arXiv:1802.09661v3 [cs.RO] 8 Dec 2018

Cloth Manipulation Using Random-Forest-Based Imitation Learning

Biao Jia, Zherong Pan, Zhe Hu, Jia Pan, Dinesh Manocha
http://cs.unc.edu/~biao/robustm

***Abstract*—We present a novel approach for manipulating high-DOF deformable objects such as cloth. Our approach uses a random-forest-based controller that maps the observed visual features of the cloth to an optimal control action of the manipulator. The topological structure of this random-forest is determined automatically based on the training data, which consists of visual features and control signals. The training data is constructed online using an imitation learning algorithm. We have evaluated our approach on different cloth manipulation benchmarks such as flattening, folding, and twisting. In all these tasks, we have observed convergent behavior for the random-forest. On convergence, the random-forest-based controller exhibits superior robustness to observation noise compared with other techniques such as convolutional neural networks and nearest neighbor searches.**

I. Introduction

High-DOF deformable object manipulation, such as cloth manipulation, is an important and challenging problem in robotics and related areas. It has many applications, including assisted human dressing [1], cloth folding [2], sewing [3], etc. Compared with rigid bodies or three-dimensional volumetric deformable objects [4], cloth can undergo large deformations and form wrinkles or folds, which greatly increases the complexity of cloth manipulation tasks. The possibility of such large deformations is the major challenge in designing a cloth manipulation controller. In a real-life cloth manipulation task, a typical robot only observes a single RGB(D) image of the cloth. As a result, we need robust methods that can perform such complex manipulation tasks based on a single view observation. This involves inferring the 3D configuration of the cloth from the image-based representation and compute the appropriate control action. For example, if a robot manipulates a piece of cloth by holding two corners of the cloth mesh, then the controller should infer the desired end-effector positions of the robot.

Several machine learning models have been proposed to parameterize such controllers, some of which have been used for cloth manipulations. Because of the recent development of deep (reinforcement) learning, one prominent method [5] is to represent feature extraction and controller parametrization as two neural networks, which are trained either jointly or separately. Other works, such as [6], use one unified neural network architecture, but the structures of these neural networks are determined via trial and error. Recently, [7] represented the controller as a set of observations/control-signal pairs constructed manually. However, due to observation noise at runtime, it is not clear whether this constructed set can cover the experienced cases.

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

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

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

Zhe Hu and Jia Pan are with the Department of Mechanical Biomedical Engineering, City University of Hong Kong. E-mail: {zhehu5-c, jiapan}@cityu.edu.hk.

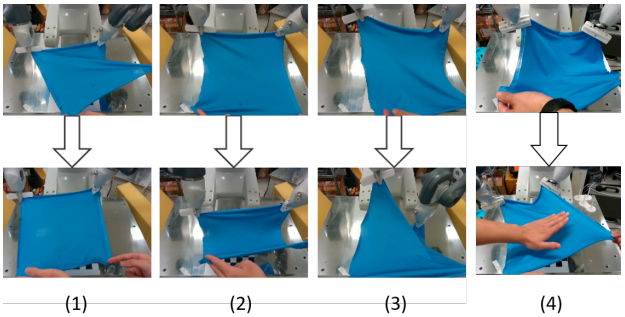


Fig. 1: Manipulation Benchmarks: We highlight the realtime performance of our algorithm on three basic robot-human collaboration tasks. (1): keep the cloth straight; (2): keep the cloth bent; (3): keep the cloth twisted. (4): add noise to the human actions and the visual RGB-D outputs and evaluated the robustness of our approach. (5): evaluate the performance on complex tasks that simultaneously perform straightening, bending, and twist operations to highlight the benefits of our approach.

Main Result: In this paper, we present a new method for cloth manipulation. Our method represents the controller as a random-forest. The random-forest takes the observation of the cloth configuration, an RGB(-D) image, as input. It then classifies the input by bringing it to a leaf-node of each decision tree. The optimal control signals are stored on the leaf-node and used as controller outputs. The random-forest is trained iteratively using imitation learning by collecting a dataset online. In each iteration, more data are collected and the random-forest is retrained to be more robust to observation noises.

Compared with other parametric models such as neural networks, random-forest is non-parametric and the number of leaf-nodes can be dynamically adjusted. As a result, arbitrarily complex cloth configurations can be represented as more training data are provided. Compared with other non-parametric methods such as nearest neighbor, random-forest exhibits better robustness in terms of avoiding over-fitting. We show that as more iterations of imitation learning are performed, the number of leaf-nodes in a random-forest will converge.

We compare the performance of different controller models on three cloth manipulation tasks involving large deformations: cloth flattening, cloth folding, and cloth twisting. The results show that our model always outperforms nearest neighbor [7] and neural networks in terms of matching optimal control signals and robustness to noise. In addition, the number of

leaf-nodes converges as imitation learning progresses.

The rest of the paper is organized as follows. Section II reviews related works. In Section III, we introduce the notation and formulate the problem. In Section IV, we provide details for training the random-forest-based controller. Finally, we highlight the performance on challenging benchmarks in Section V and compare the performance with prior methods.

II. Related Work

In this section, we give a brief summary of prior works on large deformation and manipulation, dimension reduction, and controller optimization.

Large Deformation and Manipulation: Different techniques have been proposed for motion planning for deformable objects. Most of these works (e.g., [4, 8, 9]) focus on volumetric objects such as a deforming ball or linear deformable objects such as steerable needles. By comparison, cloth-like thin-shell objects tend to exhibit more complex deformations, forming wrinkles and folds. Current solutions for thin-shelled manipulation problems are limited to specific tasks, including folding [2, 10, 11], ironing [12], sewing [3], and dressing [1]. On the other hand, deformable body tracking solves a simpler problem, namely inferring the 3D configuration of a deformable object from sensing inputs. There is literature on deformable body tracking, which infers the 3D configuration from sensor data [13, 14, 15]. However, these methods usually require a template mesh as a priori and are mainly limited to handling small deformations.

Dimension Reduction: Previous DOM methods use various feature extraction and dimensionality reduction techniques, including SIFT-features [12], HOW-features [7], and depth-based features [16, 17, 18]. Recently, deep neural networks have also been used as general-purpose feature extractors. They have also been used to manipulate low-DOF articulated bodies [5] and in DOM applications [19, 20]. For simplicity, our random-forest uses HOW-features as inputs. Another feature recently proposed in [21] represents cloth using a small set of feature points. However, these feature points can only characterize small-scale deformations because there can be a lot of occlusions under large deformations.

Controller Optimization: In robotics, reinforcement learning [22], imitation learning [23], and direct trajectory optimization [24] have been used to compute optimal control actions. Trajectory optimization, or a model-based controller, has been used in [2, 12, 25] for DOM applications. Although the resulting algorithms tend to be accurate, these methods cannot be used for realtime applications. For low-DOF robots such as articulated bodies [26], researchers have developed realtime trajectory optimization approaches, but it is difficult to extend them to deformable models due to the high simulation complexity of such models. Currently, realtime performance can only be achieved through learning-based controllers [16, 17, 7, 19], which use supervised learning to train realtime controllers. However, as pointed out in [27], these methods are not robust in handling unseen data. Therefore, we further improve the robustness by using imitation learning. Apart from imitation learning used in this work, realtime cloth manipulation controllers can also be optimized using reinforcement learning methods as done in [28, 29, 30]. Recently, [31, 32, 33] proposed using non-rigid registration to transfer human demonstrations of cloth manipulations to real robots and [34] required an adaptive cloth simulator to predict the future state of a cloth. However, these methods require the knowledge of full 3D cloth geometries, which are not available in our applications.

Symbol	Meaning
$\mathcal{C}$	3D configuration space of the cloth
$\mathbf{c}$	a configuration of the cloth
$\mathcal{O}(\mathbf{c})$	an observation of cloth
$\mathbf{c}^*$	target configuration of the cloth
$\mathbf{x}$	robot end-effectors' grasping points
$\mathbf{x}^*$	optimal grasping points returned by the expert
P	transfer function encoding cloth dynamics
$\mathbf{dist}$	distance measure between two observations
π	DOM-control policy
α	random-forest topology
β	controller parameters
γ	confidence of leaf-node
θ	parameter sparsity
K	the number of decision trees
l_k	a leaf-node of k-th decision tree
$l_k(\mathcal{O}(\mathbf{c}))$	the leaf-node that $\mathcal{O}(\mathbf{c})$ belongs to
$\mathcal{L}$	labeling function for optimal actions
$\mathcal{F}$	feature transformation for observation

TABLE I: Symbol table.

III. Problem Formulation

In this section, we introduce our notations and formulate the problem. Our goal is to compute a realtime feedback controller to deform a cloth into an unknown target configuration. We denote the 3D configuration space of the cloth as $\mathcal{C}$. Typically, a configuration $\mathbf{c} \in \mathcal{C}$ can be discretely represented as a 3D mesh of cloth and the dimension of $\mathcal{C}$ can be in the thousands. However, we assume that only a partial observation $\mathcal{O}(\mathbf{c})$ is known, which is an RGB-D image from a single, fixed point of view in our case. The goal of the controller is to transform $\mathbf{c}$ into a target configuration c^*. We assume that, over the entire process of control, the robot grasps the cloth at a fixed set of N points whose coordinates are $\mathbf{x}$, where $|\mathbf{x}| = 3N$ and the control action is constituted by the desired positions of these grasping points, denoted as $\mathbf{x}^*$. Therefore, the controller corresponds to a function:

$$\mathbf{x}^* = \pi(\mathcal{O}(\mathbf{c})|\beta), \tag{1}$$

where β are its learnable parameters. Given x^*, the corresponding joint angles of the robot can then be determined via conventional inverse kinematics. Given the control action, the configuration of the cloth and the grasping points can be given by the following distribution:

$$p(\mathbf{c}_{i+1}, \mathbf{x}_{i+1}|\mathbf{c}_i, \pi(\mathcal{O}(\mathbf{c}_i))). \tag{2}$$

This distribution can be a cloth simulator [35] in a simulated environment or it can be obtained from a real-life robot. Note that, although the action is the desired grasping points ($\mathbf{x}^*$), $\mathbf{x}^*$ and $\mathbf{x}_{i+1}$ are generally not the same because the controller's output can violate a robot's kinematic or dynamic constraints.

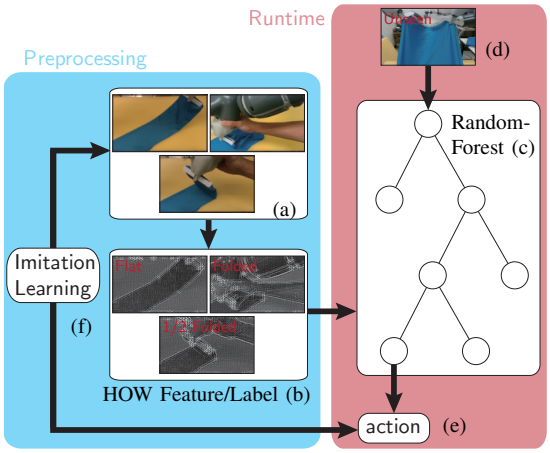


Fig. 2: Approach Pipeline: The pipeline of learning a random-forest-based DOM-controller that maps the visual feature (RGB-D image) to the control action. Given a sampled dataset (a), we first label each data point (shown as red text in (b)) to get a labeled dataset, (b). We then construct a random-forest to classify the images, (c). After training, the random-forest is used as a controller. Given an unseen visual observation (d), the observation is brought through the random-forest to a set of leaf-nodes. The optimal control actions are defined on these leaf-nodes, (e). The entire process of labeling, classification, and controller optimization can be integrated into an IL algorithm, (f).

A. Controller Optimization Problem

Our main goal is to optimize the learnable parameters β to optimize the performance of the controller, π. This controller optimization problem can take different forms depending on the available information about $\mathbf{c}^*$. If $\mathcal{O}(\mathbf{c}^*)$ is known, then we can define a reward function: $R(\mathbf{c}) = -\mathbf{dist}(\mathcal{O}(\mathbf{c}), \mathcal{O}(\mathbf{c}^*))$, where $\mathbf{dist}$ can be any distance measure between RGB-D images. In this setting, we want to solve the following reinforcement learning problem:

$$\underset{\alpha,\beta}{\mathbf{argmax}} \quad \mathbf{E}_{\tau\sim\pi}\left[\sum_{i}^{\infty}\gamma^i R(\mathbf{c}_i)\right] \tag{3}$$

where $\tau = (\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_\infty)$ is a trajectory sampled according to π, γ is the discount factor, and the subscript figures denote the timesteps. Another widely used setting assumes that $\mathcal{O}(c^*)$ is unknown, but that an expert is available to provide an optimal control action $\pi^*(\mathcal{O}(\mathbf{c}))$. The expert is a ground truth controller following the definition of [23]. In this case, we want to solve the following imitation learning problem:

$$\underset{\alpha,\beta}{\mathbf{argmax}} \quad \mathbf{E}_{\tau\sim\pi}\left[\sum_{i}^{\infty}\gamma^i \mathbf{dist}(\pi^*(\mathcal{O}(\mathbf{c}_i)), \pi(\mathcal{O}(\mathbf{c}_i)))\right] \tag{4}$$

This expert can be easily acquired in a typical human-robot collaboration task. Our method is based on the imitation learning formulation.

IV. Learning Random-Forest-Based Controller

To find the controller parameters, we use an imitation learning algorithm [27], which can be decomposed into two sub-steps: online dataset sampling and controller optimization. The first step samples a dataset $\mathcal{D} = \{\langle \mathcal{O}(\mathbf{c}), \mathbf{x}^* \rangle\}$, where each sample is a combination of cloth observation and optimal action. The second step optimizes the random-forest-based controller with respect to β, given $\mathcal{D}$.

A. Feature Extraction

Before constructing the random-forest from $\mathcal{D}$, we apply a feature transform to $\mathcal{D}$. Our raw observation of the cloth, $\mathcal{O}(\mathbf{c})$, is an RGB-D image. it has been noted, (e.g., by [36]) that applying a simple feature transform can improve the accuracy of a classifier such as random-forest. In addition, our input is a 320×240 RGB-D image of the cloth mesh, which corresponds to 76800 entries each having three colors and one depth channel, which is high-dimensional. Therefore, a feature transform effectively reduces the dimensions of the input observation and makes the classifier more robust when the size of the dataset is small.

In our approach, we use HOW-features [7] as the low-dimensional representation. HOW-features is a variant of HOG-features. HOW-features first applies Gabor filters to each patch of the image and then concatenates these patches, resulting in a 768-dimensional feature space. Since each image patch is spatially localized, HOW-features requires each image to be aligned as a pre-processing step. Because our input is an RGB-D image, we can perform a foreground extraction using the depth-channel and then align the image to the center of the screen using the same procedure as in [36]. We summarize this algorithm in Algorithm 1 and denote this feature transform as a function $\mathcal{F}$. The dataset after the feature transform is defined as $\bar{\mathcal{D}} = \{\langle \mathcal{F} \circ \mathcal{O}(\mathbf{c}), \mathbf{x}^* \rangle\}$.

Algorithm 1 Feature extraction operation $\mathcal{F}$.

Input: RGB-D image $\mathcal{O}(\mathbf{c})$
Output: Extracted HOW-feature $\mathcal{F} \circ \mathcal{O}(\mathbf{c})$
1: Foreground extraction using depth channel.
2: Resize/align image to the center of screen using [36].
3: Compute HOW-feature [7].

B. Random-Forest Construction

Our key contribution is to use a random-forest as the underlying learnable controller in an imitation learning framework. A random-forest is an ensemble of K decision trees, where the k-th tree classifies $\mathcal{F} \circ \mathcal{O}(\mathbf{c})$ by bringing it to a leaf-node $l_k(\mathcal{F} \circ \mathcal{O}(\mathbf{c}))$, where $1 \leq l_k(\mathcal{F} \circ \mathcal{O}(\mathbf{c})) \leq L_k$ and L_k is the number of leaf-nodes in the k-th decision tree. The random-forest makes its decision by classifying $\mathcal{F} \circ \mathcal{O}(\mathbf{c})$ using every decision tree and then computing the average over all the decisions of the trees in the forest. To use an already constructed random-forest as a controller, we define an optimal control action $x^*_{l,k}$ so that the final action is determined by averaging:

$$x^* = \pi(\mathcal{O}(\mathbf{c})|\beta) = \frac{1}{K}\sum_{k=1}^{K} x^*_{l_k(\mathcal{F}\circ\mathcal{O}(\mathbf{c})),k}. \tag{5}$$

To construct the random-forest, we use a strategy similar to that in [37]. We construct K binary decision trees in a top-down manner, each using a random subset of $\mathcal{D}$. Specifically,

for each node of a tree, a set of random partitions is computed and the one with the maximal Shannon information gain [38] is adopted. Each tree is grown until a maximum depth is reached or the best Shannon information gain is lower than a threshold. The optimal control action of a leaf-node is defined as the average of the control actions of the data sample belonging to that leaf-node.

C. Imitation Learning

We use an imitation learning algorithm [27] that includes two steps into an outer loop. During each outer iteration, we query an expert, which in our case is a ground-truth hard-coded control algorithm. Specifically, we generate a set of cloth simulation trajectories using a cloth simulator (Equation 2). During each timestep of these trajectories, we query the expert to get an optimal control action $\pi^*(\mathcal{O}(\mathbf{c}))$. This optimal control action is combined with the action proposed by our random-forest $\pi(\mathcal{O}(\mathbf{c}))$. The combined action is fed to the simulator to get the next observation. As a result, more data is added into $\mathcal{D}$ and a new random-forest, β, is constructed from a new $\mathcal{D}$. This algorithm is outlined in Algorithm 2.

Algorithm 2 Training DOM-controller using imitation learning algorithm.

Input: Initial guess of β, optimal policy π^*
Output: Optimized β

▷ imitation learning outer loop
while imitation learning has not converged **do**
  ▷ Generate training data based on current $\pi(\mathcal{O}(\mathbf{c})|\beta)$
  Sample $\mathcal{D}$ by querying π^* as in [27]
  ▷ Extract HOW feature for each data sample
  Define $\bar{\mathcal{D}} = \emptyset$
  for each $\mathcal{O}(\mathbf{c})$ **do**
    Extract HOW feature $\mathcal{F} \circ \mathcal{O}(\mathbf{c})$ as in [7]
    Define $\bar{\mathcal{D}} = \bar{\mathcal{D}} \bigcup \{\langle \mathcal{F} \circ \mathcal{O}(\mathbf{c}), \pi^*(\mathcal{O}(\mathbf{c}))\rangle\}$
  end for
  ▷ Construct random-forest, i.e., β
  for $1 \leq k \leq K$ **do**
    Sample random subset of $\bar{\mathcal{D}}$
    Construct k-th binary decision tree using [37]
  end for
end while

D. Analysis

In typical DOM applications, data are collected using numerical simulations. Unfortunately, the high dimensionality of $\mathbf{c}$ induces a high computational cost for simulations (i.e. evaluating P in Equation 2) and generating a large dataset can be quite difficult. Therefore, we design our method so that it can be used with a small number of data samples. Our method's performance relies on the random-forest's stopping criterion (i.e. the threshold of gain in Shannon entropy). We choose to use a large Shannon entropy threshold so that the random-forest construction stops early, leaving us with a relatively small number of leaf-nodes. We expect that, with a large enough number of imitation learning iterations, the number of nodes in each decision tree of the random-forest will converge. Indeed, such convergence can be guaranteed by the following Lemma.

Lemma: *When the number of imitation learning iterations $N \to \infty$, the distribution incurred by the random-forest-based controller will converge to a stationary distribution and the expected classification error of the random-forest will converge to zero.*

Proof: By assuming that Algorithm 2 generates a controller π^n at the n-th iteration, Lemma 4.1 of [27] showed that π^n incurs a distribution that converges when $n \to \infty$. Obviously, the number of data samples used to train the random-forest also increases to ∞ with $n \to \infty$. The expected error of the random-forest's classification on a stationary distribution converges to zero according to Theorem 5 of [39]. In Section V, we will further show that, empirically, the number of leaf-nodes in the random-forest also converges.

V. Results

We now describe our implementation and the experimental setup on both simulated environments and real robot hardware. We highlight the performance on several manipulation tasks performed by human-robot collaboration. We also highlight the benefits of using a random-forest-based controller by comparing our method with prior approaches. More implementation details are given in [40].

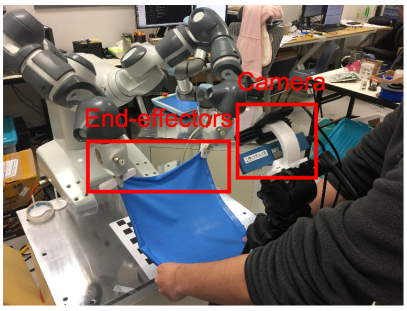


Fig. 3: Setup for Manipulation Tasks: A dual-armed robot and a human are holding four corners of the cloth. We use a 12-DOF dual-armed ABB YuMi and a RealSense RGB-D camera to perform complex manipulation tasks. Our goal is to manipulate a 35cm×30cm rectangular-shaped piece of cloth.

A. Robot Setup

We evaluate our method on a simulated environment. For the simulated environment, the robot's kinematics are simulated using Gazebo [41] and the cloth dynamics are simulated using ArcSim [35], a highly accurate cloth simulator. We use OpenGL to capture RGB-D in this simulated environment. Our goal is to manipulate a 35cm×30cm rectangular piece of cloth with four corners initially located at: $v^0 = (0, 0, 0), v^1 = (0.3, 0, 0), v^2 = (0, 0.35, 0), v^3 = (0.3, 0.35, 0)$(m). Our manipulator holds the first two corners, v^0, v^1, of the cloth and the environmental uncertainty is modeled by having a human hold the last two corners, v^2, v^3, of the cloth so that we have $\mathbf{x} \triangleq (v^0, v^1)^T$ and each control action is 6-dimensional. The

human could move v^2, v^3 to an arbitrary location under the following constraints:

$$\|v^2 - v^3\| \leq 0.3\text{m} \tag{6}$$

$$\|(v^2, v^3)_{i+1}^T - (v^2, v^3)_i^T\|_\infty < 0.1(\text{m/s}), \tag{7}$$

where the first constraint avoids tearing the cloth apart and the second constraint ensures that the speed of the human hand is slow.

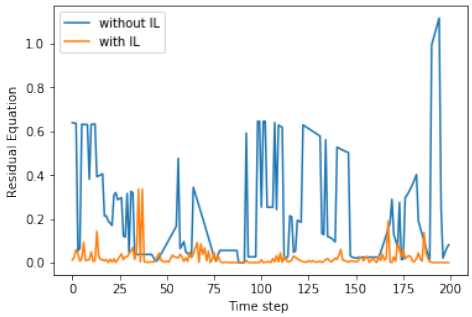


Fig. 4: Robustness of the imitation learning algorithm: In a realtime human-robot interaction, we plot the mean action error (Equation 8). The blue curve shows the performance of a controller trained using only one imitation learning iteration (this choice corresponds to supervised learning [7]) and the orange curve shows the performance of a controller trained with 20 iterations. We compare the residuals (Equation 8) between the two methods. Increasing the number of iterations in imitation learning significantly reduces the mean action error.

B. Synthetic Benchmarks

To evaluate the robustness of our method, we design the 3 manipulation tasks listed below:

- Cloth should remain straight in the direction orthogonal to human hands. This is illustrated in Figure 6 (a). Given v^2, v^3, the robot's end-effector should move to:
$$v^0 = v^2 + 0.35\frac{z \times (v^3 - v^2)}{\|z \times (v^3 - v^2)\|} \; v^1 = v^3 + 0.35\frac{z \times (v^3 - v^2)}{\|z \times (v^3 - v^2)\|}.$$
- Cloth should remain bent in the direction orthogonal to human hands. This is illustrated in Figure 6 (b). Given v^2, v^3, the robot's end-effector should move to:
$$v^0 = v^2 + 0.175\frac{z \times (v^3 - v^2)}{\|z \times (v^3 - v^2)\|} \; v^1 = v^3 + 0.175\frac{z \times (v^3 - v^2)}{\|z \times (v^3 - v^2)\|}.$$
- Cloth should remain twisted along the direction orthogonal to human hands. This is illustrated in Figure 6 (c). Given v^2, v^3, the robot's end-effector should move to:
$$v^0 = \frac{v^2 + v^3}{2} + 0.31\frac{z \times (v^3 - v^2)}{\|z \times (v^3 - v^2)\|} + 0.15\frac{(v^3 - v^2) \times (z \times (v^3 - v^2))}{\|(v^3 - v^2) \times (z \times (v^3 - v^2))\|}$$
$$v^1 = \frac{v^2 + v^3}{2} + 0.31\frac{z \times (v^3 - v^2)}{\|z \times (v^3 - v^2)\|} - 0.15\frac{(v^3 - v^2) \times (z \times (v^3 - v^2))}{\|(v^3 - v^2) \times (z \times (v^3 - v^2))\|}.$$

The above formula for determining v^0, v^1 is used to simulate an expert. Note that these equations for the expert requires the knowledge of the four corner positions of the piece of cloth, and such information may not be available in a real robot system that only observes the cloth using a single RGB(D) image. Therefore, we train our random-forest in a simulated environment. These three equations assume that the expert knows the location of the human hands, but that robot does not have this information and it must infer this latent information from a single-view RGB-D image of the current cloth configuration. We also test the performance on complex benchmarks that combine flattening, folding, and twisting, or have considerable occlusion from a single camera.

Name	Value
Fraction term used in imitation learning algorithm [27]	0.8
Training data collected in each imitation learning iteration	500
Resolution of RGB-D image	640×480
Dimension of HOW-feature used in [7]	768
Random-forest's stopping criterion when impurity decrease less than [37]	1×10^{-4}

TABLE II: Meta-parameters used for training.

Although we have only evaluated our method on a simulated environment, we can also deploy our controller on real robot hardware. For the real robotic environment, we use a RealSense depth camera to capture 640×480 RGB-D images and a 12-DOF ABB YuMi dual-armed manipulator to perform the actions, as illustrated in Figure 6.

To deploy our controller, we first use camera calibration techniques to get both the extrinsic and intrinsic matrix of the RealSense Camera. Second, we compute the camera position, camera orientation, and the clipping range of the simulator from the extracted parameters. Third, we generate a synthetic depth map using these parameters and train the three tasks using the random-forest-based controller parametrization and the imitation learning algorithm. Finally, we randomly perturb the human hand positions when collecting training data to make our random-forest robust to observation noises. A similar technique is used in [42]. We also add visual noises to the training samples and test the algorithm by posing objects between camera and object. After that, we integrate the resulting controller with the ABB YuMi dual-armed robot and the RealSense camera via the ROS platform. As shown in Figure 1, with these identified parameters we can successfully perform the same tasks that were performed in the synthetic benchmarks on the real robot platform.

C. Multi-task Controller

Unlike single-task controller, a multi-task random-forest-based controller stores multiple actions in a leaf-node. Each observed image is classified by each decision tree in a manner that is similar to that of a single-task controller. The leaf node chooses an action according to the id of the task. In this benchmark, we train a 3-task controller for the 3 synthetic tasks in Section V-B. And we transfer the controller to the real robot as benchmark (5) mentioned in Figure 1. We combines straightening, bending and twisting to show that our approach can perform complex tasks, as shown in the video. Moreover, we also show tasks which involve occlusion from a single camera viewpoint by adding noise to inputs.

We compare the performances of a single-task controller and a multi-task controller, both of which are based on random-forests. Again, during each evaluation in the simulated environment, the human hands move to 10 random target positions v^{2*}, v^{3*}. As shown in Figure 7 (red), we profile the residual (Equation 8). Our controller performs consistently well with a relative action error of 0.4954%. We then train a

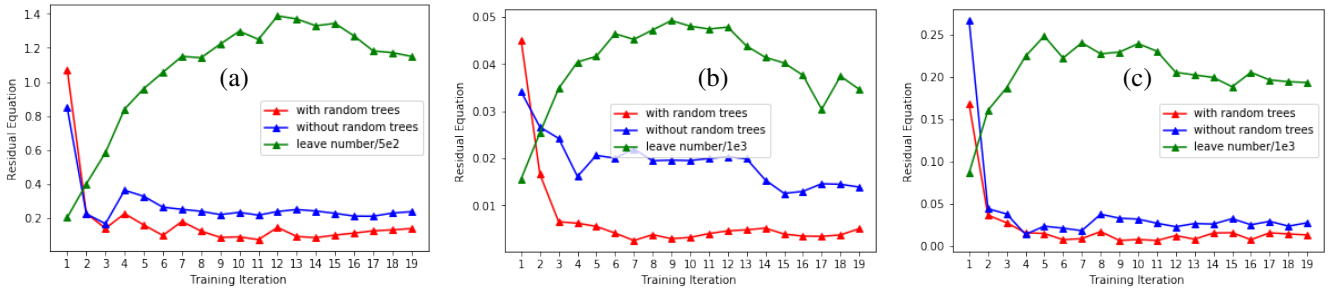


Fig. 5: Controller with and without random-forest: (Red): Residual (Equation 8) plotted against the imitation learning iterations (Line 2 of Algorithm 2). (Green): Number of leaf-nodes plotted against the imitation learning iterations. (Blue): Residual (Equation 8) plotted against the imitation learning iterations, which precludes random-forest construction. (a): Flatten the cloth; (b): Bend the cloth; (c): Twist the cloth.

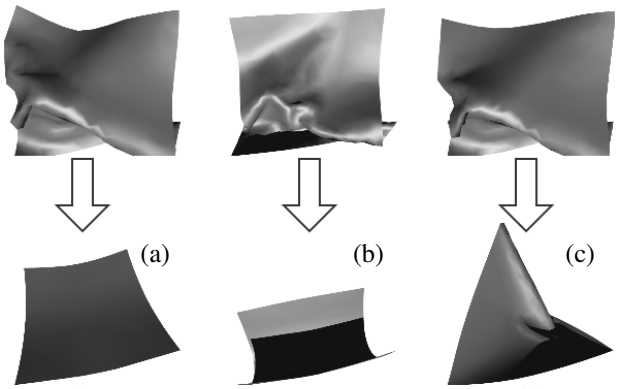


Fig. 6: Synthetic Benchmarks: We highlight the realtime performance of our algorithm on three tasks for the robot simulator (a): keep the cloth straight; (b): keep the cloth bent; (c): keep the cloth twisted.

joint 3-task controller. This is performed by defining a single random-forest and defining 3 optimal actions on each leaf-node. The performance of the 3-task controller is compared with that of the single-task controller in Figure 7. The multi-task controller performs slightly worse in each task, but the difference is quite small.

D. Comparison With Other Solutions

A key feature of our method is that it allows the robot to react to random human movements while the effect of these movements is indirectly reflected via a piece of cloth. This setting is similar to [43]. However, [43] assumes the 3D geometric mesh of cloth c is known without any sensing error, which is not practical.

Our method falls into a broader category of visual-servoing methods, but most previous work in this area (such as [44]) has focused on navigation tasks and there is relatively little work on deformable body manipulation. [45] based their servoing engine on histogram features, which is similar to our use of HOW-features. However, they use direct optimization to minimize the cost function $(\mathbf{dist}(\mathcal{O}(\mathbf{c}), \mathcal{O}(\mathbf{c}^*)))$, which is not possible in our case because our cost function is non-smooth in general.

Finally, our method is closely related to methods in [16, 17], which also use random-forest and store actions on the forest. However, our method is different from prior methods in two ways. First, our controller is continuous in its parameters, which means it can be trained using an imitation learning algorithm. Moreover, we use both feature extraction and controller parametrization in the imitation learning algorithm [27] so that both the feature extractor and the controller benefit from evolving training data.

Training Set Proportion	20%	40%	60%	80%	100%
Random-Forest	0.0154	0.0078	0.0046	0.0040	0.0038
Neural Network	0.0551	0.0469	0.0458	0.0459	0.0451
Linear Regression	$1.66e18$	$4.58e18$	$8.77e17$	$9.23e17$	$8.82e-5$

TABLE III: Comparison with Different Controllers: Residual (Equation 8) of random-forest-based controller, neural-network-based controller [27], and linear regression controller, computed with different proportions of the training set. We use a dataset collected by an expert. The dataset contains 5702 points and we randomly select 20% of the data as the test dataset. The random-forest-based controller exhibits a lower residual. Linear regression increases residual on unseen data. A neural-network-based controller does not fit well when the size of the training set is limited.

To show the benefits of random-forest, we compare three different models of controllers: random-forest, linear regression, and neural network [27]. During each evaluation in the simulated environment, the human hands move to 10 random target positions v^{2*}, v^{3*}. In Table III, we plot of the residual (Equation 8) of the tree methods against the number of imitation learning iterations. On the convergence of Algorithm 2, the random-forest-based controller outperforms the two other opponents, exhibiting a lower residual.

To implement the neural-network-based controller, we use Tensorflow, which is a neural network toolkit. The structure of the neural network is fully connected and consists of a hidden layer of 128 neurons. To implement the linear-regression-based controller, we use the apply the implementation from scikit-learn [46], which is a standard machine learning toolkit. We use the standard parameters from the linear regression module.

E. Benefits of Random-Forest

There are many standard techniques for computing low-dimensional controlling parameters from high-dimensional perceptual data such as RGB images and depth maps. These include standard regression models and neural-network-based models. We evaluate the performance of our algorithm along with the others. The test involves measuring the residual of the manipulator as it moves towards the goal configuration based on the computed control parameters, as given by Equation 8.

We obtain best results in our benchmarks using a random-forest-based controller. Using the random-forest-based con-

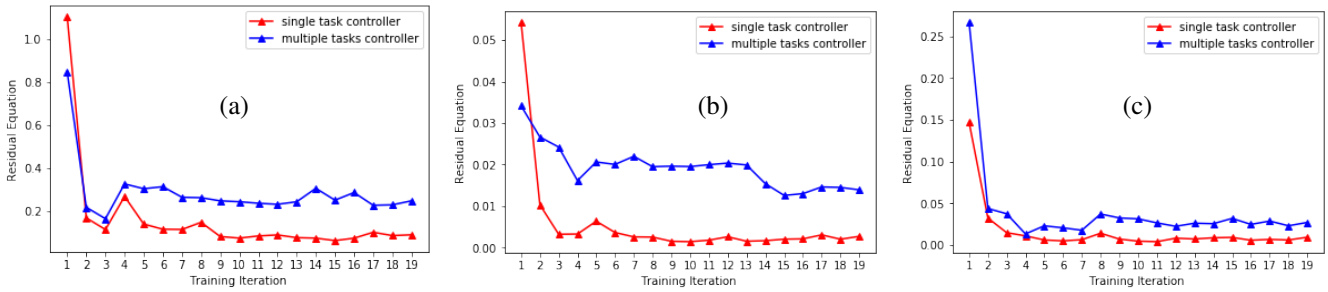


Fig. 7: Multi-Task Controller vs. Single-Task Controller: Residual (Equation 8) using a joint 3-task controller (blue) and a single-task controller (red). (a) Flatten the cloth; (b) Bend the cloth; (c) Twist the cloth. Both controllers converge after a few iterations of the imitation learning algorithm. The single-task controller performs slightly better than the multi-task controller with a relative action error of 0.4954%, but the difference is not significant.

troller and the imitation framework requires fewer parameters to configure a task. Further, the computed control parameters are limited to the labels of the random-forest, which makes the controller robust to the unseen data. In practice, the random-forest-based imitation learning requires fewer computation resources which can enable the controller to be used in real-time applications. The performance is governed by the total number of iterations of the imitation learning. As the number of iterations of imitation learning grows, the residual Equation 8 reduces. After reaching a certain iteration, the imitation learning contributes less to the performance enhancement. In other words, when the imitation learning framework converges, the overall performance of the controller is guaranteed.

VI. Conclusion, Limitations and Future Work

We present a novel controller parametrization for cloth manipulation applications. In our parametrization, the optimal control action is defined on the leaf-nodes of a random-forest. Further, both the random-forest construction and controller optimization are integrated with the imitation learning algorithm and evolve with training data. We evaluate our method using a 3-task cloth manipulation application. The result shows that our method can seamlessly combine feature extraction and controller parametrization problems. In addition, our method is robust to random noises in human motion and observations. Moreover, our controller parametrization can robustly adapt to evolving training data and quickly reduce the mean action error. During testing, the controller performs consistently well in terms of accomplishing the cloth manipulation tasks undergoing very large cloth deformations.

One major limitation is that it is difficult to extend our method to reinforcement learning scenarios because our method is not differentiable when using a random-forest construction. Therefore, reinforcement learning algorithms such as the policy gradient method [47] cannot be used. Another potential drawback is that our method is still sensitive to the random-forest's stopping criterion. In addition, we need additional dimension reduction, i.e. the HOW-feature, and action labeling in the construction of the random-forest. In this work, labeling is done by mean-shift clustering of optimal actions, but in some applications where observations can be semantically labeled, it can be advantageous to label observations instead of actions. For example, in object grasping tasks, we can construct our random-forest to classify object types instead of classifying actions. Finally, our method may not be suitable for high-level manipulation tasks such as cloth folding and laundry cleaning. These problems involve multiple smaller manipulation tasks which require a meta-algorithm that combines these tasks. In addition, these tasks usually require re-grasping between different stages of control, which is outside the domain of this paper.

References

[1] A. Clegg, J. Tan, G. Turk, and C. K. Liu, "Animating human dressing," *ACM Transactions on Graphics*, vol. 34, no. 4, pp. 116:1–116:9, 2015.

[2] Y. Li, Y. Yue, D. Xu, E. Grinspun, and P. K. Allen, "Folding deformable objects using predictive simulation and trajectory optimization," in *IROS*, 2015.

[3] J. Schrimpf and L. E. Wetterwald, "Experiments towards automated sewing with a multi-robot system," in *ICRA*, 2012, pp. 5258–5263.

[4] S. Rodriguez, J.-M. Lien, and N. M. Amato, "Planning motion in completely deformable environments," in *ICRA*, 2006, pp. 2466–2471.

[5] 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. 1, pp. 1334–1373, 2016.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

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

[8] B. Frank, C. Stachniss, N. Abdo, and W. Burgard, "Efficient motion planning for manipulation robots in environments with deformable objects," in *IROS*, 2011, pp. 2180–2185.

[9] M. Saha and P. Isto, "Manipulation planning for deformable linear objects," *IEEE Transactions on Robotics*, vol. 23, no. 6, pp. 1141–1150, 2007.

[10] H. Yuba, S. Arnold, and K. Yamazaki, "Unfolding of a rectangular cloth from unarranged starting shapes by a dual-armed robot with a mechanism for managing recognition error and uncertainty," *Advanced Robotics*, vol. 31, no. 10, pp. 544–556, 2017.

[11] J. Stria, D. Pra, V. Hlav, L. Wagner, V. Petrk, P. Krsek, and V. Smutn, "Garment perception and its folding using a dual-arm robot," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 61–67.

[12] Y. Li, X. Hu, D. Xu, Y. Yue, E. Grinspun, and P. K. Allen, "Multi-sensor surface analysis for robotic ironing," in *ICRA*, 2016, pp. 5670–5676.

[13] J. Schulman, A. Lee, J. Ho, and P. Abbeel, "Tracking deformable objects with point clouds," in *ICRA*, 2013, pp. 1130–1137.

[14] B. Wang, L. Wu, K. Yin, U. Ascher, L. Liu, and H. Huang, "Deformation capture and modeling of soft objects," *ACM Transactions on Graphics*, vol. 34, no. 4, pp. 94:1–94:12, 2015.

[15] I. Leizea, A. Mendizabal, H. Alvarez, I. Aguinaga, D. Borro, and E. Sanchez, "Real-time visual tracking of deformable objects in robot-assisted surgery," *IEEE Computer Graphics and Applications*, vol. 37, no. 1, pp. 56–68, 2017.

[16] A. Doumanoglou, T.-K. Kim, X. Zhao, and S. Malassiotis, *Active Random Forests: An Application to Autonomous Unfolding of Clothes*, 2014, pp. 644–658.

[17] A. Doumanoglou, A. Kargakos, T. K. Kim, and S. Malassiotis, "Autonomous active recognition and unfolding of clothes using random decision forests and probabilistic planning," in *ICRA*, 2014, pp. 987–993.

[18] A. Ramisa, G. Alenya, F. Moreno-Noguer, and C. Torras, "Finddd: A fast 3d descriptor to characterize textiles for robot manipulation," in *IROS*, 2013, pp. 824–830.

[19] P.-C. Yang, K. Sasaki, K. Suzuki, K. Kase, S. Sugano, and T. Ogata, "Repeatable folding task by humanoid robot worker using deep learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 397–403, 2017.

[20] D. Tanaka, S. Arnold, and K. Yamazaki, "Emd net: An encode–manipulate–decode network for cloth manipulation," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1771–1778, 2018.

[21] Z. Hu, P. Sun, and J. Pan, "Three-dimensional deformable object manipulation using fast online gaussian process regression," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 979–986, April 2018.

[22] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[23] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computer Survey*, vol. 50, no. 2, pp. 21:1–21:35, 2017.

[24] R. F. Stengel, *Stochastic Optimal Control: Theory and Application*. New York, NY, USA: John Wiley & Sons, Inc., 1986.

[25] A. X. Lee, S. H. Huang, D. Hadfield-Menell, E. Tzeng, and P. Abbeel, "Unifying scene registration and trajectory optimization for learning from demonstrations with application to manipulation of deformable objects," in *IROS*, 2014, pp. 4402–4407.

[26] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *IROS*, 2012, pp. 5026–5033.

[27] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *AISTATS*, vol. 15, 2011, pp. 627–635.

[28] A. Gupta, C. Eppner, S. Levine, and P. Abbeel, "Learning dexterous manipulation for a soft robotic hand from human demonstrations," in *IROS*, 2016, pp. 3786–3793.

[29] D. Mcconachie and D. Berenson, "Estimating model utility for deformable object manipulation using multiarmed bandit methods," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 3, pp. 967–979, July 2018.

[30] D. Berenson, "Manipulation of deformable objects without modeling and simulating deformation," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov 2013, pp. 4525–4532.

[31] S. H. Huang, J. Pan, G. Mulcaire, and P. Abbeel, "Leveraging appearance priors in non-rigid registration, with application to manipulation of deformable objects," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2015, pp. 878–885.

[32] J. Schulman, J. Ho, C. Lee, and P. Abbeel, *Learning from Demonstrations Through the Use of Non-rigid Registration*. Cham: Springer International Publishing, 2016, pp. 339–354. [Online]. Available: https://doi.org/10.1007/978-3-319-28872-7_20

[33] 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," *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5265–5272, 2015.

[34] D. Navarro-Alarcon, H. M. Yip, Z. Wang, Y. Liu, F. Zhong, T. Zhang, and P. Li, "Automatic 3-d manipulation of soft objects by robotic arms with an adaptive deformation model," *IEEE Transactions on Robotics*, vol. 32, no. 2, pp. 429–441, April 2016.

[35] R. Narain, A. Samii, and J. F. O'Brien, "Adaptive anisotropic remeshing for cloth simulation," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 152:1–152:10, 2012.

[36] G. Rogez, J. Rihan, S. Ramalingam, C. Orrite, and P. H. S. Torr, "Randomized trees for human pose detection," in *CVPR*, 2008, pp. 1–8.

[37] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *CVPR*, 2011, pp. 1297–1304.

[38] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[39] G. Biau, "Analysis of a random forests model," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 1063–1095, 2012.

[40] B. Jia, Z. Pan, Z. Hu, J. Pan, and D. Manocha, "Cloth manipulation using random forest-based controller parametrization," *arXiv.org*, p. 1802.09661, 2018.

[41] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IROS*, 2004, pp. 2149–2154 vol.3.

[42] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30, 2017.

[43] T. Wada, S. Hirai, and S. Kawamura, "Indirect simultaneous positioning operations of extensionally deformable objects," in *IROS*, 1998, pp. 1333–1338 vol.2.

[44] A. X. Lee, S. Levine, and P. Abbeel, "Learning visual servoing with deep features and fitted q-iteration," 2017.

[45] Q. Bateux and E. Marchand, "Histograms-based visual servoing," *IEEE Robotics and Automation Letters*, vol. 2, no. 1, pp. 80–87, 2017.

[46] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[47] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *IROS*, 2006, pp. 2219–2225.

VII. APPENDIX

A. Complexity and Algorithm Properties

As illustrated in Algorithm 2, the complexity of our overall approach mainly depends on three parts: dataset sampling, feature extraction, and random-forest construction. When constructing a single decision tree based on the sampled dataset $\bar{D}$, the complexity has an upper bound of $O(|\bar{D}|^2)$. For the construction of a random-forest with K decision trees, the complexity is $O(K|\bar{D}|^2)$.

To evaluate the performance of each component in our method, we run several variants of Algorithm 2. All the meta-parameters used for training are illustrated in Table II. In our first set of experiments, we train a single-task random-forest-based controller for each task and profile the mean action error:

$$err \quad = \quad \sum_{\langle \mathcal{O}(c), \mathbf{x}^* \rangle} \frac{1}{|\mathbf{x}^*||\bar{\mathcal{D}}|} \|\mathbf{x}^* - \frac{1}{K} \mathbf{x}^*_{l_k(\mathcal{F} \circ \mathcal{O}(c)),k}\|^2, \quad (8)$$

with respect to the number of imitation learning iterations (Line 2 of Algorithm 2). As illustrated in Figure 5 (red), the action error reduces quickly within the first few iterations and later converges. We also plot the number of leaf-nodes in our random-forest in Figure 5 (green). As more iterations are performed, the number of leaf-nodes in our random-forest also converges.