2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Congress Center Hamburg
Sept 28 - Oct 2, 2015. Hamburg, Germany

Self-Improvement of Learned Action Models with Learned Goal Models

Baris Akgun and Andrea L. Thomaz

Abstract— **We introduce a new method for robots to further improve upon skills acquired through Learning from Demonstration. Previously, we have introduced a method to learn both an action model to execute the skill and a goal model to monitor the execution of the skill. In this paper we show how to use the learned goal models to improve the learned action models autonomously, without further user interaction. Trajectories are sampled from the action model and executed on the robot. The goal model then labels them as success or failure and the successful ones are used to update the action model. We introduce an adaptive sampling method to speed up convergence. We show through both simulation and real robot experiments that our method can fix a failed action model.**

I. INTRODUCTION

There is a growing interest in deploying robot learning from demonstration (LfD) systems in the real world. The majority of the end-users of these systems will be *naïve* in the sense that they will not have robotics or machine learning experience. This necessitates LfD algorithms that will be able to work with naïve user demonstrations.

In a typical skill learning from demonstration setting, an action model is learned for executing the skill. In our user studies of naïve users teaching robots [1], we have observed that during their demonstrations these users concentrate on achieving the goal of the skill, rather than providing quality demonstrations of how to do it. This makes it challenging to learn successful action models from naïve users.

Motivated by this, we have developed an approach to learn a separate action model and goal model from user demonstrations for a given skill [2]. Action models are used to execute the skill and the goal models are used to monitor the success of that execution. It was shown that all users were able to teach successful goal models but only a subset of them were able to teach successful action models. The goal models were able to correctly label executions as successful or not even in the case of suboptimal action models. In this paper, we build atop this monitoring performance and introduce a novel method for using these learned goal models to guide self-improvement of the action model.

Our approach samples trajectories from the action model and executes them on the robot. If the execution is deemed successful by the goal model, it is used to update the action model, else it is discarded. We want to improve the success of the action model. We describe an adaptive sampling method to speed up convergence and compare it with a non-adaptive version. Our experimental results show that the learned goal models can be used to improve the learned action model beyond the initial demonstrations.

In our work, we utilize *keyframes demonstrations*, [1], [3] to get data from users. Keyframes are sparse (in time) set of sequential points that the teacher demonstrates to the robot. Users can effectively use keyframes during demonstrations in the context of LfD. Furthermore, keyframes help the users to provide more consistent demonstrations as well as giving them a tool to highlight salient parts of the skill, thus providing necessary information to build a good goal model.

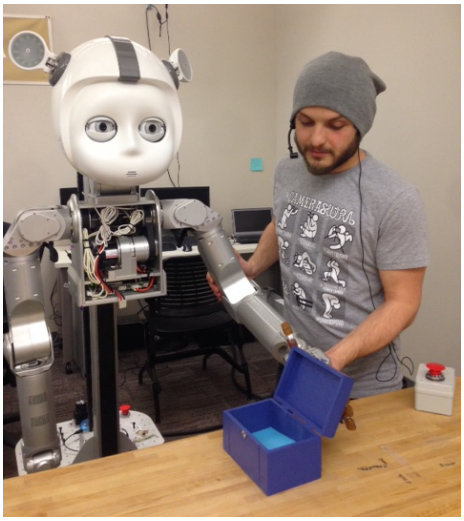


Fig. 1. A teacher providing a kinesthetic demonstration of close the box skill to the robot.

II. RELATED WORK

In general, robots are difficult to program. In addition, the needed programs may not be apparent before a robot is deployed. These two are arguably the main reasons for the field of LfD. There is a large body of work in the field and a general survey can be found in [4].

Some difficult to program skills have easy to represent goals or cost. In such cases, a reinforcement learning (RL) approach is viable. However, traditional RL methods do not scale well with high number of dimensions. Policy search methods have been shown to be suitable for skill learning with robots with high number of degrees-of-freedom (dof). In most of these methods a potentially unsuccessful initial policy is learned from demonstrations, which is then input to the policy search method along with the reward function. Surveys for RL in robotics can be found in [5], [6]. In our work, we do not specify a reward function to the robot.

Another approach to skill learning, is inverse reinforcement learning (IRL) [7] or similarly inverse optimal control (IOC) [8]. In IRL, a reward or cost function is estimated from demonstrations and then used to extract a policy. The main idea behind the IRL approaches is that the reward function is a better representation of the demonstrated skill than the policy. Our goal learning idea is similar to the main idea of inverse reinforcement learning (IRL); extracting a reward

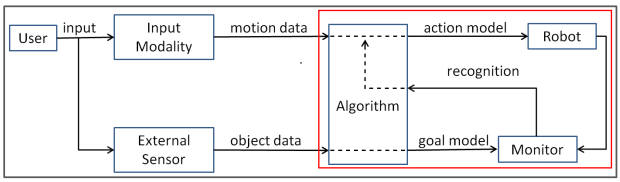


Fig. 2. The LfD system. User demonstrates the skill with keyframes and two types of data is extracted at each keyframe; motion data and object data . The same algorithm is used to learn two distinct models from the aforementioned data; an action model and a goal model. The learned action model is used to execute the skill and the learned goal model is used to monitor the execution. The monitoring output is used to update the action model during self-improvement. The red box indicates the self-improvement part of the system which is the main focus of this paper.

function from demonstrations.

A similar idea related to goal models and monitoring is presented in [9]. The robot executes its learned skill, collects sensory data and the successful executions are labeled by hand. The robot then builds a Gaussian model for the trajectories for each sensory state dimension, which requires a high number of skill executions. These models are used in hypothesis testing during future executions to monitor the skill. In contrast, we use the sensory data obtained during the skill demonstrations to learn a goal model without further manual labelling and skill repetition.

III. ACTION AND GOAL MODELS

Our work concentrates on object-based manipulation skills in which dynamics is not a major component of the goal. This class of skills encapsulates many day-to-day activities (e.g., fetching items, general cleanup, aspects of doing laundry or ironing, etc.). Keyframes are highly suitable for these skills.

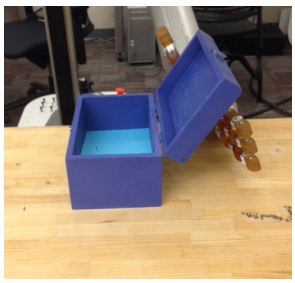
The LfD system used in this work is depicted in Fig. 2. This section provides a summary of our approach to learn and use action and goal models. Refer to [2] for further details.

A. Data

The teachers physically guide the robot's arm to provide demonstrations as shown in Fig. 1, which is called kinesthetic teaching. We utilize *keyframes*, which are sparse (in time) set of sequential points, to get data from users. During demonstrations, the teacher moves the robot to the desired pose and marks a keyframe. Two types of data are recorded at each keyframe; *motion data* and *object data*.

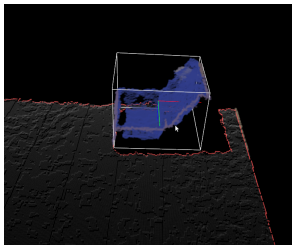
The motion data used to learn the action model is the end-effector pose with respect to the target object. We represent the end effector pose as the concatenation of a 3D vector as the translational component and a unit quaternion (4D) as the rotational component, resulting in a 7D vector, living in $\mathbb{R}^7$. The end-effector poses are transformed to the object reference frame before being input to learning. This transformed pose is called the *action keyframe*. A pose is projected onto the space of rigid body transformations, SE(3), by normalizing the quaternion part wherever necessary.

We use $x_i^d \in \mathbb{R}^7$ to denote the i^{th} action keyframe for the d^{th} demonstration, $X^d = \left\{x_1^d, x_2^d, \ldots, x_{m(d)}^d\right\}$ to denote the d^{th} demonstration. The number of keyframes in the d^{th} demonstration is denoted by $m(d)$. Finally, $D_A = \left\{X^1, \ldots, X^n\right\}$ is the set of n action demonstrations.

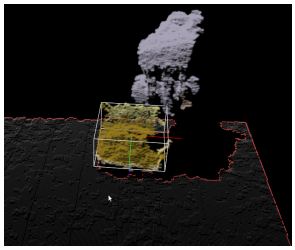


(a) A snapshot of a keyframe from the close the box skill.

(b) A snapshot of a keyframe from the pour skill.



(c) A segmented box for close the box skill.



(d) A segmented bowl for the pour skill.

Fig. 3. Image snapshots as seen by the overhead camera.

The object data consists of features extracted from an overhead RGBD camera output. We make two assumptions about the objects and the environment: (1) the objects sit on a plane (*e.g.* tabletop) and (2) the objects have relatively solid color. We segment the object using the approach in [10] to find spatial clusters of similar color.

After segmentation, we fit a rotated bounding box to the object and use the pose of this box as the object pose. An example of the segmentation and the bounding box results can be seen in Fig. 3. After fitting this box, we extract generic object features that include the bounding box coordinates and orientation, cluster centroid, minimums and maximums of the point cloud coordinates, average RGB values, average hue, point cloud size, bounding box size, volume, area and aspect ratio and bounding box area to volume ratio and bounding box volume to point cloud size ratio. We map the color values between 0 and 1. In addition we use the View Point Feature Histogram (VFH) descriptors presented in [11]. with 15 bins per angle. We remove the viewpoint component. The histogram related features are normalized proportional to the total number of points in the histogram. The resulting goal space is 71 $(26+15\times 3)$ dimensional, treated as $\mathbb{R}^{71}$. This feature vector is called the *goal keyframe*.

We use $z_i^d \in \mathbb{R}^{71}$ to denote the i^{th} goal keyframe for the d^{th} demonstration, $Z^d = \left\{z_1^d, z_2^d, \ldots, z_{m(j)}^d\right\}$ to denote the d^{th} demonstration. The number of keyframes in the d^{th} demonstration is denoted by $m(d)$. Finally, $D_G = \left\{Z^1, \ldots, Z^n\right\}$ is the set of n goal demonstrations.

B. Learning Action and Goal Models

Hidden Markov Models (HMM) are used to represent both the action model and the goal model of the skills. Keyframes lend themselves naturally to such a model since they can be treated as sequential observations. We model the emissions

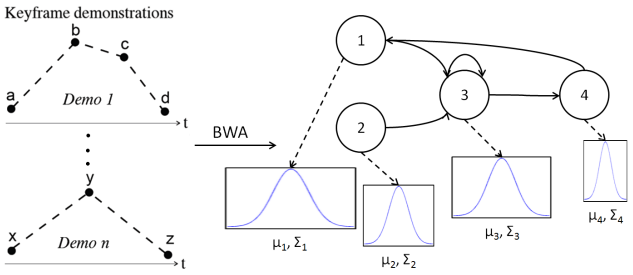


Fig. 4. A depiction of the learning process. The model a represents a HMM with states, non-zero transition probabilities and emission probabilities. In addition, we learn prior and terminal probabilities.

as multivariate Gaussian distributions on the corresponding state space (either the action space or the goal space).

The HMM's are parameterized as follows; $\theta = \{\pi, \zeta, A, \Phi\}$ with n number of states, where π is the prior probability vector, ζ is the terminal probability vector, A is the state transition matrix and $\Phi = \{\mu_1, \mu_2, \ldots, \mu_n, \Sigma_1, \Sigma_2, \ldots, \Sigma_n\}$ is the set of means and covariance matrices of the emission distributions. The terminal probabilities represent the likelihood of states being the last state for the HMM. The states with non-zero prior probabilities are called the *prior states*, $S_\pi = \{s_k : \pi(s_k) > \varepsilon\}$, and the states with non-zero terminal probabilities are called the *terminal states*, $S_\zeta = \{s_l : \zeta(s_l) > \varepsilon\}$. The θ_A represents the action HMM and the θ_G represents the goal HMM.

The HMMs are trained with the Baum-Welch algorithm (BWA). Bayesian Information Criterion (BIC) is used to select the number of states of the HMMs. We run BWA 10 times given a number of states and select the model with the highest likelihood to calculate BIC. The models are learned from multiple demonstrations. An overview of the process is depicted in Fig. 4. The action keyframes are used to learn the action model and the goal keyframes are used to learn the goal model, *i.e.* $\theta_A \leftarrow BWA(D_A)$ and $\theta_A \leftarrow BWA(D_G)$.

C. Action Execution and Goal Monitoring

The skill is executed by generating a trajectory from the action HMM. The first step is to generate a state path by finding the maximum likelihood path between the prior and terminal states by using the transition matrix (A_A).

The next step is the *sample* procedure, in which we sample the emission probabilities along this state path. If, instead of sampling, the means of the emission probabilities are used, this become the *optimalPath* procedure. Both of these procedures result in a sequence of object relative end-effector poses. To execute the skill, the poses are transformed back to the robot frame, given the current object frame (ρ_{obj}). The transformed poses are called the *execution keyframes*.

A 5^{th} order spline is fit between the execution keyframes to create a trajectory to be executed on the robot, which we call the *execute* procedure. During execution of the skill, the robot extracts object data at each execution keyframe it passes through to get a resulting observation sequence $Q = \{q_1, q_2, \ldots, q_p\}$.

In the monitoring step, we use the goal HMM to calculate the likelihood of this observation sequence Q, $p(Q;\theta_G)$ and threshold it to decide whether the executed skill succeeded or not *i.e.* the skill is deemed successful if $log(p(Q;\theta_G)) > \tau_s$. The terminal probabilities are included in the likelihood calculation to make sure that the skill has finished.

IV. SELF-IMPROVEMENT

There are cases where end-users are not able to teach acceptable skills to the robot. In these cases, an extra self-improvement step is needed to have an acceptable model of the skill.

In this section we introduce a self-improvement method that can find an acceptable skill model. We have seen that naïve users are able to teach successful goal models even if their action models are not entirely successful. In our method, we leverage these successful goal models to guide the self-improvement process. This alleviates the need to program a reward function, which would be quite difficult for typical end-users to do or may not be possible at all.

A. Overview

The approach introduced in this work starts from the learned action model, executes sampled trajectories on the robot and utilizes the output of the goal model to update the action model. The main assumption is that there is a successful goal model available after the initial user demonstrations.

The algorithm is presented in Alg. 1. It is an iterated algorithm that takes the user action demonstrations D_A, and the HMM models, θ_A and θ_G as inputs. For a given iteration, the robot samples from the action HMM (θ_A), executes the obtained trajectory and monitors it (lines 7-9), as described in Sec. III-C. If the execution is deemed successful, the sample is added to the set of successful examples, σ (lines 11-13) and the monitoring result is stored in G (line 10). After a number of iterations (n_r), the robot re-learns the action model using σ as described in Sec. III-B. This overall process, which we call an *episode*, is repeated for a predetermined number of times (n_e) but a stopping condition can be used.

Successful sampled trajectories are more relevant to learning the skill than user demonstrations since they are executed by the robot and user demonstrations can potentially be bad. Hence, the user demonstrations are "forgotten" if there are sufficient successful samples (lines 15-17).

We use two version of the sampling step of this algorithm (line 7). In the non-adaptive case, we directly use normal sampling, as described in Sec. III-C. In the adaptive case, we modify HMM sampling by incorporating a *sampling step* which is calculated according to the success of the last w samples (lines 5-6). This is introduced in the next section.

B. Adaptive Sampling

The self-improvement method we describe is essentially a search guided by the goal model and we introduce an adaptive sampling approach to adjust this search. We assume that the best opportunity for learning is on the border of success and failure, *i.e.* point of maximum entropy. In other words, we want to fail and succeed the same number of times during our search to maximize the information gain. One assumption we make is that the learned action model is either within the boundary of success or close enough to

Algorithm 1 Self-Improvement(θ_G, θ_A, D_A)

1: $\sigma \leftarrow D_A$
2: $G \leftarrow [0,1,\ldots,0,1]$
3: **for** 1 **to** n_e **do**
4: **for** 1 **to** n_r **do**
5: $r = \Sigma_{i=1}^{w}(G[(end-w+i):end])/w$
6: $\lambda = f(r,h,\alpha)$
7: $T = sample(\theta_A, \lambda)$
8: $Q = execute(T, \rho_{obj})$
9: $g = monitor(\theta_G, Q)$
10: $G \leftarrow g$
11: **if** $g == 1$ **then**
12: $\sigma \leftarrow T$
13: **end if**
14: **end for**
15: **if** $forgetUserData(G,k)$ **then**
16: $\sigma = \sigma \setminus D_A$
17: **end if**
18: $\theta_A = learn(\sigma)$
19: **end for**

the boundary to be found. This is similar to the assumption of the initial model being within the basin of attraction of a successful local minima in policy search methods.

A multiplication factor for the covariance matrices, λ, is added to the *sample* procedure described in Sec. III-C. Instead of sampling from $\mathcal{N}(\mu,\Sigma)$, we sample from $\mathcal{N}(\mu,\lambda\Sigma)$ to get execution keyframes, where $\lambda \geq 1$. This allows the method to scale its search to be between the vicinity of the current model and farther out.

Eq. 1 shows the calculation of the step size parameter (line 6).The r represents the success ratio of the last w samples as calculated at line 5 of the Alg. 1, hence $0 \leq r \leq 1$. Note that for $r = 0$, the equation is undefined but as $r \to 0$, $f(r,h,\alpha) \to 1+\alpha$. The G parameter is initialized (line 2) with a set of equal number of $1's$ and $0's$ to avoid r being over-sensitive to initial sampling results. The α parameter is responsible for the maximum step size and h is responsible for the width of the function. The output of this function for a few example parameters is shown in Fig. 5.

$$f(r,h,\alpha) = 1+\alpha\left(1-exp\left(-\frac{log^2(\frac{1-r}{r})}{h}\right)\right) \quad (1)$$

For $0 \leq r \leq 1$, the Eq. 1 is symmetric, non-negative and $1 \leq f(r,h,\alpha) \leq 1+\alpha$. It reaches its minimum at $r = 0.5$ and maximum at $r = 0$ and $r = 1$. These properties result in the self-improvement method to look farther out if the latest sampling results are similar and to stay within the vicinity of the current action model when the sampling results are different. This forces the algorithm to spend more time close to the success/failure boundary, as previously motivated.

V. EVALUATION

The experimental setup can be seen in Fig. 1. The robot used, *Curi*, has two 7-dof series-elastically actuated arms. The arms have software gravity compensation to aid in kinesthetic teaching. There is an overhead ASUS Xtion Pro

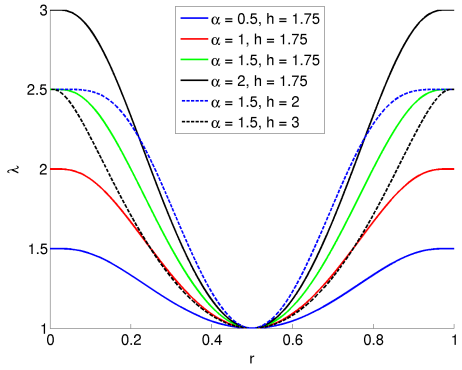
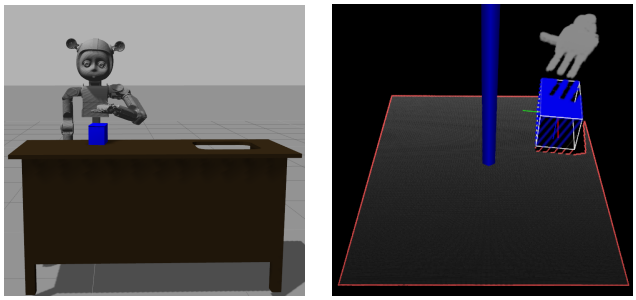


Fig. 5. The step-size parameter (λ) versus sampling success ratio for various values of α and h as calculated by the Eq. 1.



(a) The robot in simulation (b) The perception in simulation

Fig. 6. The simulated environment.

LIVE RGBD camera for perception and the Point Cloud Library [1] is used to process point cloud data.

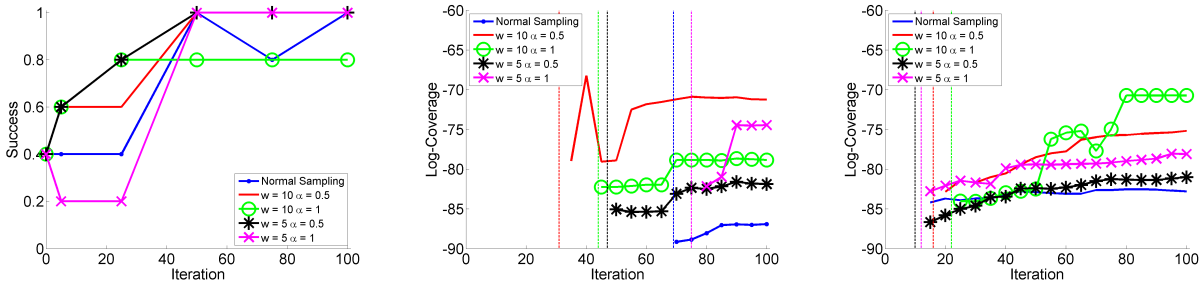
We use both simulation and real robot experiments to evaluate our approach. The evaluation starts by demonstrating skills to the robot and learning goal and action models. Then, these models are input to the self-improvement algorithm where the action models are updated. In both cases, the real robot is used to provide demonstrations and the simulated robot is programmed to mimic the real one.

A. Simulation Results

The simulator used is Gazebo 4.0[2]. Screenshots from the simulation and object segmentation (see Sec. III-A) from simulated data can be seen in Fig. 6. We use the *close the box* (CLB) skill to evaluate our approach in simulation; in which the goal is to close the lid of an open box.

Our method is tested with two initial action models, one successful (success rate 100%) and one unsuccessful (success rate 40%). Rather than purposely providing bad demonstrations, bad demonstration data is generated by modifying good ones; by adding a constant bias of $0.027m$ to the vertical and horizontal dimensions of the second keyframe, forcing it away from the box. For reference, the box dimensions are $0.165m \times 0.108m \times 0.103m$. The unsuccessful action model is then learned from this modified data. The goal models are shared for both action models.

[1] http://pointclouds.org/
[2] http://gazebosim.org/



(a) Success rates for episodes $\{1,5,10,15,20\}$ for the Unsuccessful Initial Model (b) Coverage for the Unsuccessful Initial Model (c) Coverage for the Successful Initial Model

Fig. 7. Simulation: The success rates and the coverage of the action models versus iterations of the self-improvement algorithm for the close the box skill. The vertical dashed lines represent the point of forgetting the user data.

We test our approach under multiple parameter instantiations of the Eq. 1. We fix the width parameter to be constant, $h = 1.75$ and vary the success ratio window size, w and the maximum step size, α. The list of parameters we use is $[\alpha = \{0.5, 1\}, w = \{5, 10\}]$,. We discard the user data after getting $k = 10$ successful samples (Alg. 1, line 16). We run the the algorithm for $n_e = 20$ episodes with $n_r = 5$ iterations each. The goal model log-likelihood decision threshold is set at $\tau_s = -600$. This threshold is based on our previous experience and is not set using data from the current work. In addition, we run our algorithm without adaptive sampling and compare it against the adaptive sampling version.

The results for the close the box skill is given in Fig. 7. After each episode, we learn a new HMM. The Fig. 7(a) shows the average success over 5 executions for a selected set of learned HMMs. Eventually, all the instances of the algorithm reach a successful ($\geq 80\%$) state within 10 episodes.

The interesting result is that the case without adaptive sampling (normal sampling) also managed to improve the skill model. The results for the successful initial model is not shown, since, they are all 100%. One assumption of our method is that the initial action model is not too far from the achieving the skill (see Sec. IV-B). In this case, successful skill executions were within the variance of the initial unsuccessful goal model and hence were able to recover an acceptable action model. We expect the adaptive sampling to have more impact when the initial model is farther away, which is actually the case for robot trials.

The volume of its emission probabilities is an indicator of the state space coverage for an action model. We take this volume to be the volume of the hyper-ellipsoids defined by the covariance matrices and define *coverage* of an action model to be the sum of the determinants of the emission covariance matrices. The Fig. 7(b) and Fig. 7(c) shows the coverage of the learned models after the user data is forgotten, as the user data for the unsuccessful demonstrations is un-purpose different than successful sampled trajectories and skews the results. The figures show that the adaptive sampling is faster in increasing the coverage and as a result faster at searching the state space.

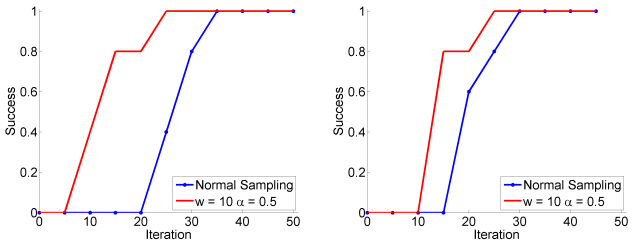
B. Robot Results

We evaluate our method on the real robot using two skills: the close the box (see Fig. 3(a)) skill, similar to the simulation case and the pour (see Fig. 3(b)) skill, where the goal is to pour uncooked macaroni from a cup to a bowl. The algorithm is run for $n_e = 10$ episodes with $n_r = 5$ iterations each. We fix the parameters of Eq. 1 to be $[\alpha = 0.5, w = 10, h = 1.75]$ based on the simulation results as these parameters resulted in a high success rate and good coverage. We discard the user data after getting $k = 10$ successful samples. In the interest of saving space, we only present the results of success rates with an initial unsuccessful model.

We teach a successful goal model using 10 demonstrations for both skills but start the self-improvement with unsuccessful action models. For close the box skill, we modify the demonstration data to result in an unsuccessful action model, similar to the simulation case. The second keyframes are pushed away by $0.03m$ in horizontal and vertical directions away from the robot and the third keyframes are pushed away from the box by $0.03m$ in the vertical direction. For the pour skill, we provide 10 bad demonstrations to the robot. The resulting unsuccessful action models has the cup 1-2 cm away from the bowl and is not tilted enough to pour the macaroni pieces. Both of the initial action models had 0% success rate. To evaluate the approach, the action models are executed 5 times after each episode. The resulting success rates are shown in Fig. 8.

For close the box skill, there are differences between the simulation and robot experiments. The robot experiment starts from a completely failed model (success rate 0%), whereas simulation experiment starts from a partially failed model (success rate 40%). Both of the robot experiment conditions reached a successful action model faster, despite the fact that they start from a worse one. There are two main reasons for this. The first is that the covariance of the robot action models are higher, which results in a larger search space. The other is that it is actually easier for the real robot to close the box, given that the robot is compliant and has soft fingers, which lets it better interact with objects.

The adaptive sampling reached a successful action model faster. The reason being that the initial action model was farther away from the successful samples than in the simulation experiment, as evidenced by the initial success rate of the model. In this case, adaptive sampling was able to sample successful executions faster than the non-adaptive case. Adaptive sampling takes better advantage of a wider variance than the non-adaptive case; the more variance the



(a) Close the box (b) Pour

Fig. 8. Real robot: The success rates for the close the box and the pour skill after each episode for 5 trials.

model has, the more the adaptive step will grow it.

The results for the pour skill is closer together (Fig. 8(b). Both the adaptive and the non-adaptive methods were able to improve the action model and did so after 3 and 4 episodes respectively. As expected, the adaptive sampling was slightly faster at improving the skill.

We also evaluate the success of the goal models since the approach depends on successful goal models The sampling and evaluation of the skill resulted in 100 iterations of monitoring for each skill. The close the box goal model had 93% recognition rate and the pour goal model had 91% recognition rate.

VI. DISCUSSION

Having a successful goal model is a pre-requisite of our self-improvement method. We have shown that such goal models, along with an initial action model, can be learned from user demonstration. We have also shown that a failed action model can be improved by using the goal models.

The method we introduced can be used with both normal and adaptive sampling. Our results suggest that the adaptive approach is more useful when finding an acceptable action model when the initial one is not successful. The difference is more apparent in the real world.

The presented method works only with a single object. Adding reference frames to each keyframe, either automatically [12] or with user interaction [13], will be necessary to handle skills with more than one object.

Similar to the IRL methods, the presented method makes use of only demonstration data to get an intermediate reward representation. This representation can be used to find a policy or even used in planning. However, the structure of the problem we are dealing with does not immediately yield itself to the existing IRL approaches. We are utilizing keyframe demonstrations which result in sparse rewards whereas IRL methods expect continuous rewards. In addition, the nature of the skills we are interested in might have binary outcomes (*e.g.* close the box). We are more interested in finding acceptable skill models that have a large space coverage, whereas the IRL methods assume the goal is to find an optimal policy. Our policy space is different than the reward space and our reward space is very high dimensional. While these issues are not insurmountable for IRL, they are not addressed by existing algorithms in the literature.

There are other avenues for improvement. Currently, the failed executions are not utilized. However, they can be used to deter the method from sampling near the failed regions. The method of forgetting the user demonstrations during self-learning is ad-hoc in this paper. Other ways, such as utilizing information measures, can provide a more elegant solution.

VII. CONCLUSIONS

We have introduced a novel approach to self-improvement of skills learned from demonstration. This approach builds on observations of naïve users being goal oriented. Goal and action models are learned using demonstration data. Then, goal models are used in a self-exploratory way to improve the action models without further user interaction.

Our results suggest that the introduced method can be used to improve an unsuccessful action model. We further show that adaptive sampling is better to increase the speed of convergence, especially for the real robot case. We provide additional evidence to support that successful goal models can be learned, which is crucial for the introduced method.

The main purpose of an LfD approach is to learn successful skill models. To the best of our knowledge, there is no LfD approach that can learn skill models for high dimensional robots (both action and sensing wise) and improve them without further user interaction, programming and/or heavy prior knowledge for object centric manipulation skills and ours is the first one.

REFERENCES

[1] B. Akgun, M. Cakmak, J. Wook Yoo, and L. A. Thomaz, "Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective," in *Intl. Conference on Human-robot interaction (HRI)*, 2012, pp. 391–398.

[2] B. Akgun and A. Thomaz, "Simultaneously learning actions and goals from demonstration," *Autonomous Robots, Online First*, 2015.

[3] B. Akgun, M. Cakmak, K. Jiang, and A. L. Thomaz, "Keyframe-based learning from demonstration," *Intl. Journal of Social Robotics*, vol. 4, no. 4, pp. 343–355, 2012.

[4] B. Argall, S. Chernova, M. M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.

[5] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The Intl. Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[6] M. P. Deisenroth, G. Neumann, J. Peters *et al.*, "A survey on policy search for robotics." *Foundations and Trends in Robotics*, vol. 2, no. 1-2, pp. 1–142, 2013.

[7] P. Abbeel and A. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the 21st Intl. Conference on Machine Learning (ICML)*, 2004, pp. 1–8.

[8] N. Ratliff, B. Ziebart, K. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. Srinivasa, "Inverse optimal heuristic control for imitation learning," in *Proc. AISTATS*, 2009, pp. 424–431.

[9] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal, "Skill learning and task outcome prediction for manipulation," in *2011 IEEE Intl. Conference on Robotics and Automation (ICRA)*, 2011.

[10] A. J. B. Trevor, S. Gedikli, R. B. Rusu, and H. I. Christensen, "Efficient organized point cloud segmentation with connected components," in *Workshop on Semantic Perception, Mapping and Exploration*, 2013.

[11] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu, "Fast 3d recognition and pose using the viewpoint feature histogram," in *IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS)*, 2010.

[12] S. Niekum, S. Osentoski, G. D. Konidaris, S. Chitta, B. Marthi, and A. G. Barto, "Learning grounded finite-state representations from unstructured demonstrations," *Intl. Journal of Robotics Research*, vol. 34, no. 2, pp. 131–157, 2015.

[13] S. Alexandrova, M. Cakmak, K. Hsiao, and L. Takayama, "Robot programming by demonstration with interactive action visualizations," in *Robotics: science and systems*, 2014, pp. 48–56.