Combining Imitation and Reinforcement Learning to Fold Deformable Planar Objects

Benjamin Balaguer and Stefano Carpin

***Abstract*— Research on robotic manipulation has primarily focused on grasping rigid objects using a single manipulator. It is however evident that in order to be truly pervasive, service robots will need to handle deformable objects, possibly with two arms. In this paper we tackle the problem of using cooperative manipulators to perform towel folding tasks. Differently from other approaches, our method executes what we call a *momentum fold* - a swinging motion that exploits the dynamics of the object being manipulated. We propose a new learning algorithm that combines imitation and reinforcement learning. Human demonstrations are used to reduce the search space of the reinforcement learning algorithm, which then quickly converges to its final solution. The strengths of the algorithm come from its efficient processing, fast learning capabilities, absence of a deformable object model, and applicability to other problems exhibiting temporally incoherent parameter spaces. A wide range of experiments were performed on a robotic platform, demonstrating the algorithm's capability and practicality.**

I. Introduction

The popularity of service robotics has unveiled a multitude of novel challenges that researchers need to undertake before "a robot in every home" [5] can become a reality. Research involving highly deformable object manipulation with cooperative manipulators is however still in its infancy. The inadequacy of deformable object models for robotic applications [6], the absence of high-fidelity simulation tools for deformable objects, and the lack of literature on the subject are all factors delaying the development of robots capable of performing a variety of tasks involving flexible objects. Moreover, the ability to grasp, manipulate, and interact with deformable objects are primordial behaviors for robots to be part of our everyday lives, since most objects we use daily are deformable. The types of object we are interested in are highly deformable and we undertake the specific problem of folding rectangular towels or napkins using two manipulators working cooperatively.

We exploit machine learning techniques, a dynamic field that is becoming more and more popular in robotics, to discard the requisite for a deformable object model, one of the major obstacles when working with deformable objects. Even though reinforcement learning has been shown to solve diverse tasks ranging from controlling a quadruped robotic dog [20] to playing the ball-in-a-cup game [8], flipping pancakes [10], weightlifting [15], and performing archery [11], towel folding offers different research challenges: learning for two independent manipulators working cooperatively; exploiting a temporally incoherent parameter space (i.e. two or more successful folds can take a different amount of time to perform); dealing with an action-to-reward function composed of many-to-one mappings (i.e. there are many different ways to appropriately fold a towel). Due to the wide range of possible manipulator movements that yield correct folds, we combine human-to-robot imitation learning with reinforcement learning to not only converge faster to a solution, but also explore a wider range of the parameter space and find the action most replicable on the robot platform. The contributions of this manuscript are the following:

- we present an algorithm that exploits human imitations with reinforcement learning;
- the algorithm we present is capable of learning from temporally incoherent examples;
- we present a model-free approach to deformable object manipulation;
- the algorithm is shown to be running in real-time and capable of online learning.

The rest of the paper is organized as follows. We start by describing, in Section II, related works relevant to both our folding application and machine learning. A formal description of the problem we are addressing is given in Section III, followed by our training data acquisition and proposed approach in Section IV and V, respectively. In Section VI, we present the experiments performed on our robotic platform. We conclude the paper with final remarks and future work in Section VII.

II. Related Work

The problem of using robotic manipulators to fold deformable objects has been studied before, although it has frequently relied on imperfect deformable object models or highly specialized robots. A review of deformable object models is beyond the scope of this paper, as is deformable one-dimensional object models, but interested readers can examine [6] from the computer animation community. The robotics community has devised its own deformable object model where the object is decomposed into rigid links and foldable creases, resulting in a well-understood kinematic description. This simplified representation has been successfully applied to metal bending processes [7], carton folding [12], paper craft [18], and towel folding [1]. Each aforementioned application has drawbacks, however, in that they do not generalize well [7], [1], or do not take into account the actuating robot when choosing a folding sequence using path planners like PRMs [18] or RRTs [12]. The kinematic representation is only suitable for deformable objects that

School of Engineering, University of California, Merced, CA, USA, {bbalaguer,scarpin}@ucmerced.edu

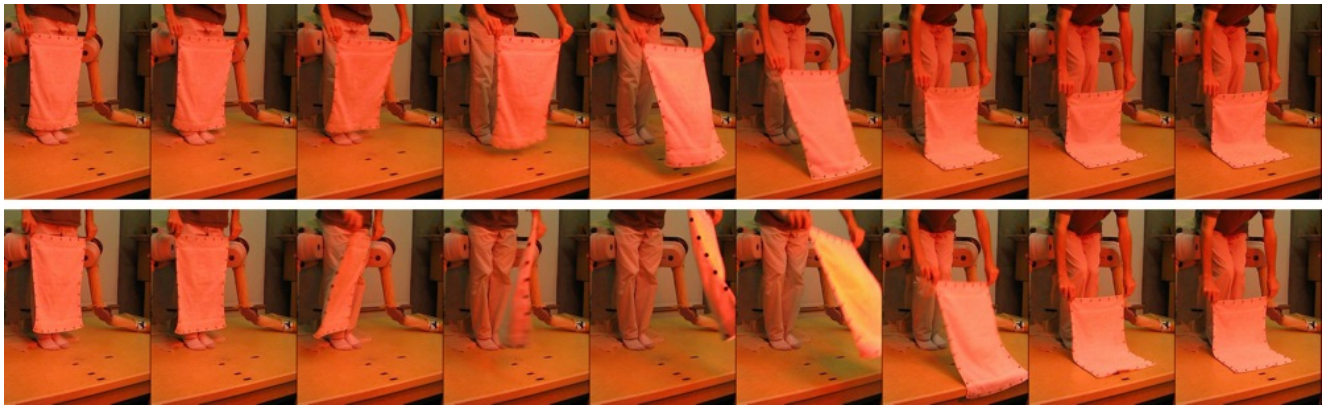


Fig. 1. Two different, yet successful, momentum folds demonstrated by a human to the robot.

retain their shape (e.g. metal) and cannot be used effectively for highly deformable objects. More practical robot systems have been designed for origami [2] and T-Shirt [3] folding. These robots are engineered for their specific tasks, however, and would be unsuitable for service robotics where one robot is assigned highly heterogeneous goals. Kobori et al. have developed an algorithm allowing an HRP-2 humanoid to unfold clothes [9]. Their approach is model-based, in that it extracts and tracks clothes' regions and requires apriori information regarding the clothes' regions and their states. One of the most promising work comes from Abeel et al. who have demonstrated the folding of towels [13] and clothes [21]. Their work depends on a parameterized shape model [14] created by a human and, as such, does not necessarily generalize to pieces of clothing that were not already parameterized. Additionally, the folding sequences are either pre-programmed or need to be manually entered.

Typical off-the-book gradient-based policy learning approaches to learning [19] have seen little practical use in the robotics community, mainly due to the lack of adaptability to high-dimensional control and the manual parameter-tuning of the learning rate. Theodorou et al. realized these problems and implemented a reinforcement learning algorithm called Policy Improvements with Path Integrals (PI^2) [20]. The authors' algorithm is capable of learning parameterized policies by using stochastic optimal control with path integrals. PI^2 does not require parameter tuning, although it requires an initial seed behavior that might be difficult to obtain, and works well with high-dimensional data, as exemplified by the learning of how to jump as far as possible on a quadruped robotic dog. Policy learning by Weighting Exploration with Returns (PoWER) [8] also solves the same problems seen in gradient-based policy learning and is, arguably, one of the leading algorithms when it comes to reinforcement learning for manipulation. Indeed, within a very short time, it has been applied to a great number of heterogeneous applications including the ball-in-a-cup task [8], flipping pancakes [10], and performing archery [11]. PoWER is based on Expectation-Maximization, exploits a weighted sampling technique for exploration of the parameters space, and only requires an example motion to bootstrap the algorithm.

In previous works, the line between imitation and reinforcement learning is often blurred. Our definition of imitation learning, and the nomenclature used in this paper, follows that of Schaal et al. as "a complex set of mechanisms that map an observed movement of a teacher onto one's own movement apparatus" [16]. The distinct features between the two are that reinforcement learning exploits a trial-and-error methodology whereas imitation learning does not. Consequently, imitation learning requires multiple sample demonstrations in order to learn something. Related works on imitation learning differ greatly from our approach since we focus on imitation for the purpose of reinforcement learning. Interested readers should see [16] for a good review of imitation learning techniques in robotics.

III. Problem Definition

The problem we aim to solve is to fold a towel symmetrically, where one half of the towel is folded on top of the other. Evidently, there are many ways that such a task can be performed and we choose to follow what we refer to as a *momentum fold*, where the force applied to grasping points on the towel is used to give momentum to the towel and lay half of it flat on the table (see Figure 1 and the accompanying video for examples of momentum folds). We note that the momentum fold is used to make sure that half of the towel lays flat on the table and, as such, this is the motion we are trying to learn. Once that state is achieved, we can straightforwardly apply motion planning to finish the fold, as shown in [1]. While the majority of previous works utilizing reinforcement learning bootstrap their algorithm using kinesthetic teaching (i.e. having a human perform actions directly on a gravity-compensated robotic manipulator and recording the parameters from the robot), the fact that we are dealing with two manipulators renders this method impractical, if not impossible. Consequently, a human demonstrates an appropriate folding motion to the robot, two examples of which are shown in Figure 1. We assume that the towel can be picked by the robot and put into a starting position similar to the one in the first frame of Figure 1, a preliminary step previously solved in [4].

We designed and implemented a hybrid method that incorporates imitation and reinforcement learning. There are two reasons for incorporating imitation learning. First, from an algorithmic standpoint, exploiting knowledge acquired from human imitations can drastically reduce the parameter search space that the reinforcement learning algorithm has to explore, as will be shown in subsequent sections. Indeed, searching the parameter space based on previously-acquired rewards is both time-consuming and unnecessary and we harness the power of imitation learning to make the search more efficient. Second, from a more practical perspective, we cannot use kinesthetic teaching for folding applications. Since we have to use a human demonstrator and not all motions performed by a human will be replicable on a robot due to mechanical constraints, it is beneficial to acquire multiple demonstrations and to learn from them.

We conclude this section introducing the symbols we will use in subsequent sections:

- O_i: i-th observation. The trajectory followed by k points on the towel during one fold motion. Each point, sampled at a constant frequency, is in 3D Cartesian space.
- θ_i: i-th action sequence. The trajectory followed by the pinch grasp of the robot, expressed in Cartesian space and sampled at a constant frequency.

In essence, we are trying to learn the relationship between θ_i and O_i in order to be able to reproduce desired observation sequences leading to successful folds.

IV. Training Data for Imitation Learning

We start by describing the procedure used to gather training data for imitation learning, whose goal is to acquire action-observation pairs that produce good folding motions. We view the demonstrator as giving perfect examples of how to accomplish the task and, as such, implicitly have maximum rewards for any action-observation pairs produced during the training data acquisition.

The demonstrator performs momentum folds, as exemplified in Figure 1. We only gather positive examples due to the fact that the action space for negative examples is too large and unstructured. In order to facilitate the collection of actions and observations, we use a motion capture system with eight infrared cameras along with a towel comprised of reflective markers that can be tracked by the system. Specifically, we use $k = 28$ markers uniformly spaced around the towel's four edges. The motion capture system has a millimeter precision and records data up to 120 Hz.

Motion capture is prone to both false positives and negatives. As is the case with many machine learning algorithms, our method necessitates fixed-size vectors (i.e. it needs to constantly track all 28 points at every time step). Therefore, it is crucial that both false positives and negatives are handled correctly. False positives often occur due to changes in illumination, reflective objects or materials, and movements that are too close to the infrared cameras. To find and remove false positives, we exploit the fact that, with a sampling rate of 120 Hz, points do not move a lot between two successive frames. Therefore we compute the nearest neighbors between time frames t and $t-1$, as shown in Figure 2(a). The Euclidean distance between the points in frame t and their respective nearest neighbors in frame $t-1$ are computed. Any points whose distance is above a threshold ε (we set ε to 1 centimeter) is labeled as a false positive and removed from the data set at time t (see Figure 2(b)). This process assumes that the first frame at time $t = 0$ contains all markers, a fair assumption since we can make sure all markers are correctly detected before starting a folding motion. This simple false positives removal method is efficient and successfully removes 100% of the false positives in our data sets.

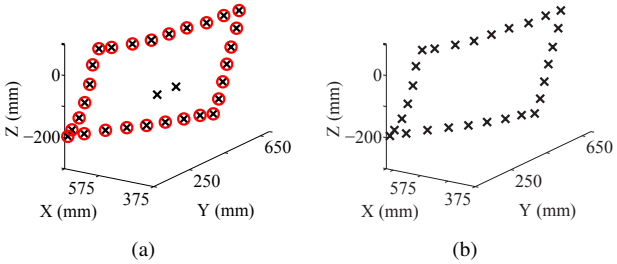


Fig. 2. Figure 2(a) shows the data before rectification (X marks) and the nearest neighbors computed based on the previous time frame (O marks). Note the two false positives in the middle of the towel. Figure 2(b) shows the resulting data points, after false positives have been removed.

False negatives take place when a marker on the towel is not registered by the motion capture system. These events mainly occur due to occlusions, are more frequent, and more challenging to deal with since we need to recover where the data points should be on the towel. We start by automatically labeling each marker on the towel as being part of the upper, lower, left, and right edges. This process can be implemented by exploiting the towel's rectangular shape at the first time step and using nearest neighbor distances for each subsequent time steps. We take advantage of the fact that all of the points forming an edge roughly exist in a two-dimensional plane and we reconstruct the false negatives in that plane. By knowing where each point lies on the towel, we can deduce which edges of the towel are missing markers, as shown in Figure 3(a), indicating that false negatives have occurred. For each data point on a towel's edge containing false negatives, we compute the best-fit plane using orthogonal distance regression [17], an example of which is shown in Figure 3(b). We then project all of the points forming the edge onto the plane using QR decomposition, consequently reducing the dimensions of the reconstruction problem from 3 to 2. In two-dimensional space, we find the best-fit 3rd-degree polynomial using the Vandermonde matrix to establish a least-squares problem. It is then possible to reconstruct the missing data points, in two-dimensional space, using the polynomial's equation and the location of the current data points, as demonstrated in Figure 3(c). We finally reconstruct the false negatives by projecting the two-dimensional points back into three dimensions, the final results of which can be seen in Figure 3(d). While being more complicated than

the false positives and having the implicit assumption that enough data points must be present for the polynomial construction, the proposed method is very efficient (since we are working with few data points in low dimensions) and accurate. In fact, the reconstruction only failed 11.11% of time, each due to a lack of data points resulting in a poor polynomial function. Furthermore, observers could not distinguish between data points that were acquired directly through motion capture and those that were reconstructed. With the aforementioned process, we are guaranteed to have data for all of the 28 markers at 120 Hz.

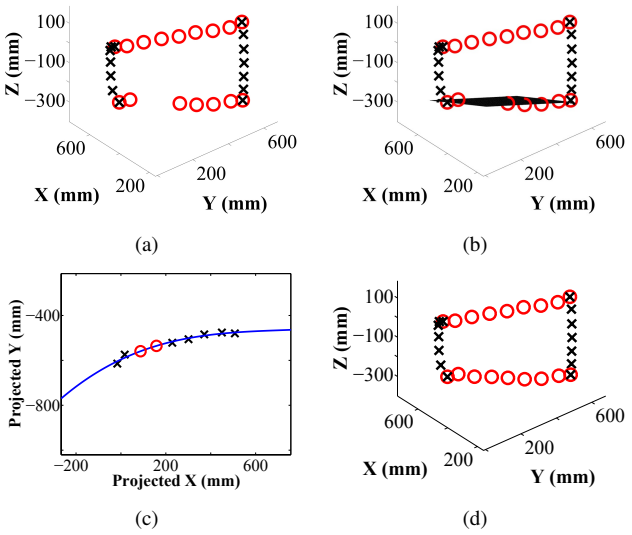


Fig. 3. Recovering false negatives. For Figures 3(a), 3(b), and 3(d), points on the upper and lower edges are marked with O and points on the left and right edges are marked with X. Figure 3(a) shows a time frame with two false negatives on the towel's lower edge. Figure 3(b) shows the best-fit plane applied to the lower edge. Figure 3(c) illustrates the polynomial curve fitting, which is built from the markers shown as X and used to recover the false positives marked as O. Figure 3(d) reveals the reconstructed towel.

We are now faced with the problem of *temporally incoherent motion sequences*. This means that multiple equally valid folding motions will take different execution times. This issue is evidenced by the two examples in Figure 1, where one folding motion is finished before the other one. While this is an additional issue that needs to be addressed, it brings up the additional benefit of covering a greater range of the action space while learning. More formally, the i-th observation sequence, O_i, is comprised of the observation's time and of the Cartesian coordinates for the 28 markers at each time step of the motion. Similarly, the i-th action sequence, θ_i, contains the trajectories of the two control points. Example folding motions each take different times to execute (between, approximately, 3.5 and 5.5 seconds). Consequently, each observation sequence lies in different dimensional-subspaces ranging from $\mathbb{R}^{35700}$ to $\mathbb{R}^{56100}$. Similarly, each action sequence ranges from $\mathbb{R}^{2940}$ to $\mathbb{R}^{4620}$. Working with fixed-sized feature vectors is primordial for most learning techniques and, as such, we convert our training data to fixed-sized vectors. Moreover we down-sample training data from 120 Hz to 30 Hz to reduce the complexity of the problem. The reduction from 120 Hz to 30 Hz was chosen because we have empirically determined that the same amount of variance was captured, using Principal Component Analysis (PCA), as shown in Figure 4. By using 30 Hz, along with an average folding length of approximately 5 seconds, we dictate the number of time frames in our sequences ($30 \times 5 = 150$). The number of time frames (150), along with the data recorded for each observations and actions, determines the size of our vectors, namely $O_i \in \mathbb{R}^{12750}$ and $A_i \in \mathbb{R}^{1050}$. We conclude this session by mentioning that we collect 80 different folding sequences for our training data, which, when oriented in a matrix as rows, create our training data set where $O^t \in \mathbb{R}^{80\times12750}$ and $\theta^t \in \mathbb{R}^{80\times1050}$.

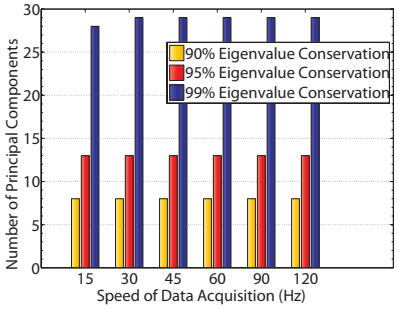


Fig. 4. The number of principal components that account for 90%, 95%, 99% of variance after down-sampling the data from 120 Hz to 15 Hz. The bar graph shows that a substantially equivalent amount of information is retained when down-sampling from 120 Hz to 30 Hz.

V. Proposed Approach

A. Reward Function

As for any reinforcement learning algorithm, it is necessary to evaluate the algorithm's exploration of the action space and guide its search. We start by defining a reward function used in both the imitation and reinforcement learning phases of the algorithm. The reward function $R(O^t, O^c)$ computes the reward for a new observation O^c based on all the observations O^t acquired during training. Its pseudo-code is shown in Algorithm 1. We note that $O^t \in \mathbb{R}^{80\times12750}$ whereas $O^c \in \mathbb{R}^{12750}$ and we use $O_i \in O^t$, in line 2 of the algorithm, to indicate that we pick the i-th sample from O^t, such that $O_i \in \mathbb{R}^{12750}$. In line 3 and 4 of the algorithm, we extract the three-dimensional data points of the last time frame for the training sample and current observation, respectively. In line 5, we run the Iterative Closest Point (ICP) algorithm to transpose both sets of data points and compute the average error, in millimeters, between the points. We repeat these steps for each sample in our training data and retain the smallest average error. Our reward is then the exponential function of the negative smallest average error in decimeter.

The reward function finds the best match between the current observation and any observations that has been recorded during training. According to our previous hypotheses, the reward function assumes that any motion in the training

Algorithm 1 Computation of $R(O^t, O^c)$

$minAvgError \leftarrow 1000$
for all $O_i \in O^t$ **do**
  $Training \leftarrow LastFrame(O_i)$
  $Current \leftarrow LastFrame(O^c)$
  $AvgError \leftarrow$ ICP($Trainning$, $Current$)
  if $AvgError \leq minAvgError$ **then**
    $minAvgError \leftarrow AvgError$
  end if
end for
return $\exp(-minAvgError/100)$

set gets the highest possible reward and, consequently, we consider every training sample when calculating a reward. An exponential function is used to generate rewards between 0 and 1, as required when using the PoWER reinforcement learning algorithm [8]. We note that, while the training motions look similar, they can yield rewards with variations of up to 15% when compared amongst each other. Since we are only trying to match 28 points in three-dimensional space, ICP is very efficient and provides the additional benefit of making the reward function translation and rotation invariant. We conclude this subsection by mentioning an interesting tradeoff to be considered with such a reward function. Instead of using the last time step of the observations, one could run ICP at every time step, or a uniformly sampled selection of time steps. By only using the last time step, we are rewarding the robot for reaching a good final towel configuration, regardless of how it got there. By introducing more time steps into the reward function, one could potentially influence the robot's behavior so that it more closely mimics human behavior. Evidently, the addition of more time steps would increase the time complexity of the reward function.

B. Imitation Learning

We use imitation learning as a two-layered hierarchical approach to reduce the search space of the reinforcement learning algorithm. In the initial *exploratory layer* we use training data to let the robot attempt to execute a set of diverse folding sequences. Next, in the *expansion layer* we expand the search to motions similar to the best one that was found during the exploratory layer. The idea behind the two-layer imitation learning section is to explore the action space based on human demonstrations, the best results of which will be used as seeds for the reinforcement learning algorithm. Since we have acquired training data using human demonstrations, the exploratory layer is crucial in eliminating, based on our reward function, motions that the robot cannot replicate successfully or that do not yield good folding motions. Indeed, there is no guarantee that the motions generated by a human will be reproducible by the robot due to, among others, mechanical constraints and joint or torque limits. In the unlikely case that none of the motions are replicable by the robot, the reinforcement learning will start from a very bad seed, requiring a lot of exploration and converging very slowly.

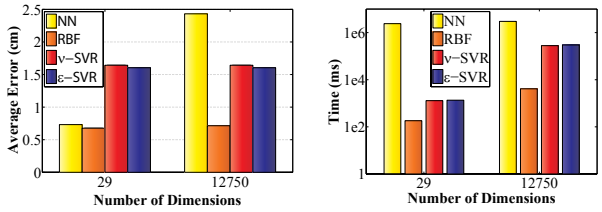
As the name suggests, the aim of the exploratory layer is to explore and find different motions in the trained action space, θ^t. Since it would be too time consuming to execute all the motions and there is some redundancy in the trained action space, we apply Lloyd's k-means clustering algorithm where we optimize the following function, where C_j, θ_i, and μ_j represent the j-th cluster, i-th training action, and j-th cluster's mean, respectively.

$$\arg\min_{C} \sum_{j=1}^{M} \sum_{\theta_i \in C_j} ||\theta_i - \mu_j||$$

We use $M = 10$ clusters, implicitly trying to find the most diverse set of 10 folding motions, although M can be modified to dictate how much initial exploration we want the robot to perform. Evidently, M dictates a tradeoff between time and amount of exploration, since more clusters will result in more exploration of the action space, but will take longer to perform. Each cluster C_j is represented by its mean, μ_j. When imitating the motions on the robot, we cannot use the mean directly since it is simply a mathematical average and might end up producing a bad folding motion. Consequently, for each cluster C_j, we find the Euclidean nearest neighbor between θ_i and μ_j where $\theta_i \in C_j$. In other words, we have assured that the M different actions are part of our training data. As a result, we have a set of M actions, $\theta^{Explore} = [\theta_1^{Explore}\theta_2^{Explore}\dots\theta_k^{Explore}]$ with $\theta_i^{Explore} \in C_i \in \theta^t$. We let the robot execute each encoded trajectory, $\theta_i^{Explore}$, record its corresponding observation, $O_i^{Explore}$, and calculate the motion's reward using $R_i^{Explore} = R(O^t, O_i^{Explore})$. We note that the M actions are temporally incoherent and will yield different execution time.

In the expansion layer, the action space is further explored, starting with the best folding motion that the robot produced. Formally, we find the best folding motion, $\theta_{Best}^{Explore}$, based on the collected rewards in the exploration layer, where $Best = \arg\max_i(R_i^{Explore})$. In this layer, we exploit imitation learning to expand the search around $\theta_{Best}^{Explore}$. We have already deduced that $\theta_{Best}^{Explore}$ provides the best folding motion in the exploration layer and want to explore the action-space around it. Consequently, we need to generate new actions. With the action space being so large and with a relatively small region encompassing valid fold motions, randomly exploring the space or sampling directly from a distribution will in all likelihood be inefficient. Instead, we train a learning algorithm using our training observations, O^t, and actions, θ^t, to learn the function $f : O_i \rightarrow \theta_i$. In other words, given an observation sequence, O_i, we want to find its corresponding action, θ_i. Evidently, the data stored inside O_i is highly correlated and a lot of redundancy exists between both the data points and the successive time frames. Keeping this remark in mind, we apply PCA to our observation data, O^t, which leads to a new observation data set, $\hat{O}^t$, projected in a lower-dimensional subspace where $\hat{O}^t \in \mathbb{R}^{80\times 29}$. The 29 dimensions were chosen by maintaining 99% of the data's variance (see Figure 4). Learning is achieved by using $\hat{O}^t$ with Radial Basis Functions (RBF), since we empirically

determined that it yielded better accuracy and trained faster than Neural Networks (NN), ν Support Vector Regression (ν-SVR), and ε Support Vector Regression (ε-SVR), as shown in Figure 5. The accuracy measures were calculated by training with 90% of the data and classifying the remaining 10%. We note that using the dimensionally-reduced data, $\hat{O}^t$, provides slightly better results due to the fact that potential noise has been removed from the data and a simpler problem needs to be learned. Additionally, the average error is very low, 0.6767cm, which is much less than the mechanical inaccuracy of the manipulator we use. We mention that RBF training is very efficient, taking 181.6ms, which would easily allow for unsupervised online learning as the robot performs new motions, a task that we leave for future work.



(a) (b)

Fig. 5. Figure 5(a) and 5(b) show the accuracy and training time for the NN, RBF, ν-SVR, and ε-SVR learning algorithms for both the dimensionally-reduced data (29) and the full data (12750). The reader shall note that we use log-scale for the Y-axis of Figure 5(b).

The RBF requires an observation as input and will output the action matching that observation. Consequently, we need a process that generates a new observation, which is then fed to the RBF. The entire process generates a new action, θ_s^{Expand}, using the Expand function shown in Algorithm 2. In line 3 and 4, we fit a multivariate Gaussian distribution to the training data. The dimensionality of the multivariate Gaussian (line 1) is 29, as dictated by the dimensional reduction through PCA. In line 6, we sample an observation, $\hat{O}^s$, from the multivariate Gaussian distribution and compare, in line 7, the time of the sampled observation, $\hat{O}^s$, and best action executed during the exploration stage, $\theta_{Best}^{Explore}$. The sampling process is repeated until the time difference between the two is less than threshold ε, which we set to 0.2 seconds. We finally generate a new action, θ_s^{Expand}, by feeding our sampled observation, $\hat{O}^s$, into the trained RBF.

The time check on line 7 of the algorithm is performed to compensate for the temporal incoherencies inherently encoded in our training data, where two valid folds can take a different amount of time to execute. We use the motion's execution times to increase the likelihood that generated actions will be similar to $\theta_{Best}^{Explore}$, since it is unlikely that two folding motions that are performed in a similar amount of time will be drastically different. In the unlikely case that they are different, the resulting motion will be executed on the robot and likely result in a poor reward relative to $\theta_{Best}^{Explore}$. Similarly to the exploratory layer, there is no guarantee that the robot will be able to successfully execute the generated actions, in which case the

Algorithm 2 Expand($\hat{O}^t$, $\theta_{Best}^{Explore}$, RBF, ϵ)

1: $n = \text{NumColumns}(\hat{O}^t)$ // $n = 29$ in our case
2: $\hat{O}^t \sim [\hat{O}_1^t \;\; \hat{O}_2^t \ldots \hat{O}_n^t]$
3: $\mu = [E[\hat{O}_1^t] \;\; E[\hat{O}_2^t] \ldots E[\hat{O}_n^t]]$
4: $\Sigma = [\text{Cov}(\hat{O}_i^t, \hat{O}_j^t)]_{i=1,2,\ldots,n;j=1,2,\ldots,n}$
5: **repeat**
6: Sample $\hat{O}^s$ from $f(x)$
s.t. $f(x) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} e^{\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)}$
7: **until** $|\text{Time}(\hat{O}^s)\text{-Time}(\theta_{Best}^{Explore})| \leq \varepsilon$
8: θ_s^{Expand} =RBF($\hat{O}^s$)
9: **return** θ_s^{Expand}

reinforcement learning will require more exploration. We run Algorithm 2 l times, resulting in a set of l new actions

$$\theta^{Expand} = [\theta_1^{Expand}\theta_2^{Expand}\ldots\theta_l^{Expand}].$$

For all of the experiments in this paper, $l = 5$, expanding $\theta_{Best}^{Explore}$ to five new, yet similar, actions. We let the robot execute each encoded trajectory, θ_i^{Expand}, record its corresponding observation, O_i^{Expand}, and calculate the motion's reward using $R_i^{Expand} = R(O^t, O_i^{Expand})$. The best folding motions acquired from both the exploration and expansion layers of our imitation algorithm will provide a good seed for reinforcement learning, allowing it to quickly converge.

C. Reinforcement Learning

We finalize the algorithm using a modified version of the state-of-the-art reinforcement algorithm PoWER [8]. PoWER tries to find new actions in such a way that the expected rewards of the rollouts is maximized. The process is iterative and the action performed at time n is updated to produce a new action θ_{n+1} for the next rollout. The process is repeated until convergence, which we choose to be when the last three rollouts' rewards are within 0.1% of each other. PoWER's update function does not work for our application, so we modify it to be

$$\theta_{n+1}^{RL} = \theta_n^{RL} + \left(\theta_{Top} - \theta_n^{RL}\right)\left[R(O^t, O_{Top}) - R(O^t, O_n)\right]$$

where Top is the index of the action that resulted in the best reward among the actions $[\theta_{Best}^{Explore}\theta_1^{Expand}\ldots\theta_l^{Expand}\theta_1^{RL}\ldots\theta_{n-1}^{RL}]$. The update function is modified to account for two major issues that occur when using PoWER's unmodified update function for our folding task. Our first modification is to only use the best action, as designated by Top, rather than employing importance sampling, which takes into account the best σ actions (i.e. σ is 1 in our case). The problem with importance sampling with our folding motions comes from the fact that very different folds lying in different regions of the action space can yield similarly high rewards (see Figure 1 for an example). This phenomenon happens quite frequently and would result in poor and slow learning performance since it would confuse the algorithm with

contrasting information. Additionally, small potential time incoherencies between multiple high reward actions can quickly lead the exploration into an action space region that is either not executable by the robot or does not resemble a folding motion. The more actions are considered with importance sampling (i.e. the greater the σ), the higher likelihood for this phenomenon to take place. In our second modification, we include the reward of the last rollout, $R(O^t, O_n)$, to influence the speed of the exploration. More specifically, the term $[R(O^t, O_{Top}) - R(O^t, O_n)]$ in the update function allows for a fine-grain search when the current action's reward, θ_n^{RL}, is close to the best action's reward, θ_{Top}. Conversely, the further our current action's reward is from the best action's reward, the more space we cover during the update step.

VI. Experimental Results

We present a real-world evaluation of the proposed algorithm on our robotic platform, a static torso comprised of two seven degree-of-freedoms (DoF) Barrett Arms, each equipped with the Barrett Hand. The task of the robot is to successfully symmetrically fold a thin hand-towel that is both light and highly susceptible to air flow resistance. As previously mentioned, the training data is comprised of 80 folding motions performed by the human demonstrator and each action is encoded as timed Cartesian coordinates for both manipulators. In order to play actions on the robot, we convert the Cartesian coordinates to robot configurations, for each manipulator, using inverse kinematics (IK). More specifically, our IK solver analytically solves for 6 DoF by uniformly sampling possible joint values for the 7th, and redundant, DoF. This yields a large space of solutions, which is dependent on the sampling step. We select the IK solution that is both collision-free and minimizes the joints' movement from the previous time frame, in order to reduce the amount of torque that the joints experience. A starting robot configuration was manually selected and the towel is grasped using a pinch grasp. Figure 7 shows some rollouts performed by our robotic platform.

The exploratory and expansion layers of the algorithm operate in constant time, since they always yield the same number of actions to be performed by the robot. Specifically, we always run the exploratory layer 10 times and the expansion layer 5 times. Once all 15 motions have been played back on the robot, the reinforcement learning iterates until convergence, which we define to be when the last three rewards are all within 0.001 of each other. Once the folding motion converges, the towel is folded using our previously published algorithm [1]. Figure 6 shows the resulting rewards for a learning session. In the exploratory and expansion stages of the algorithm, the rewards oscillate, in no particular order since the actions are acquired independently of each other, as the robot tries to find a good seed for the learning algorithm. The reinforcement algorithm converges in 4 steps since high-quality motions were found during the exploratory and expansion stages of the algorithm. Overall, only 19 rollouts are required to converge to a good solution for this complex dynamic task, as opposed to 50 or 75 rollouts for similar tasks [8], [10]. We note that the fast convergence is entirely due to the very good starting seed acquired through the first 15 rollouts, thus the benefit of combining our hierarchical imitation learning algorithm with reinforcement learning. Even though Reinforcement Learning is not necessary for the application presented, since a couple of very high quality actions were found in the exploratory layer, the algorithm is built for applications when that might not be the case, thus requiring the expansion and Reinforcement Learning layers. The accompanying video shows the robot performing all 19 trials. We encourage readers to go to our website[1] for more videos, including the human demonstrations and more trials, along with the data sets used in the paper.

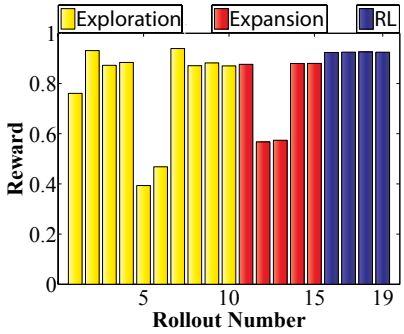


Fig. 6. Rewards given to the robot for each rollout performed.

VII. Conclusions and Future Work

We have shown that the combination of imitation and reinforcement learning provides a notable benefit to learning complex tasks. Indeed, once the exploratory and expansion steps are completed with the help of imitation learning, the reinforcement learning algorithm converges extremely quickly thanks to a very good starting seed. The approach is especially suited for tasks with different but equally-appropriate ways of solving them, where human-like motions are desirable, or where kinesthetic learning is impractical or impossible (e.g. when using two or more manipulators). These tasks range from folding towels or clothes to opening letters or boxes, tying knots, and loading or unloading grocery bags. The absence of an object model is a welcomed benefit, especially when dealing with deformable objects. We have demonstrated that the problem of time incoherencies in the training data, which are notoriously difficult to deal with, can be circumvented with our imitation learning algorithm. Last but not least, the algorithm runs in real time.

There are a few directions for future work. From a practical standpoint, the data acquisition using the motion capture system needs to be replaced by a user-friendly and cost-effective solution. Based on our data, only 29 dimensions were required to successfully interpret a 3- to 5-second towel motion, leading us to believe that some simple image processing with stereo vision or an inexpensive

[1] http://robotics.ucmerced.edu/Robotics/IROS2011/

Fig. 7. Rollouts producing rewards of 0.57391 (top) and 0.93905 (middle), along with final folding motion (bottom).

sensor such as Microsoft's Kinect might be sufficient for similar tasks. Evidently, new feature vectors would have to be acquired (e.g. SIFT features, corner or edge detection) and the algorithm would learn a different function. We note that this extension is entirely dependent on the learning algorithm and, as such, the camera viewpoint would not affect the results (assuming that the same viewpoint is used during training). An interesting algorithmic extension would be to add online learning to the approach. As we perform more actions using reinforcement learning and get an idea of how good they are through the reward function, we might not want to discard that information especially if it is likely that the task or environment conditions will change.

ACKNOWLEDGMENTS

This work is partially supported by the National Science Foundation under grant BCS-0821766.

REFERENCES

[1] B. Balaguer and S. Carpin. Motion planning for cooperative manipulators folding flexible planar objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3842–3847, 2010.

[2] D. Balkcon. *Robotic Origami Folding*. PhD thesis, Carnegie Mellon University, 2004.

[3] M. Bell and D. Balkcom. Grasping non-stretchable cloth polygons. *International Journal of Robotics Research*, 2009.

[4] M. Cusumano-Towner, A. Singh, S. Miller, J. O'Brien, and P. Abbeel. Bringing clothing into desired configurations with limited perception. In *IEEE International Conference on Robotics and Automation*, 2011.

[5] B. Gates. A robot in every home. *Scientific American Magazine*, December:58–65, 2006.

[6] S. Gibson and B. Mirtich. A survey of deformable modeling in computer graphics. Technical report, Mitsubishi Electric Research Laboratories, 1997.

[7] S. Gupta, D. Bourne, K. Kim, and S. Krishnan. Automated process planning for robotic sheet metal bending operations. *Journal of Manufacturing Systems*, 17(5):338–360, 1998.

[8] J. Kober and J. Peters. Learning motor primitives for robotics. In *IEEE International Conference on Robotics and Automation*, pages 2112–2118, 2009.

[9] H. Kobori, Y. Kakiuchi, K. Okada, and M. Inaba. Recognition and motion primitives for autonomous clothes unfolding for humanoid robot. In *International Conference on Intelligent Autonomous Systems*, pages 57–66, 2010.

[10] P. Kormushev, S. Calinon, and D. Caldwell. Robot motor skill coordination with em-based reinforcement learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3232–3237, 2010.

[11] P. Kormushev, S. Calinon, R. Saegusa, and G. Metta. Learning the skill of archery by a humanoid robot icub. In *IEEE/RAS International Conference on Humanoids Robots*, pages 417–423, 2010.

[12] L. Lu and S. Akella. Folding cartons with fixtures: a motion planning approach. *IEEE Transactions on Robotics and Automation*, 16(4):346–356, 2000.

[13] J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *IEEE International Conference on Robotics and Automation*, pages 2308–2315, 2010.

[14] S. Miller, M. Fritz, T. Darrell, and P. Abbeel. Parametrized shape models for clothing. In *IEEE International Conference on Robotics and Automation*, 2011.

[15] M. Rosenstein, A. Barto, and R. Van Emmerik. Learning at the level of synergies for a robot weightlifter. *Robotics and Automation Systems*, 54(8):706–717, 2006.

[16] S. Schaal, A. Ijspeert, and A. Billard. Computational approaches to motor learning by imitation. *Philosophical Transaction of the Royal Society of London: Series B, Biological Sciences*, 358(1431):537–547, 2003.

[17] C. Shakarji. Least-squares fitting algorithms of the NIST algorithm testing system. *Journal of Research of the National Institute of Standards and Technology*, 103(6):633–641, 1998.

[18] G. Song and N. Amato. A motion planning approach to folding: From paper craft to protein folding. *IEEE Transactions on Robotics and Automation*, 20(1):60–71, 2004.

[19] R. Sutton and A. Barto. *Reinforcement Learning: an Introduction*. MIT Press, 1998.

[20] E. Theodorou, J. Buchli, and S. Schaal. Reinforcement learning of motor skills in high dimensions: a path integral approach. In *IEEE International Conference on Robotics and Automation*, pages 2397–2403, 2010.

[21] J. van den Berg, S. Miller, K. Goldberg, and P. Abbeel. Gravity-based robotic cloth folding. In *International Workshop on "The Algorithmic Foundations of Robotics" at WAFR*, 2010.