Motion Planning for Probabilistic Representations of Robot Skills

Chris Paxton, Marin Kobilarov, Gregory D. Hager

***Abstract*— We describe an algorithm for motion planning based on expert demonstrations of a skill. In order to teach robots to perform complex object manipulation tasks that can generalize robustly to new environments, we must (1) learn a representation of the effects of a task and (2) find an optimal trajectory that will reproduce these effects in a new environment. Instead of constructing a complex cost function to optimize for each individual skill, we represent robot skills in terms of a probability distribution over features learned from multiple expert demonstrations. When utilizing a skill in a new environment, we compute feature expectations over trajectory samples in order to stochastically optimize the likelihood of a trajectory in the new environment.**

We demonstrate our algorithm on a simple Android game where users steer a needle through a sequence of gates. We also show applications to a simulated structure assembly task and with two real-world workshop assistant tasks using a UR5.

I. Introduction

Many interesting robotic tasks involve multiple steps and substantial environmental variability. Further, in a constrained environment, many steps are shared across tasks. Consider the example task of assembling a structure out of different magnetic pieces, a simple version of which is shown in Fig. 1(left). The robot must be equipped with skills to:

- successfully approach each of the pieces,
- close the gripper around each of the different pieces,
- latch pieces together,
- release and disengage from the pieces without causing damage to the structure.

We argue that successfully reproducing a task relies on (a) breaking tasks into re-usable, semantically meaningful skills, (b) learning representations of each of these skills, and (c) assembling and executing these skills in new environments. In this paper we focus on assembling and executing robot skills based on a probabilistic representation of those skills learned from demonstrations by an expert user. The problem of learning a model of a task based on such a user demonstration is referred to as imitation learning. While it has been used successfully to teach discrete movements (e.g. in [1]), teaching robots the knowledge necessary to perform complex tasks is still an open problem.

A standard approach to formulate a task is as a sequence of low-level actions, each associated with predicates, preconditions, and effects. A task T is defined as a sequence of actions A_i according to $T = \{A_i\}_{i=0}^{N_T}$ where each action is defined in terms of its specific preconditions and effects. To execute action A_i, the effects of $A_1, \ldots, A_{i-1}$ must have resulted in the preconditions of that action. Symbolic planners dating back to STRIPS [2] have used such formalisms.These high-level descriptions of tasks allow us to reason about how to complete complex tasks, as seen in However, engineering a system that allows a robot to use this sort of high-level knowledge is very challenging, and inevitably will require placing specialized domain knowledge that must be constantly updated.

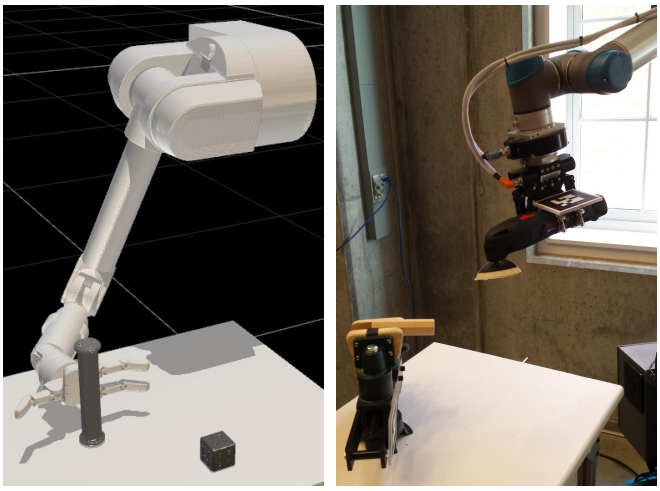


Fig. 1: Simulated WAM arm performing an `APPROACH` action using the proposed method (left), and UR5 lifting a sander as a part of a learned task (right).

Instead, it would be advantageous to ground the constraints on the performance of each action based on user demonstrations. Any such grounding should allow robust motion planning in novel environments to accomplish these low-level actions. Some previous work has attempted to solve this problem through segmentation into lower-level action models [3], [4], [5] or through learning predicate conditions and effects [6]. These approaches do not consider the difficulties inherent in reproducing tasks in complex novel environments.

We propose an approach aimed at overcoming these issues by combining a probabilistic representation of a skill learned from expert demonstrations with motion planning. Reproducing such skills in a new environment can be regarded as producing the same intended effects (represented by feature observations). However, it is difficult to determine what these intended effects are from observation. We use a probabilistic model to capture the relationships between the robot and the environment that is expected to occur during an expert

C. Paxton and G. Hager are from the Department of Computer Science, Johns Hopkins University, 3400 N. Charles St. Baltimore, MD 21218-2686, USA (email: {cpaxton3@jhu.edu, hager@cs.jhu.edu}).

M. Kobilarov is from the Department of Mechanical Engineering, Johns Hopkins University, 3400 N. Charles St. Baltimore, MD 21218-2686, USA (email: {marin@jhu.edu})

performance of a task, wherein each skill is represented as a probability distribution over expected features. These features are measurements that capture the intended effects of a skill: these may be changes in the relationships between objects in the environment or between the robot and its environment, for example.

The contributions of this paper can be summarized as:

- derivation of an algorithm for planning using probabilistic models of expert skills,
- preliminary approach for combining these learned skills for executing complex tasks,
- experimental validation of this algorithm in a number of case studies.

II. Related Work

We seek to enable combined task- and motion-planning for complex, multi-stage tasks based on learned representations of those tasks. Prior work exists in describing the relationship between high- and low-level actions and in learning representations of actions from demonstration, but does not combine learning with low-level motion planning.

Krüger et al. provide a formal description of object-action complexes [7], which describe behaviors in relation both to objects and their intended effects. However, there still exists much work to do when it comes to coming up with task plans involving a sequence of movements. These object-action complexes can be associated with learned low-level actions, and segmented based on predicates [8]. We likewise use predicates to provide the segmentation used to learn our models of low-level skills, but we do not use them during execution.

Gaussian Mixture Regression was first used in [9] to recover expected end effector and joint positions plus object distances over time, and then compute a path based on this information. Dynamic Movement Primitives (DMPs) are another imitation learning method that has proven useful for modeling low-level actions. DMPs model motions as a set of dynamical systems [10]. Pastor et al. used reinforcement learning with DMPs and multiple human demonstrations to learn a model of expected features when executing two robotic tasks in [1]: shooting pool and flipping over a box with a pair of chopsticks. Kormushev et al. used a modified version of DMPs together with reinforcement learning to adapt to new environments with a known goal [11].

There are a number of approaches for learning action primitives that can be adapted to new environments. Guenter et al. used reinforcement learning to adapt a dynamical systems model of a task to a new environment [12]. A potential field term can be added to Dynamic Movement Primitives to avoid local obstacles [13]

Otherwise, DMPs are often adapted to new environments through reinforcement learning approaches such as Path Integral Policy Improvement [1]. Work by Kober et al. uses reinforcement to adapt to new situations by adapting latent variables [14]. These methods for reinforcement learning were further expanded upon by Stulp et al., who proposed Path Integral Policy Improvement with Covariance Matrix Adaptation [15]. These techniques are also closely related to the Cross-Entropy Method for motion planning [16]. One of the major differences between the aforementioned reinforcement learning techniques and the algorithm proposed in this paper is that we do not know the correct goal state for our system. Our algorithm finds both an approximately optimal goal state and a plan for arriving at this state, expanding on the motion planning approach in [16].

In [3], [4], [5] low-level action models were learned concurrently with a task model. Grollman et al. [3] learn a finite state machine task model as an infinite mixture of Gaussian experts,which is then applied to a robot soccer task. In [4] the authors performed a similar task, using an HDP-HMM to prevent perceptual aliasing by modeling time dependencies between actions. Niekum et al. [5] used Beta Process Autoregressor Hidden Markov Models (BP-AR-HMMs) to learn tasks given unstructured expert demonstrations for a number of object manipulation tasks. After segmenting using the BP-AR-HMM, they learned DMPs representing different action primitives. In this paper, we do not use such an approach to automatically segment our data, instead using predicates as in [8]. However, these papers do not perform any motion planning, which limits their ability to adapt to new environments.

In [6] the authors learn *pull* and *push* actions and associate them with changes in the state of the world in the form of PDDL predicates. Wächter et al. reproduce a sequence of actions from a human demonstration by mapping observations onto a library of Object-Action Complexes associated with preconditions and effects [8]. These works do not examine representing and executing robot motions, but are a part of a broader framework.

III. Algorithm

Each action A in a given task is encoded as a probability distribution over a set of features that will be produced during a successful instantiation of a skill in a new environment. The features are denoted by $x \in \mathbb{R}^n$ and defined using the function ϕ through relationship

$$x = \phi(t, s, u),$$

where $t \in [t_0, t_f]$ denotes time, $s \in \mathbb{R}^d$ is the robot state, and $u \in \mathbb{R}^m$ are the applied control inputs. The model associated to each action A is denoted by $p_D(x|A)$ and is computed using unsupervised learning from expert demonstrations, typically assuming a parametric density p_D. A joint model of a task T consisting of multiple actions can be constructed using a density $p_D(x|T) \propto p_D(x|A_0) \cdots p_D(x|A_{n_T})$ assuming conditional independence between actions.

We then formulate the generalization of such a learned skill to a new environment as an optimization problem. This is accomplished by parameterizing trajectories using a parameter $\xi \in \mathcal{Z}$, where $\mathcal{Z}$ represents the space of all possible parameters resulting into valid trajectories. Since robot perception and motion are uncertain, each parameter will induces a density $p(\tau|\xi)$ where

$$\tau = \{t_0, s_0, u_0, t_1, s_1, u_1, \ldots, t_N, s_N, u_N\}$$

denotes the system trajectory. For instance, ξ would typically define a reference trajectory and an associated tracking control law resulting in the density

$$p(\tau|\xi) = p(s_0) \prod_{i=0}^{N-1} p(s_{i+1}|s_i, u_i)p(u_i|s_i, \xi).$$

In practice, given ξ the trajectory τ will either be sampled using a high-fidelity simulator or generated by the real robot.

The aim of employing an optimization-based approach is to produce a feasible trajectory which optimally approximates the effects of an expert's action, represented as a set of features capturing object-actor and object-object relationships in the workspace. Therefore, the optimization cost is encoded as the likelihood of generating new features x given the expert-derived probability distribution for the action, i.e. with respect to the constructed model $p_D(x|A)$.

We propose to employ stochastic trajectory optimization to solve the motion planning problems in constrained environments [17], [16]. This is accomplished by introducing an artificial *surrogate* distribution over $\mathcal{Z}$ that will induce a distribution over trajectories τ and over the corresponding features x along these trajectories. The surrogate will then be iteratively optimized until it becomes optimally close (in a distribution sense) to the expert density $p_D(x)$ without violating the constraints of the environment such as obstacles and joint limits. The surrogate model is built using a parametric density $\pi(\xi|v)$ such as a multivariate Gaussian or a GMM with parameters v.

More specifically, the cost is defined as the Kullback-Leibler divergence between the probability distribution of expected features and the probability of observing those new features from dynamically feasible robot states and is given by:

$$J(v) = \sum_{i=0}^{N} D_{\text{KL}}(p_D(x_i)||p(x_i|v)),$$

using the notation

$$D_{\text{KL}}(p_D(x_i)||p(x_i|v)) \triangleq \int p_D(x_i) \frac{\log p_D(x_i)}{\log p(x_i|v)} dx_i, \quad (1)$$

and $p(x_i|v) = \int I_{\{x_i=\phi(t_i,s_i,u_i)\}} p(\tau|\xi)\pi(\xi|v)d\tau d\xi$ being the density of a feature at time t_i from the surrogate model.

The optimal parameter v^* can be obtained by noting that

$$\min_v \sum_{i=0}^{N} D_{\text{KL}}(p_D(x_i)||p(x_i|v)), \quad (2)$$

$$= \min_v \sum_{i=0}^{N} \int -p_D(x_i) \log p(x_i|v) dx_i, \quad (3)$$

$$\approx \min_v \sum_{i=0}^{N} \sum_{j=1}^{M} -\frac{p_D(x_{i,j})}{p(x_{i,j}|v_0)} \log p(x_{i,j}|v), \quad (4)$$

$$\approx \min_v \sum_{i=0}^{N} \sum_{j=1}^{M} -p_D(x_{i,j}) \log \pi(\xi_j|v), \quad (5)$$

where $x_{i,j} = \phi(t_i, s_{i,j}, u_{i,j})$ is a generated feature from robot state $s_{i,j}$ at time t_i along the sampled trajectory $\tau_j \sim p(\cdot|\xi_j)$ for $\xi_j \sim \pi(\cdot|v_0)$. More specifically, step (4) corresponds to a Monte Carlo approximation of the integral using samples τ_j, ξ_j from a nominal density $p(\tau|\xi)\pi(\xi|v_0)$. The denominator in (4) corresponds to the change of measure due to sampling from $p(x_i|v_0)$ instead of $p_D(x_i)$. In step (5) the approximation

$$p(x_{i,j}|v) \approx \sum_{j=1}^{M} I_{\{x_{i,j}=\phi(t_i,s_{i,j},u_{i,j})\}} \frac{\pi(\xi_j|v)}{\pi(\xi_j|v_0)} \approx \frac{\pi(\xi_j|v)}{\pi(\xi_j|v_0)}$$

was employed. Note that due to the reuse of samples to approximate $p(x_i|v)$, the estimate (5) will be biased in the sense that the integration measure corresponding to v_0 is not compensated for.

The necessary conditions for a minimum correspond to setting the gradient of (5) to zero, i.e. by solving the equality

$$\sum_{i=0}^{N} \sum_{j=1}^{M} -w_{i,j} \nabla_v \log \pi(\xi_j|v) = 0, \quad (6)$$

where the weights $w_{i,j}$ are given by $w_{i,j} \triangleq p_D(x_{i,j})$. When $\pi(\cdot|v) = \mathcal{N}(\cdot|\mu, \Sigma)|_{\mathcal{Z}}$ (i.e. a single multivariate Guassian with domain restricted to feasible parameter set $\mathcal{Z}$), the relationship (6) can be solved in closed form as

$$\mu = \sum_{j=1}^{M} \bar{w}_j \xi_j, \quad \Sigma = \sum_{j=1}^{M} \bar{w}_j (\mu - \xi_j)(\mu - \xi_j)^T, \quad (7)$$

where $w_j = \sum_{i=0}^{N} w_{i,j}$ and $\bar{w}_j = w_j / \sum_{j=1}^{M} w_j$. When $\pi(\cdot|v)$ is a GMM the minimization (5) is performed using a weighted expectation minimization (EM) algorithm.

In practice, the optimal parameter v is computed iteratively starting with some nominal choice v_0 which approximately covers the trajectory space of interest. At each iteration we draw M samples $\xi_j \sim \pi(\cdot|v_0), j \in 1, \ldots, M$ and compute the next v by minimizing (5). At the next iteration v_0 is reset to v and the process continues until the cost converges.

A. Executing a Task

When attempting to complete a complex task, however, the approach outlined above is not sufficient. Optimizing individual actions may leave the robot in a situation where it cannot perform a necessary next step correctly, an issue we examine further in Sec. V-B.

One approach is to employ a joint model $p_D(x|T)$ over the whole task in place of $p_D(x|A)$. In such case it is expected that the parameter ξ must encodes a more complex trajectory which for instance could have discrete or binary variables corresponding to switching states (such as "close-gripper" or "open-gripper"). As the dimension and complexity of $\mathcal{Z}$ grows, solving the optimal control problem could become intractable.

Instead, we employ a simpler approach which still optimizes a trajectory τ over a single action A_K with the exception that it also ensures smooth transition to the next action A_{k+1}. This is achieved by maximizing the likelihood of the final feature x_N under action A_{k+1}. To this end we

Algorithm 1 Algorithm for optimal reproduction of expert demonstration in new environments.

Given: step size α, initial trajectory distribution v_0
for $i \in N_{iter}$ **do**
  Draw M samples $\xi_j \sim \pi(\cdot|v_0)$
  Simulate trajectories $\tau_j \sim p(\cdot|\xi_j)$
  Compute all $w_{i,j}$ via Eq. (8)
  Compute v_{i+1} via Eq. (9)
  $v_0 \leftarrow v_i$
end for

employ $p_D(x_i|A_k)p_D(x_N|A_{k+1})$ in place of $p_D(x_i|A)$ in (5) leading to the new weights

$$w_{ij} \triangleq p_D(x_{i,j}|A_k)p_D(x_{N,j}|A_{k+1}), \tag{8}$$

where $p_D(x|A_i)$ is the expert density of action A_i.

The general form of the planning algorithm is described above in Algorithm 1. We further describe extensions to this basic algorithm to deal with obstacles and to combine different states.

B. Avoiding Obstacles and Joint Limits

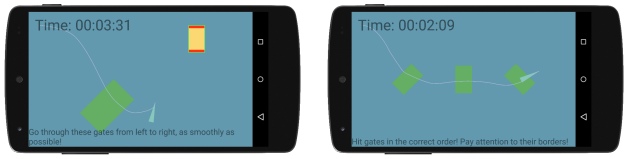
The cross-entropy method provides a straightforward way of dealing with obstacles and joint limits that may prevent us from reproducing a task in a new environment. Rather than using potential fields for object avoidance as per [13], we constrain $\mathcal{Z}$ to consist only of the space of valid trajectories. This means that when drawing our M samples, we remove samples currently in collision or past joint limits in our new environment and continue to draw sample trajectories until we have all M valid examples. This allows us to accomplish both motion planning and skill imitation through the same framework. This works effectively in practice as long as the task does not require generalization in environments with very narrow passages that the system has never been trained on. Such cases are extremely difficult since the probability of obtaining samples in the narrow passage is close to zero, unless an informative nominal density parameter v_0 is used with enough probability mass over such regions.

C. Controlling Step Size

In order to ensure we arrive at an optimal solution, we want to control the size of the steps we take during trajectory optimization. To this end we introduce an extra parameter $0 < \alpha < 1$, which controls the size of steps taken at each iteration. Given Σ_i^* as the optimal Σ at iteration i, we compute μ_{i+1} and Σ_{i+1} as:

$$\begin{aligned} \mu_{i+1} &= (1-\alpha)\mu_i - \alpha\mu_i^* \\ \Sigma_{i+1} &= (1-\alpha)\Sigma_i - \alpha\Sigma_i^* \end{aligned} \tag{9}$$

Since Σ determines our search region, introducing this step size term lets us prevent our search from converging prematurely. In practice we found setting this value to $0.50 - 0.75$ gave us the best results.



(a) *Needle Master* Level 4. (b) *Needle Master* Level 5.

Fig. 2: Examples of the more complex environments that the player must respond to in the *Needle Master* android game.

D. Implementation Details

We represent p_D as a Gaussian Mixture Model with K clusters. Using mixture models such as GMMs is suitable for providing multiple options when executing a task, e.g. for a pick-and-place task each Gaussian can essentially encode a possible approach for a grasp. In practice, we can often use as few as one GMM component to represent simple skills such as the correct approach to grasp an object. The features we selected are analogous to those used in common imitation learning approaches, e.g. [9], [1].

This GMM also encodes unknown controls (ones that we cannot reliably motion plan for, or for which motion planning would be overly expensive). For example, in the simulation case study in Sec. V we use Gaussian Mixture Regression to recover the gripper controls g at different points in a grasping skill based on current features x.

The procedure is given in Algorithm 1.

IV. Case Study: Needle Master

Exploring these ideas requires reliable perception that can extract meaningful information about of the world. We initially explored our method in simulation environments with "perfect" sensing capabilities. To this end, we developed an Android game to explore how user actions relate to different features of the environment and to allow us to test different approaches for modeling action primitives. This game, called "*Needle Master*", is inspired by the suturing task common in robotic minimally invasive surgery.

In this game, users need to steer a 2D needle through a number of target gates positioned in patient tissue, as shown in Fig. 2. These gates represent needle insertion points in a suturing task. An example of a *Needle Master* level with two gates is shown in Figure 2a. We aim to learn the way users respond to variations in the features of the environment, particularly in the position and orientation of these gates. Maintaining the correct orientation of the needle relative to gates and tissue is very important, as is making a path that does not go into high-risk "tissue" regions.

Users can rotate the needle and control its velocity. The rules of the game are:

- The needle must exit off of the right side of the screen.
- The needle must pass through gates in order. Hitting the top or bottom of the gate will "break" that gate, causing the user to score less points for passing through it.
- The needle must not hit dark red "deep tissue", representing vital organs.

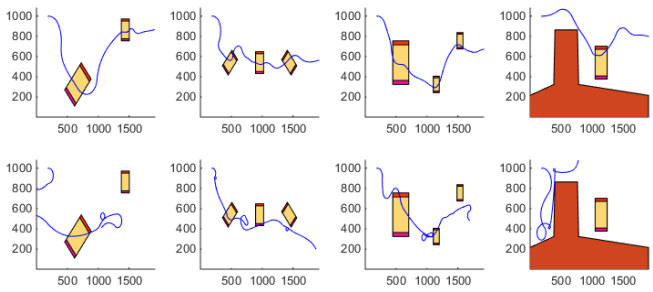


Fig. 3: Comparison showing our method on Needle Master levels $4-7$ (top) vs. a version of the model without goal term introduced in Eq. (8) (bottom).

- The needle must not cause too much damage to pink "tissue". Players can damage this tissue if they rotate the needle too much.

Fig. 2 shows a variety of complex environments users must respond to in the *Needle Master* game. Environments can have many different gates the player should pass through. Later on, we add additional obstacles for players to respond to, such as *tissue*. Players cannot rotate the needle as much in tissue, and if they do, they will damage it. Causing too much tissue damage causes them to lose the game.

This game is available for Android smart phones and tablets, and can be downloaded for free from the Google Play Store[1]. Player data can be exported through the application to files that are saved on the phone's internal storage; data is not collected without users opting in.

A. Task Representation

The Needle Master task is to move from left to right, hitting gates in a specific order if possible while attempting to avoid the dark red areas on the top and bottom of the gate. The needle in this game has a state defined $s = [\mathtt{x}, \mathtt{y}, \theta]^\mathtt{T}$, where $\mathtt{x}$ and $\mathtt{y}$ denote the position of the needle and θ represents its orientation. It is associated with an action $u = [\mathtt{u}, \mathtt{v}]$. The dynamics of the needle are given by:

$$\begin{bmatrix} \mathtt{x_{i+1}} \\ \mathtt{y_{i+1}} \\ \theta_{i+1} \end{bmatrix} = \begin{bmatrix} \mathtt{x} + \mathtt{v}\cos(\theta) \\ \mathtt{y} + \mathtt{v}\sin(\theta) \\ \theta + \mathtt{u} \end{bmatrix} \tag{10}$$

We can represent a trajectory produced by these dynamics as a series of curves with constant rotation $\mathbf{u}$ and constant velocity $\mathbf{v}$, as described in [16].

We segment task demonstrations based on a set of predicates. These predicates describe the relationship of the needle to its environment, as a proxy for what the expert demonstrating the trajectory is responding to. Predicates used in these examples are:

- `NEEDLE-IN-GATE`: true if the needle is within a gate
- `HAS-PREV-GATE`: true if we are leaving one gate and still have gates to visit
- `GATE-CLOSED`: true if a gate has been closed
- `GATE-OPEN`: true if a gate is open still
- `AT-EXIT`: true if the needle has moved off the screen to the right, thus ending the level

These predicates divide user demonstrations into five different states, parameterized by different entities in the environment: (1) approaching the first gate, (2) passing through a gate, (3) connecting two gates, (4) moving to the exit, and (5) at the exit and done with a level. There are features associated with each entity in the environment. Different actions use whichever features are appropriate: the action for connecting two gates uses the features associated with both gates, for example, while passing through a gate is based on the features associated with that gate alone.

Each gate is defined by its position, height, width, and its orientation. Top and bottom bars are a constant fraction of the gate's height. The features for each gate project the needle's current position into the gate's frame of reference, and compare the distance between the angle of the needle and the angle of the gate:

$$\phi^{gate}(s) = \begin{bmatrix} \cos\left(\text{atan2}\left(\mathtt{y}, \mathtt{x}\right) + \theta^{gate}\right) - \mathtt{x}^{\mathtt{gate}} \\ \sin\left(\text{atan2}\left(\mathtt{y}, \mathtt{x}\right) + \theta^{gate}\right) - \mathtt{y}^{\mathtt{gate}} \\ \sqrt{(\mathtt{x} - \mathtt{x}^{\mathtt{gate}})^2 + (\mathtt{y} - \mathtt{y}^{\mathtt{gate}})^2} \\ |\theta^{gate} - \theta| \end{bmatrix} \tag{11}$$

We use the y distance to the edge of the level as a feature for skills learned after the `GATE-CLOSED` predicate is true for all gates, and we use the rotation control $\mathtt{u}$ as a feature for all actions. Finally, we use the change in the above variables as an additional set of features.

We used GMMs with $k = 3$ components to model skills and fit them through the Expectation-Maximization algorithm. These were learned from a total of 44 expert demonstrations from the first six levels of the game. Our approach allows us to complete very complex Needle Master levels, as shown in Fig. 3. In particular, our approach can avoid challenging obstacles such as that shown in Fig. 3(far right), and it can also chain together the necessary actions to complete each level.

B. Comparison To Other Methods

To validate our approach, we showed how our algorithm could generate trajectories for 10 different randomly generated levels with multiple gates and obstacles. We compare these results to two other approaches: (1) a naive version of the algorithm without performing planning, and (2) a version of the algorithm that does include the goal term discussed in Eq. (8).

All three versions of the algorithm rely on the same GMM learned from user data. In the "Naive" version of the algorithm, at each time step we take the action corresponding to the highest probability by searching for the optimal $(\mathbf{u}, \mathbf{v})$ given the current features x.

We compare five different metrics: (1) how many gates were ever entered by the needle, (2) of those, how many gates were "cleared" successfully, (3) how many gates were "broken" by the needle either touching the top and bottom bars, (4) how many levels ended by touching an obstacle,and

[1] `https://play.google.com/store/apps/details?id=edu.jhu.lcsr.needlemaster`

With Obstacles	Gates Entered	Gates Cleared	Gates Broken	Levels Finished
Naive Approach	3	1	2	2
No Goal Term	13	7	6	2
Full Algorithm	20	16	4	20
No Obstacles	**Gates Entered**	**Gates Cleared**	**Gates Broken**	**Levels Finished**
Naive Approach	1	0	1	1
No Goal Term	12	6	6	2
Full Algorithm	20	15	5	20

TABLE I: Comparison of performance on the Needle Master task with different methods implemented.

how many levels were successfully finished (final position off the screen to the right). In one set of experiments (top), we generated 10 levels with two gates and two obstacles. In the second (bottom) set, we generated 10 levels with two gates and no obstacles.

Table I shows the results of these experiments. The naive approach performed very poorly, usually because it has trouble aligning with the gates. When gates are in unexpected positions, it choses high-probability movements that end up moving it away from the correct path. The approach without a goal term similarly performs poorly: the behavior of each component action is not well defined at the beginning and end.

V. Case Study: Structure Assembly

Our algorithm is demonstrated in an object manipulation task where we need to build a structure of increasing complexity out of magnetic blocks, as per the task described in [18].

Our simulated assembly task consists of a number of skills which need to be executed in the same order to connect two pieces together. In the structure assembly task, we are learning the actions:

- `APPROACH(link)`: move to a "link" object
- `GRASP(link)`: close the gripper and secure the link object.
- `ALIGN(link,node)`: holding the link, move so that it is above the node.
- `PLACE(link, node)`: place the link on top of the node.
- `RELEASE(link)`: open the gripper
- `DISENGAGE(link)`: move away from the link without knocking it over

These skills must be chained together in a specific order, and each skill is critical to the completion of the next skill. Figure 4 shows the procedure as executed by our algorithm.

When teaching the `APPROACH` and `GRASP` skills, we are teaching one particular grasp, associated with a range of valid positions. Note that the `link` object has eight-way rotational symmetry: we can flip it over or rotate in 90 degree increments and end up with a valid grasp. We can also mirror our grasp, putting two fingers on either side of the object. The same is true of the `ALIGN` and `PLACE` actions for connecting the link and the node: there are four possible mates for each of the cube's six surfaces. We are teaching one specific mate and grasp in this example, but we could use these symmetries to apply our grasp and mate to any of the applicable surfaces on our two objects.

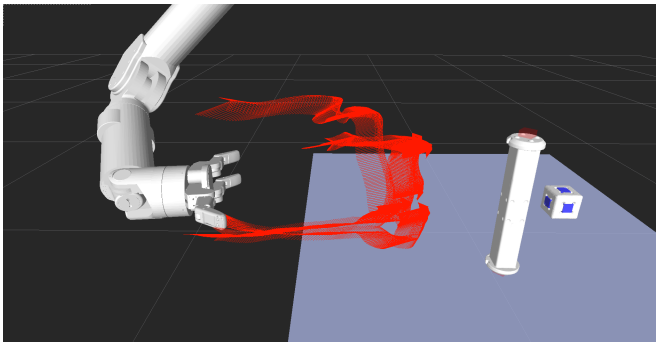


Fig. 5: Four trajectories showing user demonstrations used to train the `APPROACH` skill for the simulated Barrett WAM arm task. Note that these trajectories capture a very wide range of possible approaches, being of different lengths and in different regions.

To create a model of each of these skills, we collect four demonstrations of the object manipulation action starting from the same grasp, with two of the Barrett Hand's fingers on the left side of the link and one on the right. The simulated WAM arm was teleoperated: users controlled the end effector frame of the robot, and joint torques were computed by a Jacobian Transpose Null Space controller. As a result training data was very noisy, as we might expect from similar non-expert instruction of a real robot. Fig. 5 shows the trajectories used to train one of these skill models, that of the `APPROACH` action. Trajectories are shown in red, relative to the `link` object that parameterizes the skill.

A. Task Representation

We use four sets of features: (1) the time in a particular state, (2) the gripper command variables, (3) the transform between the end frame and the "link" object, and (4) the transform between the end frame and the "node" object. We used the `x`,`y`, and `z` offsets from this transform, the distance (norm of the position), and the quaternion representation of the rotation as features in the transform.

The feature function $\phi(s)$ computes the forward kinematics of the robot arm: our variables are the relative end effector position in comparison to objects in the environment and the norm of the position. For our experiments, we selected the relevant objects in each demonstration for which we compute the features. For the `APPROACH`, `GRASP`, `RELEASE`, and `DISENGAGE` actions, these are computed based on the `link` object alone. For the `ALIGN` and `PLACE` actions, we compute parameters based on the `node` object.

We represent a robot trajectory as a set of end-effector poses to attain. In the WAM arm simulation, we use three such poses, and interpolate between them to create trajectories. Each individual primitive consists of an $(\texttt{x}, \texttt{y}, \texttt{z}, \texttt{roll}, \texttt{pitch}, \texttt{yaw})$ offset from the current end effector location. We interpolate between these points to create a trajectory.

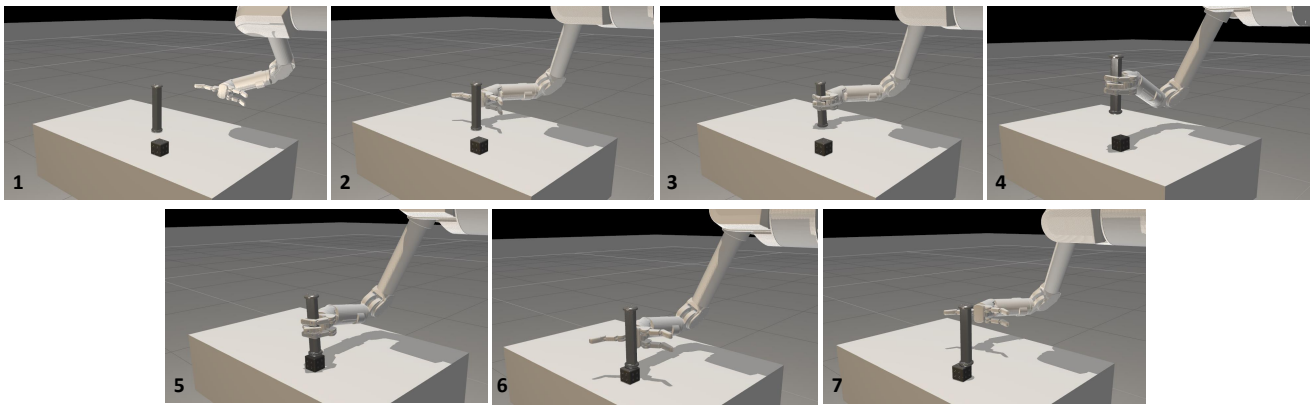


Fig. 4: Sequence of actions necessary to perform the structure assembly task. Starting in the upper left: initial state, `APPROACH(link)`, `GRASP(link)`, `ALIGN(link,node)`, `PLACE(link,node)`, `RELEASE(link)`, and `DISENGAGE(link)`.

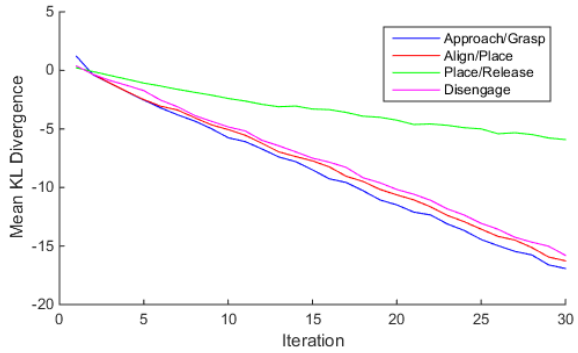


Fig. 6: Plot showing improvement of mean KL divergence over 30 iterations of the algorithm with $\alpha = 0.75$.

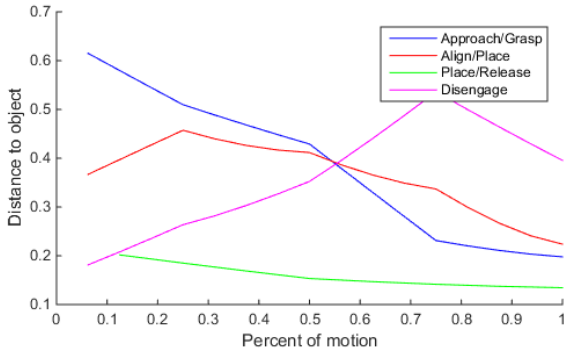


Fig. 7: Plot showing distance between the end frame and relevant objects for learned skills. For `APPROACH` and `DISENGAGE`, this object is the link. For `ALIGN` and `PLACE` this is the node.

B. Results

We collected three demonstrations of each of the different skills with a dynamic simulation of the Barrett WAM arm. We then place these pieces in different positions in the environment, and validated our method by performing the task in different locations. The results of one version of the algorithm are shown in Fig. 4.

In particular, we find that as in the Needle Master example discussed above, it is very important to have good models of the next skill in order to complete each previous skill: motion in the `PLACE` and `APPROACH` actions are often very noisy, but successful grasps mates (as exemplified by the `GRASP` and `RELEASE` skills) are much more restrictive.

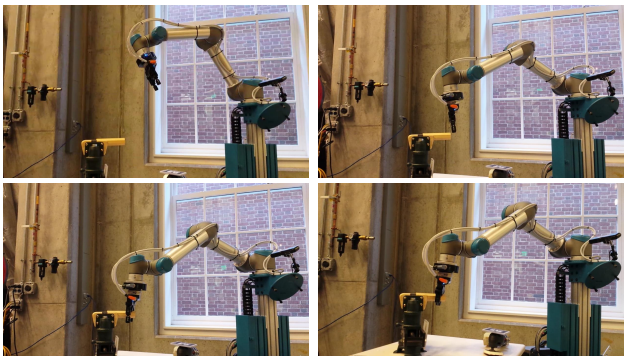
Actions such as `APPROACH` and `ALIGN` are very vaguely defined actions, as well. An example of an `ALIGN` action is shown in Fig. 5. The training data used to create these skills are extremely noisy, due to the wide variety of different approach directions and distances from which a user can move. One advantage of our method, therefore, is that these actions become better defined when combined with a following action, like the `GRASP` or `PLACE`.

Fig. 6 shows how our approach iteratively decreases the KL divergence between the features produced by the current trajectory distribution parameterized by v_i and the expert probability distribution. Each line indicates the average KL divergence during iteration for one action (either `APPROACH`, `ALIGN`, `PLACE`, or `DISENGAGE`) and its goal. The distribution for the action `ALIGN` with the goal of `PLACE` is far less constrained, since there are many ways of approaching the region above the node before a mate.

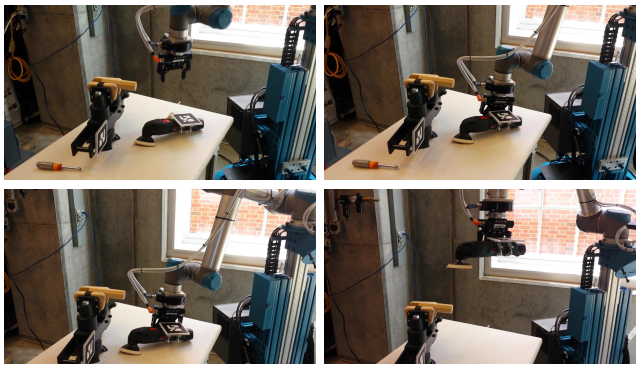
Fig. 7 explores how the distance between various objects changes over the course of these trajectories. We can see that during the `ALIGN` action, distance does not change very much, partly because much of the action is lifting the link up so that it can be mated to the top. Contrast this with the `APPROACH` action, where the end effector quickly approaches the link and shifts into a high-quality position to transition to `GRASP`.

VI. Case Study: Workshop Assistant

We applied the algorithm described above to training and using an intelligent workshop assistant. This assistant should be able to be easily taught new skills by a non-programmer, but also able to intelligently apply those skills



(a)



(b)

Fig. 8: Workshop assistant experiments. Fig. 8a shows the UR5 performing a grasp action via our algorithm. The goal is to grasp the wooden block held in the vise. Fig. 8b shows the first portion of a sanding task: the robot grabs a sander and lifts it up over the vise.

in new contexts. Motions and features are represented in exactly the same way as described in Sec. V.

We used Alvar, an open-source AR tracker compatible with ROS, to implement the demo by providing locations of a handheld sander with attached tool collar and a vise. This tool collar is designed to allow the UR5 to hold a workshop tool with a Robotiq 2-finger gripper, as shown in Fig. 8b. The gripper gives us a position tolerance of about 0.5 cm to successfully grab the tool. We the kinesthetic teach mode of the UR5 to provide two to three demonstrations for each component skill.

We demonstrate two skills in Fig. 8. In Fig. 8a we show the robot grabbing a wooden block held in the vise, and in Fig. 8b we show the robot picking up a sander and lifting it over the vise. In general these later actions need not be demonstrated on the robot itself, since we are capturing the motion in each frame of reference in our model, not changes in the robot's joint angles.

VII. Conclusions

We described a practical approach for motion planning based on a probabilistic model of a skill. One of the goals of this approach is to enable task-level planning: by representing skills as probability distributions, we create a framework that allows us to describe and combine a broad range of cost functions. Future work will integrate this approach with higher-level task planning. In addition, we will examine automatically segmenting demonstrations of tasks into sub-skills and identifying useful invariant features.

References

[1] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal, "Skill learning and task outcome prediction for manipulation," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 3828–3834.

[2] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3, pp. 189–208, 1972.

[3] D. H. Grollman and O. C. Jenkins, "Incremental learning of subtasks from unsegmented demonstration." in *Intelligent Robots and Systems (IROS 2010), 2010 IEEE/RSJ International Conference on*, 2010, pp. 261–266.

[4] J. Butterfield, S. Osentoski, G. Jay, and O. C. Jenkins, "Learning from demonstration using a multi-valued function regressor for time-series data," in *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*. IEEE, 2010, pp. 328–333.

[5] S. Niekum, S. Osentoski, G. Konidaris, and A. G. Barto, "Learning and generalization of complex tasks from unstructured demonstrations," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5239–5246.

[6] S. R. Ahmadzadeh, A. Paikan, F. Mastrogiovanni, L. Natale, P. Kormushev, and D. G. Caldwell, "Learning symbolic representations of actions from human demonstrations," 2014.

[7] N. Krüger, J. Piater, F. Wörgötter, C. Geib, R. Petrick, M. Steedman, A. Ude, T. Asfour, D. Kraft, D. Omrcen, *et al.*, "A formal definition of object-action complexes and examples at different levels of the processing hierarchy," *PACO-PLUS Technical Report, available fro m http://www. paco-plus. org*, 2009.

[8] M. Wachter, S. Schulz, T. Asfour, E. Aksoy, F. Worgotter, and R. Dillmann, "Action sequence reproduction based on automatic segmentation and object-action complexes," in *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on*. IEEE, 2013, pp. 189–195.

[9] S. Calinon, F. Guenter, and A. Billard, "On learning, representing, and generalizing a task in a humanoid robot," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 37, no. 2, pp. 286–298, 2007.

[10] S. Schaal, "Dynamic movement primitives-a framework for motor control in humans and humanoid robotics," in *Adaptive Motion of Animals and Machines*. Springer, 2006, pp. 261–280.

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

[12] F. Guenter, M. Hersch, S. Calinon, and A. Billard, "Reinforcement learning for imitating constrained reaching movements," *Advanced Robotics*, vol. 21, no. 13, pp. 1521–1544, 2007.

[13] D. Park, H. Hoffmann, P. Pastor, and S. Schaal, "Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields," in *8th IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2008, pp. 91–98.

[14] J. Kober, E. Oztop, and J. Peters, "Reinforcement learning to adjust robot movements to new situations," in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, no. 3, 2011, p. 2650.

[15] F. Stulp and O. Sigaud, "Path integral policy improvement with covariance matrix adaptation," *arXiv preprint arXiv:1206.4621*, 2012.

[16] M. Kobilarov, "Cross-entropy motion planning," *International Journal of Robotics Research*, vol. 31, no. 7, pp. 855–871, 2012.

[17] R. Y. Rubinstein and D. P. Kroese, *The cross-entropy method: a unified approach to combinatorial optimization*. Springer, 2004.

[18] J. Bohren, C. Papazov, D. Burschka, K. Krieger, S. Parusel, S. Haddadin, W. L. Shepherdson, G. D. Hager, and L. L. Whitcomb, "A pilot study in vision-based augmented telemanipulation for remote assembly over high-latency networks," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 3631–3638.