Coupled Learning of Action Parameters and Forward Models for Manipulation

Sebastian Höfer Oliver Brock

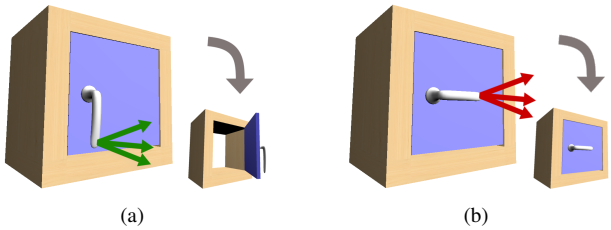
***Abstract*— The effectiveness of robot interaction depends on the robot's ability to perform task-relevant actions and on the degree to which it is able to predict the outcomes of these actions. In this paper we argue that the two learning problems – learning actions and learning forward models – must be tightly coupled for each of them to be successful. We present an approach that is able to learn a set of continuous action parameters and relational forward models from the robot's own experience. We formalize our approach as simultaneously clustering experiences in a continuous and a relational representation. Our experiments in a simulated manipulation experiment show that this form of coupled subsymbolic and symbolic learning is required for the robot to acquire task-relevant action capabilities.**

I. Introduction

In order to act autonomously in unknown environments, robots must be able to learn novel action capabilities from their own experience. To be effective, an action capability must fulfill two requirements: the robot must know how to physically instantiate the action (i.e. know its *parameters*), and the robot must have the ability to predict the possible effects of this action, given the state of the world (i.e. have a *forward model*). The latter is required to devise action plans and the former to execute them.

Learning action parameters and forward models for robotics are both difficult problems, and they are usually approached independently [1]–[3]. But the relevance of an action is tightly coupled to its forward model – and vice versa: Given a forward model that predicts effects of an action, the model is only valid if the underlying action parametrization reliably evokes the predicted effects. Conversely, an action is only relevant if the robot can predict its effects with high certainty. Thus, the two learning problems are intrinsically coupled and should be solved jointly.

This insight motivates our approach of **c**oupled **a**ction **p**arameter and **e**ffect **l**earning (CAPEL, pronounced "couple"). CAPEL gets as input a set of experiences, and learns actions and their corresponding forward models. We represent actions as *continuous vectors of parameters* which encode *low-level* instantiations, and forward models as *symbolic, relational* sets of rules, predicting *high-level*, abstract effects [4]. Relational representations do not attempt to represent the world in full detail, but equip the robot with a task-relevant abstraction of the world state. This type of abstractions is particularly well-suited for manipulation tasks and for efficiently planning sequences of actions [5]. Therefore, CAPEL not only couples action and forward model learning, but also learns across two different representations, low-level action parameters and high-level forward models.



(a) (b)

Fig. 1. CAPEL enables the robot to learn action capabilities by taking into account both the similarity of actions as well as their effects: (a) The robot can successfully open the door if it pulls the handle in a direction roughly normal to the door plane (green arrows). (b) The same action fails when the door is locked.

We implement this idea by formalizing CAPEL as an iterative clustering method that groups experiences in both representations in a coupled fashion. A cluster in the continuous representation then corresponds to an action, and a cluster in relational representation to a forward model.

To demonstrate the effectiveness of CAPEL, we apply it in a manipulation experiment that requires manipulating articulated objects in a simulated environment. As the focus of this paper lies on studying how coupling action and forward models learning improves learning, and not on learning complex action skills, we confine ourselves to a low-dimensional parametrization of actions in terms of pushing and pulling directions. Our results show that CAPEL enables the robot to reliably plan sequences of actions to operate the articulated objects, overcoming problems that arise when treating the learning problems separately. This indicates that *combining subsymbolic and symbolic learning* is an effective paradigm for learning action capabilities, a paradigm that has only recently been gaining attention [6], [7], and that should be pursued further in the future.

II. Related Work

In previous work, the term *action* or *skill* is usually defined as a set of parameters. In the following, we will contrast this term to an *action capability* which we define as a set of parameters *and* a corresponding forward model [8], [9]. We will now discuss previous work on learning parameters and

Both authors are with the Robotics and Biology Laboratory, Technische Universität Berlin, Germany.

We gratefully acknowledge the funding provided by the German Research Foundation (DFG, Exploration Challenge, BR 2248/3-1), and the Alexander von Humboldt foundation through an Alexander von Humboldt professorship (funded by the German Federal Ministry of Education and Research). We would like to thank W. Böhmer, R. Jonschkowski, L. Kaelbling, R. Lieck, T. Lozano-Pérez and R. Martín-Martín for advice and fruitful discussions.

forward models, first covering approaches that address them independently, and then approaches that combine them.

Learning action parameters: There has been remarkable success in learning action skills for complex robot motion [1], [2], [10], [11]. These approaches make the learning problem tractable by training a skill for producing only *a single effect*. CAPEL allows to learn multiple skills and learns how to sequence them appropriately to solve a task.

Learning forward models: To sequence skills, a robot requires a forward model that predicts the effect of each skill. Learning forward models is difficult if it is not known a priori which properties of the world are task-relevant. One way is to learn models that predict effects at the level of physics [3], [12] but these models do not easily capture abstract properties of the world. This problem can be overcome by using symbolic, relational models [4]–[7], [13], [14]. However, the applicability of symbolic models hinges on the premise that the symbols reflect meaningful entities in the world. While integrated task and motion planning [14] alleviates the problem to some extent, it still depends on suitable state symbol groundings and action templates. Thus, it seems indispensable that the robot learns how to ground these state and action symbols. Recent work has focused on learning *state symbols* [6], [7], [15]. In contrast, CAPEL relies upon pre-defined state symbols (see Sec. VII-6), but it learns task-relevant *action symbols* (and forward models for each of them).

Combined action parameter and forward model learning: There exist previous approaches to combined skill and forward models learning, but these approaches make simplifying assumptions that CAPEL removes. One assumption is to only learn *binary forward models* which predict the success or failure of actions [1], [10]. These binary models do not allow to plan sequences of actions, in contrast to the symbolic forward models learned by CAPEL. Other approaches assume that *only one action capability* at a time is learned: similar to our work, Orthey et al. [16] optimize an action such that the associated relational forward model becomes more predictive. But different from CAPEL, their approach cannot learn multiple action capabilities. Finally, one can leverage *human demonstrations* [11]. These simplify the learning problem significantly by narrowing down the state and action spaces to task-relevant regions. In contrast, CAPEL is able to learn from the robot's own and even randomly chosen actions.

To summarize, CAPEL combines low-level continuous and high-level symbolic learning to acquire a set of multiple actions capabilities from the robot's own experiences, without requiring any demonstrations.

III. Overview

We begin with an intuitive explanation of CAPEL, and we formalize in the following sections. CAPEL receives as input a set of experiences, generated by interaction of the robot with the world. In this paper, the world consists of articulated objects (cabinets with different opening mechanisms, Fig. 1) and the robot has to learn how to operate them.

Such real world tasks are difficult because different parametrizations of an action can cause the same effect, but the same action can also have different effects, depending on the state of the world. This difficulty is illustrated in Fig. 1(a): several actions (with only slightly differing parametrizations) open the door, but the same actions fail to open the door if it is locked (Fig. 1(b)). When learning from experience, the robot must understand whether different experiences result from the same or different actions – otherwise, it learns wrong parametrizations or forward models.

To cope with this issue, CAPEL exploits two priors: that *similarly parametrized actions have similar effects*, and that every action has *few and predictable effects*. We formalize the learning problem as finding a good *clustering* of experiences, and we compute a continuous action parametrization and a relational forward model for each cluster. The goal is to find a clustering that obeys the two priors. Notably, the first prior is implemented in the low-level, continuous action representation, and the second one in the high-level relational model representation. Thus, we can think of CAPEL as clustering in both representations, as illustrated by Fig. 2. However, the representations are coupled, allowing the priors to shape the actions and forward models in both representations.

IV. Coupled Continuous and Relational Reinforcement Learning

Formally, CAPEL can be cast as a reinforcement learning (RL) problem. We will now introduce this problem and explain technical prerequisites of CAPEL. The next section will focus on the novel aspects of CAPEL and explain how it solves the given RL problem.

1) Reinforcement Learning in Coupled MDPs: The robot interacts with the world by executing low-level continuous actions, but perceives the state of the world and the effects of its actions in a high-level relational representation. We use $\underline{\cdot}$ to indicate *low*-level, continuous-valued, and $\bar{\cdot}$ to indicate *high*-level relational quantities. When referring to both the continuous and relational quantity we omit the bars.

We represent the world by a pair of coupled MDPs. The first, *continuous MDP* is defined as $\langle \underline{S}, \underline{A}, \underline{T}, \underline{R}, \gamma \rangle$, with states $\underline{S}$, actions $\underline{A}$, a stochastic transition/forward model $\underline{T} : \underline{S} \times \underline{A} \times \underline{S} \to [0,1]$, a reward function $\underline{R} : \underline{S} \times \underline{A} \to \mathbb{R}$, and discount factor $\gamma \in [0,1)$. The second, *relational MDP* is defined as $\langle \bar{S}, \bar{A}, \bar{T}, \bar{R}, \gamma \rangle$. We ensure that the two MDP are consistent by deterministically mapping each continuous state $\underline{s} \in \underline{S}$ to a relational state $\bar{s} \in \bar{S}$, and by assuming equivalent reward functions ($\underline{R}(\underline{s}) = \bar{R}(\bar{s})$ for all states).

Given these MDPs, the robot's goal is to find a relational policy $\pi : \bar{S} \to \bar{A}$ that maximizes the expected discounted sum of rewards: the well-known relational RL problem [13]. CAPEL performs *model-based* RL by learning the relational forward model $\bar{T}$ (assuming $\bar{R}$ as given) and then applying a relational planner [5], but with an additional complication: the set of relational actions $\bar{A}$ is not known beforehand. Thus, CAPEL must first learn to translate each continuous action to a relational action. Before we explain how CAPEL solves

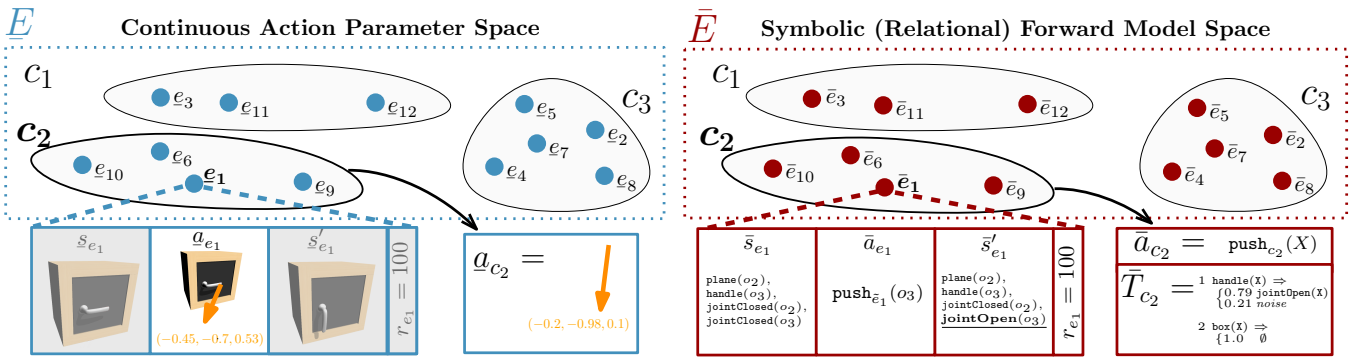


Fig. 2. The coupled learning problem. We see three clusters c_1, c_2, c_3, each containing a subset of all experiences in continuous and relational representation. From the continuous representation of the experiences in cluster c_k we compute the action $\underline{a}_{c_k}$ and associate with it the relational action symbol $\bar{a}_{c_k}$. From the relational representation we compute the relational forward model $\bar{T}_{c_k}$. In the example, $\bar{T}_{c_2}$ contains two rules which predict that the downward-pointing action $\underline{a}_{c_2}$ opens handle joints with probability 79%. (Continuous states are grayed out to indicate that they are not used during learning.)

this problem, we will, in the remainder of this section, define all entities occurring in the previously defined MDPs.

2) State and Action Representation: In our experiments, a continuous state $\underline{s} \in \underline{S}$ represents a scene consisting of different objects. A relational state $\bar{s} \in \bar{S}$ maps $\underline{s}$ to a set of logical predicates capturing task-relevant properties of the scene. It corresponds to a set of positive literals (defined over a set of symbols) which each takes a number of arguments. All literals omitted from the state are implicitly false.

The robot can push and pull objects into different directions applying an action $\underline{a} \in \underline{A}$. We assume $\underline{a}$ to correspond to a vector in some finite-dimensional parameter space $\underline{A}$. In our experiments, $\underline{A}$ is a 3D space encoding push and pull directions (Sec. VI) The relational action $\bar{a}$ maps the continuous action to an action predicate and also instantiates the manipulated object o, e.g. $\texttt{push}(o)$.

Example 1: The cabinet in Fig. 1(a) can be described by the following relational state[1]:

$$\texttt{frame}(o_1), \texttt{plane}(o_2), \texttt{handle}(o_3), \texttt{joint}(o_2), \texttt{joint}(o_3),$$
$$\texttt{jointClosed}(o_2), \texttt{jointClosed}(o_3),$$

The action depicted in Fig. 1(a) is given by $\underline{a} = (0.12, 0.24, 0.96)$ in continuous, and by $\bar{a} = \texttt{pull}(o_3)$ in relational form (see also Sec. VI).

3) Experiences: The robot collects data by randomly sampling continuous actions $\underline{a} \in \underline{A}$, executing them and observing their effects $\bar{s}$. Formally, this gives a set of experiences in continuous and relational form $\mathcal{E} = \{(\underline{e}, \bar{e})_i\}_{i=1}^N$ where $\underline{e}$ denotes the continuous experience $\underline{e} = (\underline{s}_e, \underline{a}_e, \underline{s}'_e, r_e)$, containing continuous predecessor and successor state, continuous action and reward. The relational experience is defined analogously as $\bar{e} = (\bar{s}_e, \bar{a}_e, \bar{s}'_e, r_e)$. Moreover, we will denote by $\underline{E}$ the set of continuous experiences and by $\bar{E}$ the set of relational experiences. Finally, $\Delta\bar{e}$ denotes the *state effect*, i.e. the set of all logical predicates differing between $\bar{s}_e$ and $\bar{s}'_e$; we call $\Delta\bar{e} = \emptyset$ a *void effect*.

Example 2: For the experience $\bar{e}$ in Fig. 1(a) the state effect is $\Delta\bar{e} = \texttt{jointOpen}(o_2), \neg\texttt{jointClosed}(o_2)$.

[1] To simplify the kinematic representation in the relational state, we assume a kinematic tree and set root it at some (arbitrary) object o_i, in our experiments the cabinet frame. Then, a joint between o_i and o_j is represented by the unary predicate $\texttt{joint}(o_j)$, and so forth.

4) Forward Models: In relational model-based RL, the goal is to learn a forward model $\bar{T}$ for each $\bar{a}$, and use it for planning. We will now formalize $\bar{T}$, assuming that actions $\bar{a}$ are given, and cover learning $\bar{T}$ in Sec. V-D.

A forward model $\bar{T}$ consists of a set of *noisy indeterministic deictic (NID) rules* [4]. Each NID rule $\tau \in \bar{T}$ predicts which *outcomes* can occur if action $\bar{a}$ is executed in a *state context*. NID rules are a more expressive version of STRIPS rules with probabilistic outcomes, which account for uncertainty and failures in unstructured environments.

Example 3: The following rule predicts how likely it is that action $\bar{a} = \texttt{push}(\cdot)$ opens the door handle:

$$\begin{aligned} &\textit{CONTEXT}: && \texttt{handle}(X), \texttt{jointClosed}(X) \\ &\textit{ACTION}: && \texttt{push}(X) \\ &\textit{OUTCOMES}: && \begin{cases} 0.59 & \texttt{jointOpen}(X), \neg\texttt{jointClosed}(X) \\ 0.40 & \emptyset \\ 0.01 & \textit{noise} \end{cases} \end{aligned} \tag{1}$$

Note that the NID rule removes reference to concrete objects $o_1, o_2, \ldots$ by substituting variables $X, Y, \ldots$.

To use $\bar{T}$ for planning towards a goal state, we must compute whether one of its rules $\tau \in \bar{T}$ *covers* a state $\bar{s}$. We can then search a sequence of rules/actions that attains the desired goal state. We define that τ covers $\bar{s}$ (we write $\tau \models \bar{s}$) if we can substitute the variables in τ by object identifiers $o \in \mathcal{O}$ such that the rule's context and action are consistent with $\bar{s}$ and $\bar{a}$. Similarly, we define that τ covers an experience $\bar{e}$ (writing $\tau \models \bar{e}$), if at least one of the rule's outcomes covers the experience's successor state $\bar{s}'_e$. Moreover, we write $p(\bar{T} \models \bar{e}) = \bar{p}$ if there exists a rule $\tau \in \bar{T}$ that covers $\bar{e}$ with outcome probability $\bar{p}$, and set $\bar{p} := 0$ otherwise.

Example 4: The NID rule from *Example 3* covers $\bar{s}$ from *Example 1* if we substitute $\texttt{X} := o_3$. It also covers the experience $\bar{e} = (\bar{s}, \bar{a}, \bar{s}', r)$, predicting that action $\bar{a} = \texttt{push}(o_3)$ will lead to successor state $\bar{s}' = (\bar{s} \cup \{\texttt{jointOpen}(o_3)\} \setminus \{\texttt{jointClosed}(o_3)\})$ with probability $p(\tau \models \bar{e}) = 0.59$. (NID rules do not make use of reward r).

V. Coupled Learning of Action Parameters and Relational Forward Models

We can now formalize the problem CAPEL solves as learning a set of K actions, represented by $\underline{a}_{c_1}, \ldots, \underline{a}_{c_K} \in \underline{A}$

in the continuous and by $\bar{a}_{c_1},\dots,\bar{a}_{c_K} \in \bar{A}$ in the relational domain, and their corresponding relational forward models $\bar{T}_{c_1},\dots,\bar{T}_{c_K}$, from experiences $\mathcal{E}$. We index the k-th action and forward model by c_k to indicate that each of them is associated with a cluster of experiences $\mathcal{E}_{c_k} = (\underline{E}_{c_k}, \bar{E}_{c_k})$. Conceptually, the learning problem is a latent variable inference problem: we assume that every experience is an instance of exactly one action $\underline{a}_{c_k}$ and that it is explained by forward model $\bar{T}_{c_k}$. Therefore, the latent variables to be learned are ($\underline{a}_{c_k}$, $\bar{T}_{c_k}$, $\mathcal{E}_{c_k}$). The idea of CAPEL is to jointly cluster the experiences in their continuous and relational representation (Fig. 2), such that we obtain task-relevant $\underline{a}_{c_k}$ and $\bar{T}_{c_k}$.

A. Learning Objective

Our key insight is that finding this clustering can be solved most effectively if all $\underline{a}_{c_k}$ and $\bar{T}_{c_k}$ obey the two priors stated in Sec. III: that *similarly parametrized actions have similar effects* and that *actions have few and accurately predictable effects*, given experiences $\mathcal{E}$. These priors are represented by the following loss function:

$$\mathcal{L} = \underbrace{\sum_{c_k=1}^{K} \mathcal{L}_{\text{action}}(\underline{a}_{c_k}, \mathcal{E}_{c_k})}_{=\mathcal{L}^c_{\text{action}}(\underline{a}_{c_1},\dots,\underline{a}_{c_K},\mathcal{E})} + \underbrace{\sum_{c_k=1}^{K} \mathcal{L}_{\text{effect}}(\bar{T}_{c_k}, \mathcal{E}_{c_k})}_{=\mathcal{L}^c_{\text{effect}}(\bar{T}_{c_1},\dots,\bar{T}_{c_K},\mathcal{E})}. \tag{2}$$

where each term corresponds to one prior, which will be defined in Sections V-C and V-D. Note, that the first term only applies to $\underline{a}_{c_k}$ and the second term only to $\bar{T}_{c_k}$. We will now explain how to exploit this structure for optimization.

B. CAPEL: Iterative Clustering

To optimize (Eq. 2), we apply an iterative k-means-style clustering scheme: we start by randomly sampling K actions $\underline{a}_{c_k}$, assigning each experience to the most similar $\underline{a}_{c_k}$. Then we alternate the following two steps until convergence:

a) Update step: Given an assignment of experiences to clusters $\mathcal{E}_{c_k}$, we compute the actions $\underline{a}_{c_k}$ and forward models $\bar{T}_{c_k}$ that minimize the loss (Eq. 2). Since the loss is a linear combination of two terms, optimization can be carried out separately for $\underline{a}_{c_k}$ wrt. $\mathcal{L}^c_{\text{action}}$ and for $\bar{T}_{c_k}$ wrt. $\mathcal{L}^c_{\text{effect}}$.

b) Assignment step: While keeping all $\underline{a}_{c_k}$ and $\bar{T}_{c_k}$ fixed, we reassign experiences to clusters c_k s.t. (Eq. 2) is minimized. To do so, we must for each experience $\underline{e}$ compute the most similar cluster, according to a suitable similarity metric that we need to provide. The linear combination of the loss terms in (Eq. 2) allows us to decompose the similarity metric likewise ($\underline{a}_e$ denotes the action executed in $\underline{e}$):

$$\text{sim}(c_k, (\underline{e}, \bar{e})) = \text{sim}_{\text{action}}(\underline{a}_{c_k}, \underline{a}_e) + \text{sim}_{\text{effect}}(\bar{T}_{c_k}, \bar{e}), \tag{3}$$

The decomposition of the update and assignment steps into $\mathcal{L}^c_{\text{action}}$ ($\text{sim}_{\text{action}}$) and $\mathcal{L}^c_{\text{effect}}$ ($\text{sim}_{\text{effect}}$) gives us several options for optimizing (Eq. 2): as default, we *simultaneously* optimize a linear combination of the two terms, as described above and summarized in Alg. 1; alternatively, we can perform a *pre-train and finetune* procedure by first optimizing $\mathcal{L}^c_{\text{action}}$ until convergence and then continuing with $\mathcal{L}^c_{\text{effect}}$ – or vice versa. We will evaluate all three procedures in Sec. VII, and now explain how to define the terms in (Eq. 2) and (Eq. 3), and the assignment and update steps for them.

Algorithm 1 CAPEL

```
Pick a_{c_1}, ..., a_{c_K} randomly
for all (e, ē) ∈ (E, Ē) do
    Assign (e, ē) to c_k := arg max_{c_j} sim_action(a_{c_j}, a_e)
L_last := ∞
while true do
    for all c_k ∈ {1, ..., K} do
        # Update step
        Compute a_{c_k} := 1/|Ē_{c_k}| Σ_{e∈E_{c_k}} w_e a_e
        Optimize T̄_{c_k} wrt. L_effect(T̄_{c_k}, E_{c_k})
    Compute L given {a_{c_k}, T̄_{c_k}, E_{c_k}}_{1≤c_k≤K}
    if L = L_last then
        return {a_{c_k}, T̄_{c_k}, E_{c_k}}_{1≤c_k≤K}
    L_last := L
    # Assignment step
    for all (e, ē) ∈ (E, Ē) do
        for all c_k ∈ {1, ..., K} do
            Compute sim(c_k, (e, ē)) using Eq. (3)
        Assign (e, ē) to c_k := arg max_{c_j} sim(c_j, (e, ē))
```

C. Action Similarity Prior: Learning Action Primitives

The first term $\mathcal{L}_{\text{action}}$ expresses the prior that similar actions have similar effects. To implement it, we require all actions in a cluster to be similar, by penalizing low similarity of the cluster centroid $\underline{a}_{c_k}$ to the actions $\underline{a}_e$ assigned to c_k:

$$\mathcal{L}_{\text{action}}(\underline{a}_{c_k}, \mathcal{E}_{c_k}) = -\sum_{e \in \underline{E}_{c_k}} \underline{w}_e \ \text{sim}_{\text{action}}(\underline{a}_{c_k}, \underline{a}_e), \tag{4}$$

where $\text{sim}_{\text{action}}$ is a similarity metric defined on $\underline{A}$, and $\underline{w}_e$ is an experience-specific weight (defined below).

a) Update step: It is easy to show that, given a set of experiences $\underline{E}_{c_k}$, we can compute the optimal $\underline{a}_{c_k}$ as the (weighted) mean over the continuous actions in $\underline{E}_{c_k}$.

b) Assignment step: We use $\text{sim}_{\text{action}}(\underline{a}_{c_k}, \underline{a}_e)$ to determine the cluster with the highest similarity to experience $\underline{e}$.

Note that these two steps implement the standard k-means algorithm, applied to the continuous actions of the experience set $\underline{E}$. However, for learning task-relevant actions, we have to ignore experiences with void effects ($\Delta\bar{e} = \emptyset$) during the update step: we can't decide whether an action instance $\underline{a}_e$ has no effect due to the world state or due to being an illegal variation of $\underline{a}_{c_k}$; in the latter case, $\underline{a}_e$ would wrongly skew the centroid $\underline{a}_{c_k}$, and so we set $\underline{w}_e = \mathbb{1}\{\Delta\bar{e} \neq \emptyset\}$ where $\mathbb{1}\{\cdot\}$ returns 1 for a true and 0 for a false statement. (This has no effect on the assignment step or the forward model.)

D. Effect Prior: Learning Relational Forward Models

The second term $\mathcal{L}_{\text{effect}}$ implements the prior that actions have few and accurately predictable effects. It imposes a loss over the set of forward models $\bar{T} = \{\bar{T}_{c_1},\dots,\bar{T}_{c_K}\}$:

$$\mathcal{L}_{\text{effect}}(\bar{T}_{c_k}, \mathcal{E}_{c_k}) = -\sum_{\bar{e} \in \bar{E}_{c_k}} \bar{w}_e \log p(\bar{T}_{c_k} \models \bar{e}) + \alpha\ \text{PEN}(\bar{T}_{c_k}) + \beta\ \text{EFF}(\bar{E}_{c_k}), \tag{5}$$

where α and β are (positive) hyperparameters. (Eq. 5) is an extension of the loss defined in [4], which originally contains

only the first two terms: the *weighted (log-)likelihood* of experiences given the model, which ensures that the rules explain the experiences well; and the *regularization* term PEN that penalizes rules that overfit to outliers (by counting the number of literals in the rules). We add the *multi-effect penalty* term EFF to force each model to specialize on few effects. It does not depend on $\bar{T}_{c_k}$ and thus only influences the assignment step. We implement it by computing for each (non-void) experience the fraction of experiences in $\bar{E}_{c_k}$ with the same effect:

$$\text{EFF}(\bar{E}_{c_k}) = \sum_{\bar{e} \in \bar{E}_{c_k}} \mathbb{1}\{\Delta\bar{e} \neq \emptyset\} p(\Delta\bar{e} \in \bar{E}_{c_k}). \tag{6}$$

Additionally, we weigh experiences $\underline{e}$ according to the similarity of their *continuous* action $\underline{a}_e$ to the action centroid $\underline{a}_{c_k}$ by setting $\bar{w}_e = \text{sim}_{\text{action}}(\underline{a}_{c_k}, \underline{a}_e)$. (Note how $\bar{w}_e$ and $\underline{w}_e$ provide additional coupling, although $\underline{w}_e \neq \bar{w}_e$.)

However, the update and assignment steps for optimizing $\mathcal{L}^c_{\text{effect}}$ turn out to be much more difficult than for $\mathcal{L}^c_{\text{action}}$:

a) Update step: Computing the optimal $\bar{T}_{c_k}$ from a set of relational experiences $\bar{E}_{c_k}$ is an NP-complete learning problem. We therefore apply a local optimization by performing a greedy search over models $\bar{T}_{c_k}$ wrt. loss $\mathcal{L}_{\text{effect}}(\bar{T}_{c_k}, \bar{E}_{c_k})$. Due to space constraints we cannot provide further details about learning $\bar{T}_{c_k}$ and refer the reader to [4], [17].

b) Assignment step: To assess whether to reassign an experience $\bar{e}$ to a different $\bar{T}_{c_k}$, we define the similarity metric $\text{sim}_{\text{effect}}(\bar{e}_{c_k}, \bar{T}_{c_k})$ as the relative increase of $\mathcal{L}_{\text{effect}}$ when moving $\bar{e}$ to cluster c_k [18]. It is computed as the difference between the cost of removing $\bar{e}$ from its current cluster $\bar{T}_{c_\ell}$ and the cost of adding it to $\bar{T}_{c_k}$:

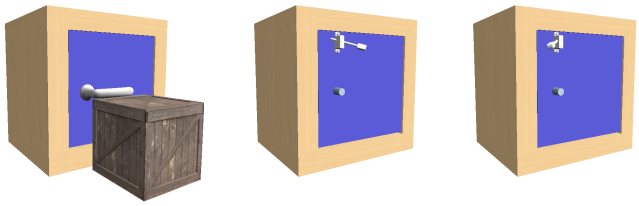
$$\begin{aligned}\text{sim}_{\text{effect}}(e_{c_k}, \bar{T}_{c_k}) = [&\mathcal{L}_{\text{effect}}(\bar{T}'_{c_k}, \mathcal{E}_{c_k} \cup \{e\}) - \mathcal{L}_{\text{effect}}(\bar{T}_{c_k}, \mathcal{E}_{c_k})] \\ -[&\mathcal{L}_{\text{effect}}(\bar{T}'_{c_\ell}, \mathcal{E}_{c_\ell} \setminus \{e\}) - \mathcal{L}_{\text{effect}}(\bar{T}_{c_\ell}, \mathcal{E}_{c_\ell})].\end{aligned} \tag{7}$$

However, this way of implementing the assignment step puts us at risk that $\mathcal{L}_{\text{effect}}$ increases (e.g. when experiences with contradicting effects are moved to the same cluster). We mitigate this effect by allowing CAPEL to move an experience only if the overall similarity (Eq. 3) is above a certain threshold, in our experiments 0.05.

Moreover, computing $\text{sim}_{\text{effect}}$ is costly as it requires learning two new forward models $\bar{T}'_{c_k}$ and $\bar{T}'_{c_\ell}$. To compute it efficiently, we do not relearn the models but instead apply different incremental modifications to $\bar{T}_{c_k}$, e.g. adding a new rule, adding a new outcome, dropping literals from the context of existing rules, not changing $\bar{T}_{c_k}$ at all, etc. and select the modification $\bar{T}'_{c_k}$ that decreases the loss most.

VI. Simulated Furniture Scenarios

We evaluate CAPEL in experiments with articulated objects, inspired by modular furniture [17]. To collect a large batch of data to evaluate CAPEL we conduct experiments in simulation. Although the physical realization of the perceptual and motor capabilities is outside the scope of this paper, we provide references to prior works that implement them on a physical system [19], [20].



(a) Box and Handle (b) Latch Type-1 (c) Latch Type-2

Fig. 3. Articulated furniture objects used in experiments.

Objects: The objects are cabinets with two different types of opening mechanisms, a rotary *handle* and a *latch with a knob* (Fig. 3). The handle and the latch have to be unlocked before the door can be opened by pulling the handle or the knob, respectively. The handle and knob have a large variability in terms of the action parameters required to actuate them, whereas the latch requires the direction of motion to be precisely parallel to its joint axis. Additionally, the cabinet can be obstructed by a *box* that must be pushed either to the left or right before accessing the door.

TABLE I

State Symbols for the Furniture Scenario

Relation/Function	Arity	Explanation
Object properties		
`plane, handle, knob, latch1, latch2, box`	1	object type
`blocked`	1	object is blocked (by box)
`blocking`	1	object is blocking some object
Kinematic structure		
`joint`	1	connected by a joint
`jointOpen, jointClosed`	2	joint configuration

States: A continuous state $\underline{s} \in \underline{S}$ consists of a set of objects $o \in \mathcal{O}$ in the scene, including their type, pose and geometry (represented as 3D meshes), and a set of joints and their configuration [20]. The continuous state $\underline{s}$ is only available during simulation, whereas the relational state $\bar{s} \in \bar{S}$ (composed of symbols listed in Table I) is used for learning.

Actions: We assume that the robot can establish a fixed grasp with an object $o \in \mathcal{O}$ and can choose a direction to push or pull it. Thus, we define the continuous action $\underline{a} \in \underline{A}$ as a normalized directional vector $\underline{A} = \mathbb{R}^3, ||\underline{a}|| = 1$, defined in the coordinate frame of object o, and the relational action by $\bar{a} = \texttt{push}(o)$. We define the similarity $\text{sim}_{\text{action}}$ between two actions as the dot product if this product is positive, and to 0 otherwise (when vectors point into opposing directions).

Action Execution: To push or pull the robot applies force control: it starts with zero force and linearly increases it until the object starts moving by at least $1cm/s$. Motion is aborted if a force limit or time limit of $5s$ is reached. We adapt the force direction compliantly to the motion of an object [19]. To simulate motion uncertainty we add Gaussian noise with a standard deviation of 10 deg. to the direction vector $\underline{a}$.

VII. Experiments

The goal of the experiments is to verify our initial hypothesis from Sec. III: for learning task-relevant action capabilities, the robot must consider both (i) the continuous action representation and (ii) its effects in relational representation. To demonstrate that coupled learning in both representations is required, we devise two scenarios, each requiring the robot to learn (at least) five different action capabilities:

The *box & handle scenario* mainly requires learning in continuous action parameter space: it contains the cabinet with a handle attached to it, which is additionally obstructed by a box (Fig. 3(a)). The scenario requires six actions: two for opening and closing the handle, two for moving the box left or right, and two for opening/closing the door. We hypothesize that purely effect-based learning is unable to learn the two opposing direction vectors for removing the box (as they produce the same relational effect).

The *latch scenario* requires learning in relational effect space: it consists of two cabinets, each with a distinct latch type (`latch1/2`; Fig. 3(b)–3(c)). The latches are oriented such that no single action opens or closes both of them (six actions required). Thus, the robot must consider the effects to distinguish the latches and to learn accurate parameters.

1) Data collection: In each scenario, the robot interacts with cabinets in different states (open, locked, closed, unlocked) by randomly selecting the handle, box, latch or knob, uniformly sampling a direction vector and then applying the action. This way the robot collects roughly 500 experiences for the box & handle and 1000 for the latches scenario (the latter contains more experiences with void effects because the latches require precise motion to be actuated). We repeat this procedure three times to obtain three datasets per scenario.

2) CAPEL and Baselines: We evaluate three variants of our method: *CAPEL (simul.)* jointly optimizes $\mathcal{L}$ as described in Sec. V; the other two, *CAPEL (act.→eff.)*, *CAPEL (eff.→act.)*, apply the pre-train-and-fine-tune optimization described in Sec. V-B. The hyperparameters for CAPEL only pertain to the forward model learner. Preliminary experiments and previous work [17] showed that it is not very sensitive to these values, and we thus use low regularization $\alpha = 10^{-5}$ and a moderate multi-effect penalty $\beta = 1$.

Additionally, we test three baselines: only optimizing $\mathcal{L}^c_{\text{action}}$ (*action only*, equivalent to k-means), only optimizing $\mathcal{L}^c_{\text{effect}}$ (*effect only*), and random (uniform) sampling of K continuous actions (for which we also learn forward models). We test $K = \{5, \ldots, 12\}$ and run each strategy five times with different random initializations.

3) Evaluation: First, we evaluate (i) the loss (Eq. 2) for each strategy. Next, we evaluate all the optimized quantities: (ii) to evaluate the *centroids* $\underline{a}_{c_k}$, we expose the robot to every mechanism in all configurations and test whether it is able to change its state (open to closed, locked to unlocked, etc.) with only one action. (iii) To test the clustering $\mathcal{E}_{c_k}$, we treat every cluster as a non-parametric action parameter *distribution*: we perform the same test as in (ii), but instead of applying $\underline{a}_{c_k}$ we randomly sample an experience $\underline{e} \in \underline{E}_{c_k}$ and execute its action $\underline{a}_{\underline{e}}$. (iv) To test the *planning* capabilities of the forward models $\bar{T}_{c_k}$, the robot plans and executes a full sequence of actions for opening and then closing (and locking) each door (maximum of 5 actions). In all tests, we use the relational planner PRADA [5].

4) Results: Fig. 5 shows results for different K, averaged over five random strategy initializations and three datasets.

Loss: As expected, *CAPEL (simul.)* performs best wrt. the loss (Eq. 2; Fig. 5, left column). The inferior performance of *action only*, *CAPEL (eff.→act.)* and the random strategy is due to the fact that the learned forward models contain highly stochastic rules.

Centroids, Distribution and Planning: *CAPEL (simul.)* manages to learn better centroids and distributions, in particular for lower values of K: it performs best for $K = 9$. The superior performance of *CAPEL (simul.)* is even more pronounced in the planning experiment. Interestingly, in the centroid and distribution experiments, *action only* and *CAPEL (eff.→act.)* perform only slightly worse than *CAPEL (simul.)* on average, and even tend to perform better with higher K. *CAPEL (act.→eff.)* performs moderately well, whereas *effect only* exhibits performance close to random.

No strategy reaches 100% success: the first reason is the noise we apply to the action parameters. Second, bad random initializations cause the strategies to converge to non-global optima, preventing the robot from learning task-relevant actions. (For every dataset and at least one initialization, though, *CAPEL (simul.)* with $K > 5$ learns optimal actions and forward models; with $K = 5$ no strategy is able to actuate all mechanisms.) Third, in some training samples an action evokes undesired side effects; e.g., in the box-and-handle scenario the robot simultaneously unlocks the handle and opens the door, or locks the handle while opening the door.

5) Discussion: We now discuss whether the results confirm that coupled learning is required to solve the task.

Action parameter learning is required: Both, the box-and-handle as well as the latch scenario show that pure effect-based learning fails. In both scenarios there are identical effects (removing the box, operating `latch1` or `latch2`) that require different action primitives. Additionally, in the box-handle scenario a wide variety of effects can occur, e.g. handle and door opening simultaneously. Although these effects are results of slight variations of the action required to open a door, the *effect only* strategy wrongly considers them different actions – as explained in Sec. III and Fig. 1.

Effect learning is required: Fig. 5 shows that *action only* performs well on average. The reason is that most required actions are roughly orthogonal (e.g. opening door vs. opening handle) and do not require an exact parametrization. However, the precise actions required to operate the latches pose a challenge for *action only*: Fig. 4 shows that, indeed, *action only* (and also *CAPEL (eff.→act.)*) perform worse at locking and unlocking these latches. This is confirmed by visual inspection of the clustering in action parameter space: Fig. 6, left, shows that *CAPEL (simul.)* correctly learns the distribution of the latch opening action parameters (c_3 and c_5), even though they overlap with the cluster for door opening (c_4). Fig. 6, right, shows that *action only* overfits to the sampling distribution.

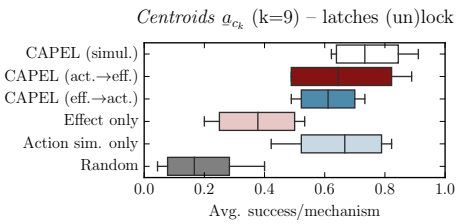


Fig. 4. Centroids in latch scenario $K = 9$.

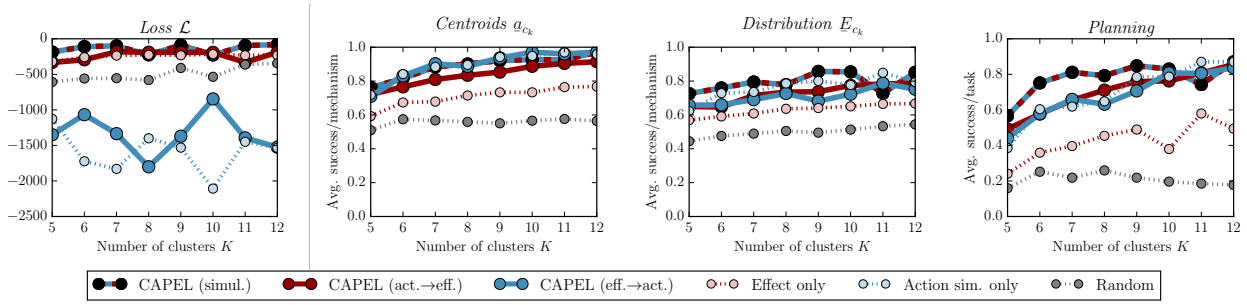


Fig. 5. Averaged results for the two learning scenarios. The plots show (from left to right): loss function score, avg. success in centroid experiment, in distribution experiment and in planning experiment. We have omitted the standard error to increase readability; in general, we observed the standard error to be lower for *CAPEL (simul.)* than for the other strategies.

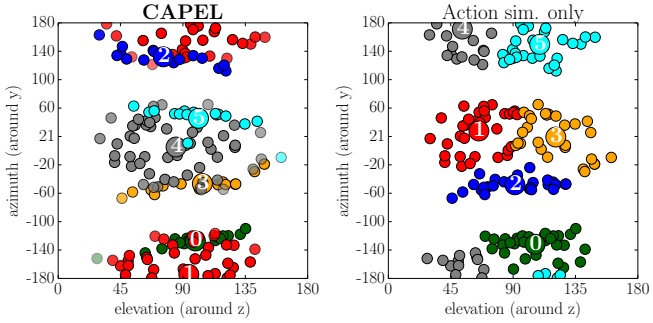


Fig. 6. Latch scenario, exemplary clustering results in action parameter space for $K = 6$. To provide a 2D visualization, we translate 3D actions to spherical coordinates. Only experiences with non-void effects are shown.

In summary, these results confirm that CAPEL is required to learn task-relevant actions for both scenarios.

6) Limitations: First, CAPEL requires the training data to cover all relevant parts of the action parameter space. To enable learning in high-dimensional action spaces, CAPEL needs to perform efficient exploration. It is unclear, though, how to perform simultaneous exploration in continuous and relational representations. Second, CAPEL relies upon predefined state symbols. This is orthogonal to the assumption made by approaches to state symbol learning [6], [7], [15], as these require the robot to be equipped with task-relevant, grounded action symbols. How to learn both types of symbols simultaneously is an open research question. Finally, CAPEL's performance is likely to degrade if the state contains task-irrelevant distractors. To address this issue, we need to find better regularization for forward model learning.

VIII. Conclusion

In this paper, we presented CAPEL, an approach for action parameter and forward model learning for manipulation. CAPEL solves these two learning problems in a coupled fashion by simultaneously clustering a set of experiences in a continuous and relational representation. Our simulation experiments show that CAPEL enables the robot to learn task-relevant action parameters and forward models, by effectively combining subsymbolic and symbolic learning.

References

[1] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal, "Skill learning and task outcome prediction for manipulation," in *ICRA 2011*, 2011, pp. 3828–3834.

[2] K. Mülling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *The International Journal of Robotics Research*, vol. 32, no. 3, pp. 263–279, Mar. 2013.

[3] A. Lerer, S. Gross, and R. Fergus, "Learning Physical Intuition of Block Towers by Example," *arXiv:1603.01312 [cs]*, Mar. 2016.

[4] H. M. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling, "Learning symbolic models of stochastic domains," *Journal of Artificial Intelligence Research*, vol. 29, no. 1, pp. 309–352, 2007.

[5] T. Lang and M. Toussaint, "Planning with noisy probabilistic relational rules," *Journal of AI Research*, vol. 39, pp. 1–49, 2010.

[6] N. Jetchev, T. Lang, and M. Toussaint, "Learning grounded relational symbols from continuous data for abstract reasoning," in *ICRA 2013 Workshop on Autonomous Learning*, 2013.

[7] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "Constructing symbolic representations for high-level planning," in *Twenty-Eighth AAAI Conference on Artificial Intelligence*, Québec, Canada, 2014.

[8] E. Şahin, M. Çakmak, M. R. Doğar, E. Uğur, and G. Üçoluk, "To afford or not to afford: A new formalization of affordances toward affordance-based robot control," *Adaptive Behavior*, vol. 15, no. 4, pp. 447–472, Dec. 2007.

[9] N. Krüger, C. Geib, J. Piater, R. Petrick, M. Steedman, F. Wörgötter, A. Ude, T. Asfour, D. Kraft, D. Omren, A. Agostini, and R. Dillmann, "Object-action complexes: Grounded abstractions of sensory-motor processes," *Robotics and Autonomous Systems*, vol. 59, no. 10, pp. 740–757, Oct. 2011.

[10] F. Stulp, L. Herlant, A. Hoarau, and G. Raiola, "Simultaneous online discovery and improvement of robotic skill options," in *IEEE/RSJ IROS 2014*, Chicago, Illinois, USA, 2014.

[11] N. Abdo, L. Spinello, W. Burgard, and C. Stachniss, "Inferring what to imitate in manipulation actions by using a recommender system," in *ICRA 2014*, 2014, pp. 1203–1208.

[12] M. Kopicki, S. Zurek, R. Stolkin, T. Morwald, and J. Wyatt, "Learning to predict how rigid objects behave under simple manipulation," in *ICRA 2011*, 2011, pp. 5722–5729.

[13] S. Džeroski, L. De Raedt, and K. Driessens, "Relational reinforcement learning," *Machine Learning*, vol. 43, no. 1-2, pp. 7–52, 2001.

[14] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, Aug. 2013.

[15] J. Mugan and B. Kuipers, "Autonomous learning of high-level states and actions in continuous environments," *IEEE Transactions on Autonomous Mental Development*, vol. 4, no. 1, pp. 70–86, Mar. 2012.

[16] A. Orthey, M. Toussaint, and N. Jetchev, "Optimizing motion primitives to make symbolic models more predictive," in *ICRA*, 2013.

[17] S. Höfer, T. Lang, and O. Brock, "Extracting kinematic background knowledge from interactions using task-sensitive relational learning," in *ICRA 2014*, Hong Kong, China, 2014.

[18] J. A. Hartigan, *Clustering Algorithms*, 99th ed. New York, NY, USA: John Wiley & Sons, Inc., 1975.

[19] Y. Karayiannidis, C. Smith, F. E. Vina, P. Ogren, and D. Kragic, "Model-free robot manipulation of doors and drawers by means of fixed-grasps," in *ICRA*, 2013.

[20] R. Martín-Martín, S. Höfer, and O. Brock, "An Integrated Approach to Visual Perception of Articulated Objects," in *ICRA 2016*, Stockholm, Sweden, 2016.