform IOC, which is intractable in the 5DoF probe insertion task.

5.1 Recovered Behavior

We first demonstrate the performance of our framework in the 3D simulated environment without using the automatic demonstration task generator, and show that our demonstration capture environment and parameter recovery process produce acceptable object sensitivity values. Using our RRT* planner, we show that the recovered sensitivities produce paths that imitate the expert demonstrations.

The test environment, as shown in Figure 2, consists of six deformable objects forming the faces of a hollow cube. These objects form three classes; each pair of opposing faces has the same sensitivity assigned, with the lowest sensitivity (L_1, L_2) shown in blue, an intermediate sensitivity (M_1, M_2) shown in green, and high sensitivity (H_1, H_2) shown in yellow. For testing purposes, the "true" sensitivity values of these objects are set as $L_1, L_2 = 0.2$, $M_1, M_2 = 0.4$, $H_1, H_2 = 0.8$. We use the true values to evaluate the quality of paths planned with the recovered sensitivity values, but they are unknown to our IOC method.

Using the conservative approach discussed in Section 4.2, 12 demonstrations were performed, one for each pair of adjacent objects. Several examples of these demonstrations can be seen in Figure 2 and Figure 5. While time-consuming, this approach ensures that sufficient demonstrations have been collected to capture the desired behavior. In these demonstrations, lower-sensitivity objects were preferentially deformed instead of higher-sensitivity objects.

Using the set of 12 demonstrations, we recovered the object sensitivity parameters using our parameter recovery process. We generated a set of 100 sample paths around each demonstration using a multivariate gaussian distribution using the process described in [9], which produces smooth noisy path samples around an initial path. Features for all demonstrations and samples were computed by executing paths in the demonstration capture environment, and all feature values were normalized

relative to the highest feature value. Using the PIIRL formulation of IOC, the optimal weights were recovered using the convex optimization problem in Equation (1); we used the function minimization tools in MATLAB to perform this optimization. For optimization, the lower bound of possible weight values was 0.1, and the upper bound was 1000, with the weights initialized to 500. The recovered sensitivity values were $L_1 = 0.10004$, $L_2 = 0.10092$, $M_1 = 2.8523$, $M_2 = 8.5683$, $H_1 = 958.92$, $H_2 = 999.51$. Note that both high sensitivity objects (H_1 and H_2) were avoided in all demonstrations, and thus received maximum weights in the optimization. Again, recovery of the true sensitivities is impossible and we must evaluate our method in terms of the cost of the path planned using the recovered sensitivities.

5.1.1 Recovered Parameter Verification

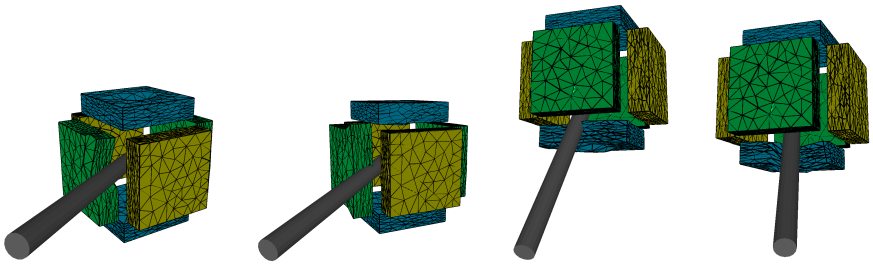


Fig. 5: Examples of goal configurations from demonstrations (a,c) and corresponding goals of paths planned using recovered sensitivity values (b,d). Full paths are not shown for clarity.

Using the object sensitivity parameters recovered using PIIRL, we planned for all 12 demonstration tasks using RRT*. Table 1 compares the demonstrations with results for planning times of 30 and 60 minutes, with 30 and 15 trials of each, respectively. Figure 5 shows examples of demonstrated paths compared with paths produced by RRT*. As shown in the table, paths produced using the recovered parameters imitate the behavior of the demonstrations by deforming the same objects with similar amounts of deformation except for two demonstrations (namely 9 and 12) for which the planner found a path superior to the original demonstration. Note that due to the difficulty of the planning problem and the finite planning time for RRT*, we do not expect planned paths to exactly match the demonstrations. Two notable types of error resulted in sub-optimal plans, namely cases where planned paths clip the edge of higher-sensitivity objects, and cases where planned paths simply result in higher cost than the demonstration. In both cases, errors are indicated by high standard deviations; this is expected if a small number of the planned paths exhibit particularly sub-optimal behavior. These errors are caused by the limited time available to RRT*, which restricts the number of goal states sampled and the

	Demonstrated			Recovered (30 plans, 30 min/plan)			Recovered (15 plans, 60 min/plan)		
	L	M	H	L	M	H	L	M	H
1	5.61	0.0	0.0	12.14 [4.68]	0.0 [0.0]	0.0 [0.0]	10.81 [1.55]	0.0 [0.0]	0.0 [0.0]
2	7.14	0.0	0.0	12.35 [2.7]	0.0 [0.0]	0.0 [0.0]	11.57 [2.82]	0.0 [0.0]	0.0 [0.0]
3	4.87	0.0	0.0	10.65 [3.71]	0.16 [0.82]	0.0 [0.0]	9.82 [2.84]	0.0 [0.0]	0.0 [0.0]
4	5.30	0.0	0.0	10.27 [2.75]	0.07 [0.38]	0.01 [0.03]	11.06 [2.22]	0.0 [0.0]	0.0 [0.0]
5	7.69	0.0	0.0	13.65 [4.18]	0.0 [0.0]	0.1 [0.53]	14.17 [3.62]	0.0 [0.0]	0.0 [0.0]
6	7.92	0.0	0.0	10.73 [2.09]	0.0 [0.0]	0.0 [0.01]	11.24 [4.7]	0.0 [0.0]	0.0 [0.0]
7	7.27	0.0	0.0	11.86 [2.93]	0.1 [0.54]	0.0 [0.0]	12.48 [3.5]	0.0 [0.0]	0.0 [0.0]
8	9.55	0.0	0.0	13.48 [3.52]	0.34 [1.47]	0.0 [0.0]	11.97 [2.97]	0.26 [0.97]	0.0 [0.0]
9	0.0	35.59	0.0	0.02 [0.13]	32.55 [9.13]	0.0 [0.01]	0.03 [0.11]	31.51 [10.48]	0.0 [0.0]
10	0.0	20.38	0.0	0.9 [1.63]	21.8 [8.44]	0.0 [0.01]	1.44 [2.69]	20.51 [8.5]	0.0 [0.0]
11	0.0	18.67	0.0	0.02 [0.08]	23.79 [5.62]	0.29 [1.29]	0.17 [0.56]	24.68 [5.55]	0.0 [0.0]
12	0.0	17.17	0.0	9.58 [1.95]	0.1 [0.32]	0.07 [0.25]	9.36 [1.96]	0.02 [0.09]	0.03 [0.09]

Table 1: Comparison between demonstrated behavior and paths planned using object sensitivity values recovered from 12 demonstrations between each pair of adjacent objects. Costs reported (mean [std.dev.]) are the integral of volume change multiplied by the true object sensitivity values, separated by class of object (L = low-sensitivity, including objects L_1 and L_2, M = medium-sensitivity, including objects M_1 and M_2, H = high-sensitivity, including objects H_1 and H_2).

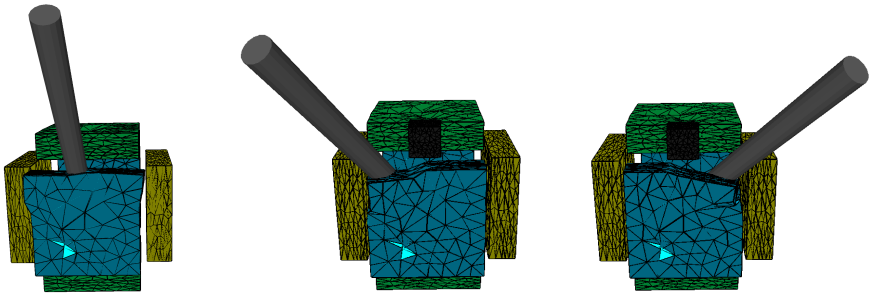
refinement of the path. Results for 60-minute planning times shown in Table 1 show that in most cases, increased planning time reduces these errors. Note that the high planning times used here are partially a consequence of our puncture test, which adds considerable computation in addition to the deformation cost function.

5.1.2 Generalization of Recovered Parameters

The importance of recovering sensitivity parameters is not to reproduce the demonstrations, since these could simply be replayed; rather, recovering the sensitivity parameters allows us to generalize the behavior displayed in the demonstrations to other tasks in the test environment. To demonstrate that the recovered sensitivity parameters generalize, we performed a set of tests shown in Figure 6. Starting from one of the demonstrations (demonstration task 6), we adjusted the target point and inserted rigid obstacles that block the demonstrated path. As shown in Figure 6, our planner produces paths that exhibit the same behavior as the demonstration path; while the new path differs from the demonstration and thus results in different cost, the preferential deformation of the blue object over the green one indicates that the expert's preference was correctly captured.

5.2 Automatic Generation of Demonstration Tasks

Using the same test environment, we tested our active learning method for automatically collecting demonstration tasks. Examples of these demonstration tasks are shown in Figure 2. Unlike the conservative approach discussed previously,



(a) L=7.27,M=0.0,H=0.0 (b) L=8.20, M=0.0, H=0.0 (c) L=21.24, M=0.0, H=0.0

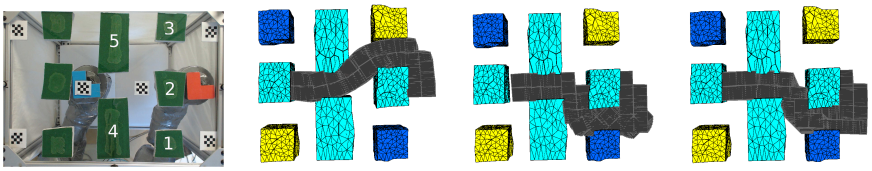
Fig. 6: Paths planned to show the generality of recovered sensitivity values, (a) goal configuration of demonstration 6, and (b)(c) two goals of paths planned with target points offset from the center of the environment when the direct path from start to target is blocked by a rigid obstacle (black).

which used demonstrations between all pairs of adjacent objects, the active learning method generates only enough tasks to form a ranking of all objects in the environment. We tested the active learning method in the same test environment as above and allowed it to select a subset of tests from the set of comprehensive demonstrations. Using this method, between 8 and 10 demonstrations were required to capture features for all objects, compared to the 12 used by the conservative approach. As before, 100 sample paths were generated around each demonstration, and sensitivities were recovered using the PIIRL optimization problem. Since the active learning process involves some random selections, we ran 15 trials; 10 demonstrations were required in 14 cases, and 8 demonstrations in 1 case, with average recovered sensitivities (average [std.dev.]) being $L_1 = 0.100[0.0], L_2 = 0.101[0.0002]$, $M_1 = 2.858[0.012], M_2 = 8.630[0.071], H_1 = 984.12[29.272], H_2 = 999.509[0.165]$. Comparing these results with the sensitivities learned using the full set of demonstrations (see Section 5.1), we observe that the values are not meaningfully different, which shows that the active learning method can infer very similar sensitivity relationships with fewer demonstrations.

5.3 Physical Environment Tests

In addition to testing with our simulated environment, we have also applied our framework to a planar physical test environment shown in Figure 7 with an L-shaped block, similar to those used in our previous work [16]. Like our previous work, the use of a planar 3DoF environment allows for the deformation of objects in the environment to be tracked in real time by an overhead camera. Paths in the environment were planned using the same RRT* planner as before, albeit in $SE(2)$.

For comparison purposes, we first planned using uniform sensitivity values for all objects, as shown in Figure 7b. A demonstration path through a narrower, higher-



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

Fig. 7: Testing for our physical test environment (a), with objects numbered and start (red) and goal (blue) states shown. Swept volumes of (b) path planned with uniform object sensitivity values, (c) demonstration path, and (d) path planned with recovered sensitivity values.

	Object deformation				
	O_1	O_2	O_3	O_4	O_5
Uniform	0	3367	2442	0	554148
Demonstration	23451	0	0	0	35222
Recovered	51569	38798	0	0	73013

Table 2: Deformation comparison for the five left-hand objects in our physical test environment between a path planned with uniform object sensitivity values, the demonstrated path, and a path planned using the recovered sensitivity values. Reported deformation values are in pixels.

deformation passage was provided using our demonstration capture environment, as shown in Figure 7c. As with the simulated environment, 100 samples were generated around the demonstration, and object sensitivity values $O_1 = 1.00$, $O_2 = 200.00$, $O_3 = 100.03$, $O_4 = 200.00$, $O_5 = 36.21$ were recovered using a lower bound of 1, upper bound of 200, and initial value of 100. These parameters are expected, as the demonstration path deforms O_1, O_4 and O_5, while avoiding the other objects. Planning using the recovered values is shown in Figure 7d; planning was performed with a planning time of 5 minutes. Following planning, all three paths were executed in our test environment by an industrial robot, with object deformations tracked by our tracking camera and reported in Table 2. As before, we do not expect the planned path to exactly match the demonstration; in particular due to the narrow low-cost passages in the environment, it is unsurprising that the planned path has significantly higher cost than the expert demonstration. However, the planned path does avoid O_3, instead preferring the passage between O_1 and O_2, which matches the preferences demonstrated by the expert.

6 Conclusion

We have developed a framework for recovering sensitivities of deformable objects so that our motion planners imitate the behavior of expert users in deformable environments. By formulating the problem of motion planning in deformable environments in terms of generating optimal paths that minimize deformation, we can recover object sensitivity parameters from demonstrated optimal paths using IOC. We also propose an active learning algorithm to generate demonstration tasks. Our framework has two advantages over existing similar techniques. First, by using

sampling-based techniques for IOC that avoid the need to solve the forward problem and sampling-based asymptotically-optimal planners, our framework is more applicable to higher-dimensional problems than existing approaches. Second, our method for automatically generating demonstration tasks for users to perform reduces the number of demonstration tasks needed to capture the desired behavior. We tested our framework in simulated and physical test environments, and showed that it recovers object sensitivities suitable for planning paths that imitate the behavior of expert demonstrations. We also showed that these preferences can generalize to new tasks.

References

1. Abbeel, P., Ng, A.Y.: Apprenticeship learning via inverse reinforcement learning. In: ICML (2004)
2. Alterovitz, R., Goldberg, K.: Motion Planning in Medicine: Optimization and Simulation Algorithms for Image-Guided Procedures (Springer Tracts in Advanced Robotics). Springer (2008)
3. Chen, L., Rong, Y.: Linear time recognition algorithms for topological invariants in 3D. In: International Conference on Pattern Recognition (2008)
4. Coumans, E.: Bullet 2.73 Physics SDK Manual (2010)
5. Faure, F., Gilles, B., Bousquet, G., Pai, D.K.: Sparse meshless models of complex deformable solids. ACM Transactions on Graphics pp. 73–73 (2011)
6. Gibson, S.F.F., Mirtich, B.: A survey of deformable modeling in computer graphics. Tech. rep., Mitsubishi Electric Research Laboratories (1997)
7. Irving, G., Teran, J., Fedkiw, R.: Invertible finite elements for robust simulation of large deformation. In: ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '04. New York, New York, USA (2004)
8. Jiménez, P.: Survey on model-based manipulation planning of deformable objects. Robotics and Computer-Integrated Manufacturing **28**(2), 154–163 (2012)
9. Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., Schaal, S.: STOMP: Stochastic Trajectory Optimization for Motion Planning. In: ICRA (2011)
10. Kalakrishnan, M., Pastor, P., Righetti, L., Schaal, S.: Learning Objective Functions for Manipulation. In: ICRA (2013)
11. Kalman, R.: When is a linear control system optimal? Journal of Basic Engineering (1964)
12. Karaman, S., Frazzoli, E.: Sampling-based Algorithms for Optimal Motion Planning. The International Journal of Robotics Research **30**(7), 20 (2010)
13. Maris, B., Botturi, D., Fiorini, P.: Trajectory planning with task constraints in densely filled environments. In: IROS (2010)
14. Müller, M., Dorsey, J., McMillan, L., Jagnow, R., Cutler, B.: Stable real-time deformations. In: ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '02. New York, New York, USA (2002)
15. Ng, A.Y., Russell, S.J., et al.: Algorithms for inverse reinforcement learning. In: ICML (2000)
16. Phillips-Grafflin, C., Berenson, D.: A Representation Of Deformable Objects For Motion Planning With No Physical Simulation. In: ICRA (2014)
17. Ratliff, N., Silver, D., Bagnell, J.A.D.: Learning to search: functional gradient techniques for imitation learning. Autonomous Robots (2009)
18. Şucan, I.A., Moll, M., Kavraki, L.E.: The Open Motion Planning Library. IEEE Robotics & Automation Magazine **19**(4), 72–82 (2012). http://ompl.kavrakilab.org
19. Ziebart, B.D., et al.: Maximum Entropy Inverse Reinforcement Learning. In: AAAI, pp. 1433–1438 (2008)