Towards Learning Hierarchical Skills for Multi-Phase Manipulation Tasks

Oliver Kroemer, Christian Daniel, Gerhard Neumann, Herke van Hoof, and Jan Peters

***Abstract*— Most manipulation tasks can be decomposed into a sequence of phases, where the robot's actions have different effects in each phase. The robot can perform actions to transition between phases and, thus, alter the effects of its actions, e.g. grasp an object in order to then lift it. The robot can thus reach a phase that affords the desired manipulation.**

In this paper, we present an approach for exploiting the phase structure of tasks in order to learn manipulation skills more efficiently. Starting with human demonstrations, the robot learns a probabilistic model of the phases and the phase transitions. The robot then employs model-based reinforcement learning to create a library of motor primitives for transitioning between phases. The learned motor primitives generalize to new situations and tasks. Given this library, the robot uses a value function approach to learn a high-level policy for sequencing the motor primitives. The proposed method was successfully evaluated on a real robot performing a bimanual grasping task.

I. Introduction

In order to perform a wide range of tasks, robots need to learn how to interact and manipulate objects. A robot can manipulate different parts of its environment by fulfilling certain preconditions. For example, a robot can apply forces to an object by first moving into contact with it. The robot may in turn need to alter other parts of the environment's state to achieve these conditions. Hence, a robot will often have to perform a sequence of manipulations to alter different parts of the environment's state. These stages of the manipulation task are referred to as phases [12], [19], and a phase transition usually results in the robot's actions having different effects.

The robot will often need to transition through multiple phases before reaching one that allows for the desired manipulation. The conditions for transitioning between phases therefore represent important subgoals of the overall task. A robot can learn individual skills for transitioning between pairs of phases. In this manner, the robot learns skills that can be reused between tasks with similar phase sequences. The skills for accessing different parts of the environment's state are thus also separate from the task-specific skills for changing those parts of the state.

In this paper, we present a method for learning hierarchical manipulation skills based on the task's phase structure. The robot first ① learns a probabilistic multi-phase model of the task using a state-based transitions autoregressive hidden Markov model (STARHMM) [25]. This model captures the phase transition conditions that represent the subgoals of the task. The robot then ② learns a motor primitive for each of these subtasks ③ and optimizes it using a policy search reinforcement learning approach. Finally, ④ the robot uses a value function approach to learn a high-level policy for sequencing the motor primitives. In this manner, the robot can learn policies that reuse motor primitives between tasks with the same phase transitions.

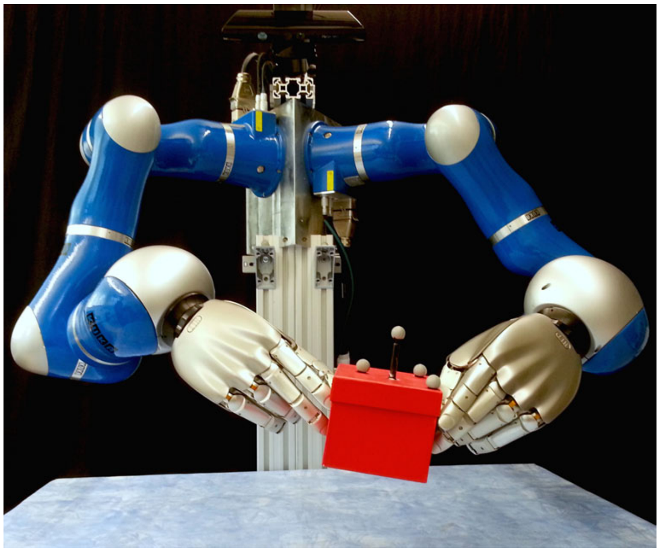

Fig. 1. The Darias robot performing a bimanual grasp of a box. The motor primitives used to perform the task, as well as the policy for sequencing them, were learned using a reinforcement learning approach. The model of the task's phases was learned from human demonstrations.

The proposed method was evaluated using the robot shown in Fig. 1. Given only two demonstrations of bimanual grasping, the robot successfully decomposed the task into five phases, and it learned a library of motor primitives for transitioning between them. The learned motor primitives generalize to different object locations, as well as objects with different shapes. The robot then learned high-level policies for both bimanual grasping task and pushing tasks.

This paper presents three main contribution: 1) An extension to the STARHMM that allows the robot to model the entry and exit conditions of phases, which leads to more efficient learning of the phase transitions. 2) Using the learned phase transitions to optimize motor primitives for reliably transitioning between phases. 3) Using non-parametric dynamic programming to learn a high-level policy for sequencing motor primitives, including primitives that were learned from other tasks.

All authors are members of the Intelligent Autonomous Systems group at the Technische Universitaet Darmstadt, Germany. Jan Peters is also a member of the Max Planck Institute for Intelligent Systems, Germany.
{kroemer, daniel}@ias.tu-darmstadt.de
{neumann,hoof,peters}@ias.tu-darmstadt.de

II. Multi-Phase Models

This section explains how the robot learns a multi-phase model of the manipulation task. We assume that the robot is initialized with human demonstrations of the task. The model can be updated to incorporate additional data as the robot attempts the task. The model structure is outlined in Section II-B, and the learning is explained in Section II-C.

A. Related Work

Previous work on learning sequences of motor primitives for manipulation tasks has mainly focused on imitation learning from human demonstrations. While some methods assume that a library of motor primitives already exists and focus on sequencing these primitives [30], [27], other approaches also learn the individual skills by segmenting the human demonstrations [29], [6], [14], [38]. The resulting segments are therefore linked to specific movements and the sequencing of primitives is done according to the demonstrated task. In contrast, our approach performs the segmentation based on the effects of the actions, and uses the resulting subgoals to optimize the motor primitives. Multi-modal planning methods have also been used to sequence actions in order to achieve different manipulation goals [3], [16]. However, the actions, the discrete modes of the system, and the system model are usually predefined.

Some methods have also been proposed for using reinforcement learning to improve the performance of skill sequences [21], [35], [9]. These approaches learn to optimize sequences of skills for specific tasks, and the skills are generally executed in a fixed order. Hiearchical reinforcement learning methods learn both a library of skills as well as a high-level policy for sequencing them [37], [13], [20], [34], [15]. However, these methods often rely on a predefined set of subtasks or salient events [20], [34], or use a scaffolding approach to direct the hierarchical learning process [13], [15]. In contrast, our approach ① first learns a model of the task to determine the salient subgoals, and ② then learns a corresponding skill library. ③ The robot subsequently learns a high-level policy, which can reuse skills between tasks.

Previous work in robotics has already shown the benefits of incorporating phases into the design of controllers [32], [10], and several methods have been proposed for learning controllers for multi-phase tasks [22], [26], [1], [28]. Levine el al. [26] proposed learning monolithic neural network controllers for multi-phase manipulation tasks. Koval et al. [22] decompose a grasping policy into pre- and post-contact policies. Mugan and Koipers [28] learn a model by discretizing the entire state and action spaces, and then applying reinforcement learning to the discrete domain.

B. Modeling Multi-Phase Manipulation Tasks

The observed state of the robot and its environment at time t are given by the state $\boldsymbol{s}_t \in \mathbb{R}^n$. The robot then performs an action $\boldsymbol{a}_t \in \mathbb{R}^m$, which results in the state transitioning to the next state $\boldsymbol{s}_{t+1} \in \mathbb{R}^n$. This change in state depends on the current phase $\rho_t \in \{1, ..., \kappa\}$, which is hidden. In this paper, the *phase* corresponds to the hidden state of a HMM, and the *state* refers to the observed state. As the phases are not directly observed, the robot needs to infer the phase from the observed states and the actions' effects.

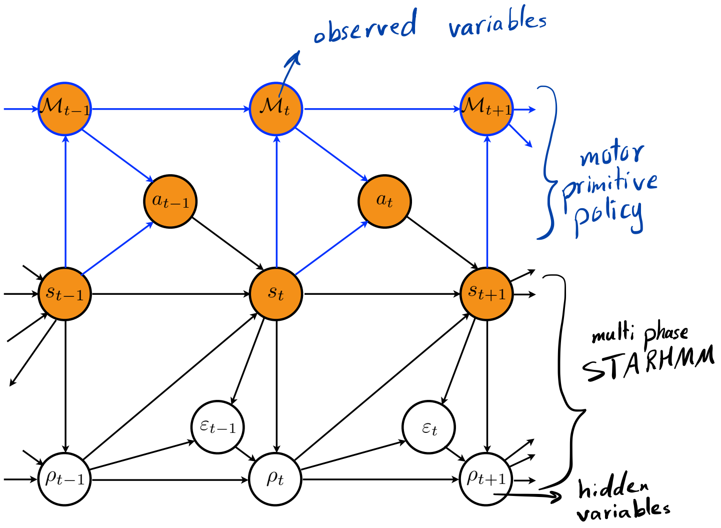

Fig. 2. Graphical model of the multi-phase STARHMM (Black) and the motor primitive policy (Blue). The orange nodes indicate observed variable. The white nodes indicate hidden variables. The model includes the states $\boldsymbol{s}$, actions $\boldsymbol{a}$, motor primitives $\mathcal{M}$, termination variables ε, and the phases ρ. The figure also shows how the motor primitives reflect the phase structure.

The effects of performing an action $\boldsymbol{a}_t$ in state $\boldsymbol{s}_t$ and phase ρ_t are modeled by the transition probability $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t, \rho_t)$. We represent the state transitions $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t, \rho_t)$ using a linear Gaussian model such that $\boldsymbol{s}_{t+1} \sim \mathcal{N}(\boldsymbol{A}_{\rho_t}\boldsymbol{s}_t + \boldsymbol{B}_{\rho_t}\boldsymbol{a}_t, \boldsymbol{\Sigma}_{\rho_t})$, where $\boldsymbol{A}_i \in \mathbb{R}^{n\times n}$, $\boldsymbol{B}_i \in \mathbb{R}^{n\times m}$, and $\boldsymbol{\Sigma}_i \in \mathbb{R}^{n\times n}$ correspond to phase $\rho = i$.

The phase transition distribution depends on the current state and the previous phase $p(\rho_t|\boldsymbol{s}_t, \rho_{t-1})$. The dependency on the previous phase allows the model to represent hysteresis effects, and transient state information. The dependency on the current state allows the model to learn the conditions where phase transitions are more likely to occur. In order to learn the entry and exit conditions of phases, we incorporate a binary termination variable $\varepsilon_t \in \{0, 1\}$, which is distributed according to the termination distribution $p(\varepsilon_t|\boldsymbol{s}_{t+1}, \rho_t)$. If the termination variable is zero $\varepsilon_t = 0$, no phase transition can occur, i.e., $p(\rho_{t+1} = \rho_t|\rho_t, \varepsilon_t = 0) = 1$. If the termination variable is one $\varepsilon_t = 1$, the next phase ρ_{t+1} is distributed according to the initiation distribution $p(\rho_{t+1}|\boldsymbol{s}_t, \varepsilon_t = 1)$. The next phase can be the same as the current phase $\rho_{t+1} = \rho_t$ even if the termination variable is one $\varepsilon_t = 1$. The phase transition distribution can be computed by marginalizing out the termination variable

$\int p(\rho_t|\rho_{t-1}, \varepsilon_{t-1})\, d\varepsilon$ marginalization of ε

$$\begin{aligned} p(\rho_t|\boldsymbol{s}_t, \rho_{t-1}) &= p(\rho_t|\rho_{t-1}, \varepsilon_{t-1} = 0)p(\varepsilon_{t-1} = 0|\boldsymbol{s}_t, \rho_{t-1}) \\ &+ p(\rho_t|\boldsymbol{s}_t, \varepsilon_{t-1} = 1)p(\varepsilon_{t-1} = 1|\boldsymbol{s}_t, \rho_{t-1}). \end{aligned}$$

The phase termination distribution is modeled using logistic regression

$$p(\varepsilon_{t-1} = 1|\boldsymbol{s}_t, \rho_{t-1} = i) = (1 + \exp(-\hat{\boldsymbol{\omega}}_i^T \boldsymbol{\phi}(\boldsymbol{s}_t)))^{-1},$$

where $\hat{\boldsymbol{\omega}}_i \in \mathbb{R}^d$ is a weight vector for terminating phase $\rho = i$, and $\boldsymbol{\phi}(\boldsymbol{s}_t) \in \mathbb{R}^d$ is a feature vector. Features may, for example, be a subset of the full state vector or additionally include the positions of objects relative to each other. We model the phase initiation distribution as

$$p(\rho_t = j|\boldsymbol{s}_t, \varepsilon_{t-1} = 1) = \frac{\exp(\check{\boldsymbol{\omega}}_j^T \boldsymbol{\phi}(s_t))}{\sum_k \exp(\check{\boldsymbol{\omega}}_k^T \boldsymbol{\phi}(s_t))},$$

where $\check{\boldsymbol{\omega}}_j \in \mathbb{R}^d$ is a weight vector for initiating phase $\rho = j$. We assume that the previous phase terminated at the start of the trajectory, such that $p(\rho_1 = j|\boldsymbol{s}_1) = p(\rho_1 = j|\boldsymbol{s}_1, \varepsilon_0 = 1)$. The policy for selecting actions will be discussed in Section III. To improve clarity, we can assume that the actions are drawn from some fixed distribution $p(\boldsymbol{a}_t)$ when learning the multi-phase model.

Given the components of the model, the probability of observing a sequence of N samples of states $\boldsymbol{s}_{1:N+1} = \{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{N+1}\}$, actions $\boldsymbol{a}_{1:N} = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N\}$, phases $\rho_{1:N} = \{\rho_1, \ldots, \rho_N\}$, and phase terminations $\varepsilon_{0:N-1} = \{\varepsilon_0, \ldots, \varepsilon_{N-1}\}$ is given by

$$p(\boldsymbol{s}_{1:N+1}, \boldsymbol{a}_{1:N}, \rho_{1:N}, \varepsilon_{0:N-1}) =$$

$$p(\varepsilon_0, \rho_1, \boldsymbol{s}_1)\prod_{t=1}^{N} p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t, \rho_t)p(\boldsymbol{a}_t)\prod_{t=2}^{N} p(\rho_t, \varepsilon_{t-1}|\boldsymbol{s}_t, \rho_{t-1}),$$

where $p(\varepsilon_0, \rho_1, \boldsymbol{s}_1) = p(\boldsymbol{s}_1, \varepsilon_0)p(\rho_1|\boldsymbol{s}_1, \varepsilon_0)$ and

$$p(\rho_t, \varepsilon_{t-1}|\boldsymbol{s}_t, \rho_{t-1}) = p(\rho_t|\boldsymbol{s}_t, \rho_{t-1}, \varepsilon_{t-1})p(\varepsilon_{t-1}|\boldsymbol{s}_t, \rho_{t-1}).$$

The graphical model of this probability factorization is shown in Fig. 2 in black. The key difference to an autoregressive HMM is the additional edge from the current state to the current phase. As a result of this edge, the transitions between phases depend on the observed state. The STARHMM with entry and exit conditions, presents important benefits over the original model [25]. Rather than having to learn $\kappa + 1$ multi-class classifiers, the robot only needs to learn κ binary classifiers and one multi-class classifier. The extended version also results in a more consistent mapping from the state space to phases.

In the proposed model, we assumed that the state is observable. This assumption is common for segmenting manipulations movements [6], [29]. However, one could also extend the model to use observations of hidden states [2].

C. Model Learning Using Expectation-Maximization

Having defined the structure of the model, we now focus on learning the model parameters $\hat{\boldsymbol{\omega}}$, $\check{\boldsymbol{\omega}}$, $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{\Sigma}$, which we will refer to jointly as $\boldsymbol{\theta} = \{\hat{\boldsymbol{\omega}}, \check{\boldsymbol{\omega}}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\Sigma}\}$. Given a set of sampled trajectories of states and actions, we propose using the *expectation-maximization* (EM) algorithm [4] to estimate the parameters. The EM algorithm iterates between an expectation step and a maximization step in order to find maximum likelihood estimates of the model parameters $\boldsymbol{\theta}$ given that some of the variables are hidden $\boldsymbol{\mathcal{H}} = \{\rho_{1:N}, \varepsilon_{0:N-1}\}$, i.e., the phases and the phase terminations of the samples are not known. The steps of the algorithm are explained below.

EXPECTATION STEP*:* In the expectation step, we compute the distribution over the hidden states, i.e., the phases, given the observed sequence of variables. In particular, we compute the distribution $p(\rho_t, \varepsilon_t, \rho_{t+1}|\boldsymbol{s}_{1:N+1}, \boldsymbol{a}_{1:N})$ for the computations in the maximization step. We compute these marginal probabilities efficiently by using a forward-backward message passing approach [4]. During the expectation step, we assume that the parameters $\boldsymbol{\theta}$ of our model are fixed.

To improve the clarity of the methodology below, we define $\boldsymbol{z}_t = \{\boldsymbol{s}_t, \boldsymbol{a}_t\}$ as the observed states and actions together. The final sample is given by $\boldsymbol{z}_{N+1} = \boldsymbol{s}_{N+1}$. Thus, we have $p(\boldsymbol{z}_{t+1}|\rho_t, \boldsymbol{z}_t) = p(\boldsymbol{s}_{t+1}|\rho_t, \boldsymbol{s}_t, \boldsymbol{a}_t)p(\boldsymbol{a}_{t+1})$, and $p(\rho_t|\rho_{t-1}, \boldsymbol{z}_t) = p(\rho_t|\rho_{t-1}, \boldsymbol{s}_t)$ as ρ_t is not conditioned on $\boldsymbol{a}_t$. We also implicitly marginalize out the termination variables ε and use $p(\rho_t|\boldsymbol{s}_t, \rho_{t-1})$ to define the messages.

We first send a series of messages forward through the network from $t = 1$ to $t = N$. The forward messages α give the probability of observing the sequence of states, actions, and next state up to the current time step $\alpha_j(t) = p(\boldsymbol{z}_{1:t+1}, \rho_t = j)$. The backward messages β are sent backwards from $t = N$ to $t = 1$ and give the probability of observing the remainder of the observed sequence of states, actions, and next states given the current phase and next state $\beta_j(t) = p(\boldsymbol{z}_{t+2:N}|\rho_t = j, \boldsymbol{z}_{t+1})$. Both the forward and backward messages are computed recursively on the previous message. Given these messages, we can easily compute the joint distribution

$$p(\rho_t = i, \varepsilon_t = j, \rho_{t+1} = k|\boldsymbol{z}_{1:N+1}) =$$

$$\frac{\alpha_i(t)p(\boldsymbol{z}_{t+2}, \varepsilon_t = j, \rho_{t+1} = k|\rho_t = i, \boldsymbol{z}_{t+1})\beta_k(t+1)}{\sum_l \alpha_l(t)\beta_l(t)},$$

where $p(\boldsymbol{z}_{t+2}, \rho_{t+1} = k, \varepsilon_t = j|\rho_t = i, \boldsymbol{z}_{t+1})$ is given by

$$p(\varepsilon_t = j, \rho_{t+1} = k|\rho_t = i, \boldsymbol{z}_{t+1})p(\boldsymbol{z}_{t+2}|\rho_{t+1} = k, \boldsymbol{z}_{t+1}),$$

and $p(\varepsilon_t = j, \rho_{t+1} = k|\rho_t = i, \boldsymbol{z}_{t+1})$ is given by

$$p(\varepsilon_t = j|\rho_t = i, \boldsymbol{z}_{t+1})p(\rho_{t+1} = k|\rho_t = i, \varepsilon_t = j, \boldsymbol{z}_{t+1}).$$

Having computed these probabilities, we can now proceed to the maximization step.

MAXIMIZATION STEP*:* In the maximization step of the EM algorithm, we compute the parameters that maximize the expected log-likelihood of the observed and hidden variables

$$\boldsymbol{\theta}' = \arg\max_{\boldsymbol{\theta}} \sum_{\rho,\varepsilon} p(\boldsymbol{\mathcal{H}}|\boldsymbol{z}_{1:N+1}; \boldsymbol{\theta}_{old}) \ln p(\boldsymbol{\mathcal{H}}, \boldsymbol{z}_{1:N+1}; \boldsymbol{\theta}),$$

where the hidden variables $\boldsymbol{\mathcal{H}}$ are given by $\boldsymbol{\mathcal{H}} = \{\rho_{1:N}, \varepsilon_{0:N-1}\}$, the summation is over all possible sequences of ρ and ε, and the conditional distributions $p(\boldsymbol{\mathcal{H}}|\boldsymbol{z}_{1:N+1}; \theta_{\text{old}})$ are computed using the old model parameters $\boldsymbol{\theta}_{\text{old}}$ as indicated. By factorizing the joint distribution $p(\boldsymbol{\mathcal{H}}, \boldsymbol{z}_{1:N+1}; \boldsymbol{\theta})$, decomposing the log of a product into a sum of logs, and marginalizing out variables, the maximization problem can be rewritten as

$$\boldsymbol{\theta}' = \arg\max_{\boldsymbol{\theta}} \sum_{t=1}^{N}\sum_{\rho_t} p(\rho_t|\boldsymbol{z}_{1:N+1};\boldsymbol{\theta}_{\text{old}}) \ln p(\boldsymbol{z}_{t+1}|\rho_t,\boldsymbol{z}_t;\boldsymbol{\theta})$$

$$+ \sum_{t=1}^{N}\sum_{\rho_t} p(\rho_t,\varepsilon_{t-1}{=}1|\boldsymbol{z}_{1:N+1};\boldsymbol{\theta}_{\text{old}}) \ln p(\rho_t|\varepsilon_{t-1}{=}1,\boldsymbol{z}_t;\boldsymbol{\theta})$$

$$+ \sum_{t=1}^{N-1}\sum_{\rho_t}\sum_{\varepsilon_t} p(\rho_t,\varepsilon_t|\boldsymbol{z}_{1:N+1};\boldsymbol{\theta}_{\text{old}}) \ln p(\varepsilon_t|\rho_t,\boldsymbol{z}_{t+1};\boldsymbol{\theta}).$$

The marginal distributions $p(\rho_t|\boldsymbol{z}_{1:N+1},\boldsymbol{\theta}_{\text{old}})$, $p(\rho_t,\varepsilon_{t-1} = 1|\boldsymbol{z}_{1:N+1};\boldsymbol{\theta}_{\text{old}})$, and $p(\rho_t,\varepsilon_t|\boldsymbol{z}_{1:N+1};\boldsymbol{\theta}_{\text{old}})$ are straightforward to compute from the joint distributions computed in the expectation step. The new parameters can then be computed for the phase- and state- transition distributions. For the state transition distribution, the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are computed using weighted linear regression. When learning the matrices $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$, the weight for sample $\{\boldsymbol{s}_t,\boldsymbol{a}_t,\boldsymbol{s}_{t+1}\}$ is given by $p(\rho_t = i|\boldsymbol{s}_{1:N+1},\boldsymbol{a}_{1:N},\boldsymbol{\theta}_{\text{old}})$. The phase transition parameters $\hat{\boldsymbol{\omega}}$ and $\check{\boldsymbol{\omega}}$ are computed using the weighted version of interatively reweighted least squares. When learning the phase termination model for phase $\rho = i$, the sample $\{\varepsilon_t,\boldsymbol{s}_t\}$ is weighted by $p(\rho_t = i,\varepsilon\,_t|\boldsymbol{s}_{1:N+1},\boldsymbol{a}_{1:N},\boldsymbol{\theta}_{old})$. Similarly, the sample $\{\rho_t,\boldsymbol{s}_t\}$ is weighted by $p(\rho_t,\varepsilon_t = 1|\boldsymbol{s}_{1:N+1},\boldsymbol{a}_{1:N},\boldsymbol{\theta}_{old})$ when learning the phase initiation model $p(\rho_t|\boldsymbol{s}_t,\varepsilon_{t-1} = 1)$. Regularization can be added to both linear regression and logistic regression to incorporate prior knowledge and avoid overfitting.

After the maximization step, the algorithm computes the expectation step again with the new parameters. This process iterates until the model has converged to a solution.

INITIALIZATION*:* The EM learning method converges to a local optimum and requires a suitable initialization. There are various potential methods for initializing the model and selecting the number of phases κ. We initialized the model using spectral clustering [8]. Given that phase transitions often correspond to the making or breaking of contacts [12], [19], the similarity between samples was computed using contact distribution kernels [24]. Thus, samples were clustered together if they had similar contacts between the hands and object, as well as the object and the table. The computation was based on point-cloud models of the objects, and a point was considered to be in contact if it was within 1cm of the other object. The clustering was performed with different numbers of clusters κ. The clustering with the highest intra-cluster and lowest inter-cluster similarities was used to initialize the model.

The samples assigned to a cluster were used to compute the initial parameters and hyperparameters for the corresponding phase's state-transition distributions, as well as the phase initiation distribution. This approach to initializing the model worked well in the experiments, and found the correct number of phases for the demonstrated task. The initial clustering was also used to create a set of κ contact features. One representative sample from each phase was selected to define the basis for a contact feature using the contact distribution kernel.

III. LEARNING MOTOR PRIMITIVE LIBRARIES FOR MULTI-PHASE TASKS

Having learned a multi-phase model, the next step is to learn movements for transitioning between the different phases. In this manner, the model decomposes the original task into a series of subtasks and learns a motor primitive policy for each subtask. The learned motor primitives should generalize between different scenarios and be reusable for different tasks. The process of learning a library of motor primitives is explained in Sections III-A and III-B.

A. Dynamic Motor Primitives

The robot's skills are represented using dynamic motor primitives (DMPs) [17], [33]. These motor primitive representations are easily adapted to different situations and can be learned in a straightforward manner [18], [33]. The robot learns one DMP for each of the phase transitions observed in the demonstrations. The shape of a DMP trajectory is defined by a set of shape parameters $\boldsymbol{w}$, and a goal state $\boldsymbol{g}$.

Our initial approach learned a mapping directly from the object's position to the parameters of the motor primitives. However, this method ignored the geometry of the object, which lead to crude grasps. The DMPs are therefore now defined in task frames that generalise between object shapes.

Computing a suitable task frame is similar to the grasp synthesis problem [5]; i.e. the robot needs to select a hand pose for transitioning to the next phase. Although this pose is not the goal state, it should allow the hand to establish suitable contacts with the object. In our experiments, the current hand pose was used as the task frame if the hand was already in contact with the object or making contact would decrease the likelihood of transitioning to the next phase. Otherwise, the robot computed the task frame by sampling different hand poses and evaluating them based on their contact features. The hand pose with the highest likelihood of transitioning to the goal phase was then selected as the task frame. The robot also tested if breaking contact between the object and the table increased the likelihood. In the future, one could extend this approach to sampling different object poses in the scene.

The DMPs are incorporated into the model as shown in Fig. 2. At time step t, the robot is executing motor primitive $\mathcal{M}_t$, which includes the linear system as well as the canonical system. The robot performs an action $\boldsymbol{a}$ according to the DMP's desired trajectory and the current state $p(\boldsymbol{a}_t|\mathcal{M}_t,\boldsymbol{s}_t)$. Once a DMP has finished, a new DMP is selected according to the current state $p(\mathcal{M}_t|\mathcal{M}_{t-1},\boldsymbol{s}_t)$. The graphical model illustrates the complementary nature of motor primitives and phases.

B. Learning Motor Primitives for Phase Transitions

Having defined a task frame, the next step is to learn the goal state $\boldsymbol{g}$ and shape parameters $\boldsymbol{w}$. As the purpose of each DMP is to bring the robot from one phase to another, the

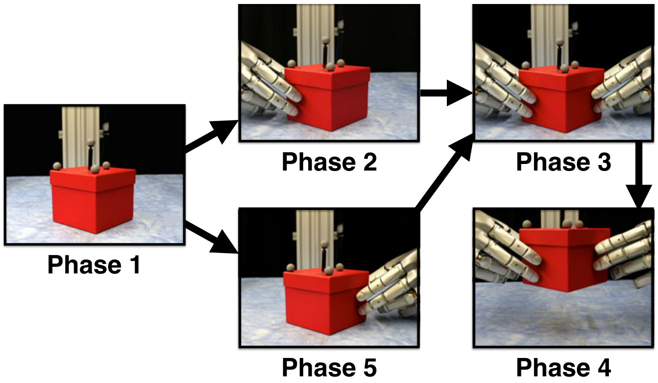

Fig. 3. The five phases detected in the task demonstrations. The pictures show the contacts between the objects in each of the phases. The arrows show the phase transitions that were observed during the demonstrations. The robot learns a motor primitive for each of these transitions.

robot should optimize the parameters accordingly. Hence, the phase transition distribution $p(\rho_t|\rho_{t-1}, \boldsymbol{s}_t)$ is used to define a reward function for transitioning between different phases. The parameters of the DMP are then learned using relative entropy policy search (REPS) [31].

In order to learn motor primitives for transitioning from phase ρ_0 to ρ_g, we define the reward at each time step as the probability of transitioning to the goal phase $p(\rho_{t+1} = \rho_g|\boldsymbol{s}_{t+1}, \rho_t)$. This reward is discounted over time by the probability of transitioning to a phase other than ρ_0 or ρ_g. Thus, the reward function directs the robot towards the goal phase's conditions, while avoiding other phases. A squared cost term was also applied to penalize large actions as well as large deviations of the goal state from the task frame's origin to keep the goal grounded. By including smooth features for modelling the phase transition distribution $p(\rho_t|\rho_{t-1}, \boldsymbol{s}_t)$, the reward function can guide the robot during the learning process. This reward function worked well in the experiments, but other reward functions could also be constructed using the phase transition distribution.

Given the reward function and a set of starting states, the parameters $\boldsymbol{g}$ and $\boldsymbol{w}$ of the the DMP are learned using episodic REPS. The robot begins by learning the goal parameters $\boldsymbol{g}$, which are defined relative to the task frame. The distribution over the parameters is modeled as a Gaussian $\boldsymbol{g} \sim \mathcal{N}(\boldsymbol{\mu}_{g0}, \boldsymbol{\Sigma}_{g0})$, where the initial mean $\boldsymbol{\mu}_{g0}$ is given by the origin of the task frame. Parameters are sampled from the distribution and evaluated for each starting state using the learned multi-phase model. The rewards are averaged over the starting states.

After evaluating multiple samples of parameter sets from the current policy $\mathcal{N}(\boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g)$, a policy update is performed to determine a new policy $\mathcal{N}(\boldsymbol{\mu}'_g, \boldsymbol{\Sigma}'_g)$. REPS computes a new policy that maximizes the expected reward, while limiting the Kullback Leibler divergence between the old and new policies $\epsilon \geq D_{\mathrm{KL}}(\mathcal{N}(\boldsymbol{\mu}'_g, \boldsymbol{\Sigma}'_g)||\mathcal{N}(\boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g))$. Limiting the KL divergence between the policies makes learning the policy more robust to noisy rewards, and the robot is less likely to converge prematurely to a poor local optimum. For more details on REPS, we refer the reader to the survey of Deisenroth et al. on policy search methods in robotics [11]. The shape parameters $\boldsymbol{w}$ were learned in the same manner using REPS. The initial mean parameters $\boldsymbol{\mu}_{w0}$ were learned from the demonstrations using imitation learning [18].

Once a motor primitive $\mathcal{M}$ has been learned, it can be executed using the learned model to get the starting states for the next phase. This process is repeated until motor primitives have been learned for all of the phase transitions. The model can potentially also be used to learn motor primitives for specific tasks by defining a suitable reward, e.g., moving to a goal while remaining in the same phase.

C. Learning Policies for Sequencing Motor Primitives

Given a library of DMPs, the robot must now learn high-level policies for sequencing these motor primitives $\mathcal{M}$ to perform different manipulation tasks. We assume that the task is defined by an expected reward function $r(\tilde{\boldsymbol{s}}, \mathcal{M})$, where $\mathcal{M}$ is the executed motor primitive, and the extended state $\tilde{\boldsymbol{s}}$ includes the state $\boldsymbol{s}$ and the robot's estimate of the phase $\tilde{\rho}$. The robot's high-level policy $\pi(\mathcal{M}|\tilde{\boldsymbol{s}})$ selects motor primitives $\mathcal{M}$, according to the current state and phase. The motor primitives should be selected such that they maximize the reward accumulated over time

$$\max_{\pi} \sum_{t=1}^{\infty} \gamma^t r(\tilde{\boldsymbol{s}}(t), \mathcal{M}(t)),$$

where γ is a discount factor on future rewards $0 \leq \gamma < 1$ and t indicates the steps in the motor primitive sequence. In our experiments, all of the motor primitives had the same duration. However, one could also use different discount factors for motor primitives with different durations.

The robot uses a policy iteration approach to learn the high-level policy [36]. This approach iterates between computing a value function $V^{\pi}(\tilde{\boldsymbol{s}})$ for the current policy and improving the policy according to the value function. The value function $V^{\pi}(\tilde{\boldsymbol{s}})$ is defined as the expected future rewards when in state $\boldsymbol{s}$ and phase ρ, and following policy π. We estimate the value function using non-parametric dynamic programming (NPDP) [23]. The robot first computes a set of m prototypical samples $\{\tilde{\boldsymbol{s}}_i\}$ for $i \in \{1, ..., m\}$. Given a set of starting states, the prototype samples can be obtained by sampling different sequences of motor primitives using the multi-phase model. In our experiments, we sampled every sequence of motor primitives using the maximum-likelihood state transitions. The estimated value function has the form

$$V(\tilde{\boldsymbol{s}}) = \frac{\sum_{i=1}^{m} \theta_i \phi\left(\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{s}}_i\right)}{\sum_{j=1}^{m} \phi\left(\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{s}}_j\right)},$$

where we model $\phi\left(\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{s}}_i\right)$ using a squared exponential kernel for the state $\boldsymbol{s}$ and multiply it by a Kronecker delta function for the phase $\tilde{\rho}$. The parameters $\boldsymbol{\theta}$ of the value function are given by $\boldsymbol{\theta} = (\boldsymbol{I} - \gamma\boldsymbol{\lambda})^{-1}\bar{\boldsymbol{r}}$, where the ith element of $\bar{\boldsymbol{r}}$ is the expected reward $[\bar{\boldsymbol{r}}]_i = r(\tilde{\boldsymbol{s}}_i, \mathcal{M}_i)$, and the elements of the transition matrix $\boldsymbol{P}$ are defined as

$$[\boldsymbol{\lambda}]_{ij} = \int \frac{\phi\left(\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{s}}_j\right) \psi_i\left(\tilde{\boldsymbol{s}}\right)}{\sum_{k=1}^{m} \phi\left(\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{s}}_k\right)} \mathrm{d}\tilde{\boldsymbol{s}}.$$

The function $\psi_i\left(\tilde{\boldsymbol{s}}\right)$ is the distribution over next states $\tilde{\boldsymbol{s}}'$ after executing motor primitive $\mathcal{M}_i$ from state $\tilde{\boldsymbol{s}}_i$. The integral is computed by drawing samples from the multi-phase model to approximate the function $\psi_i(\tilde{\boldsymbol{s}})$. Even though a DMP is meant to transition to a specific phase, some of the samples may not reach this desired phase due to the stochasticity of the multi-phase model. The model thus also incorporates the failure rates of the DMPs, and the robot can learn a high-level policy that avoids motor primitives that tend to fail.

Given the value function $V(\tilde{\boldsymbol{s}})$, the policy is updated by selecting a new motor primitive for each of the prototypical samples. For each of these samples, the robot selects the motor primitive that maximizes the expected immediate reward plus the expected discounted value for the next state

$$\mathcal{M}_i^{\text{new}} = \arg\max_{\mathcal{M}} \mathbb{E}(r(\tilde{\boldsymbol{s}}_i, \mathcal{M}) + \gamma V(\tilde{\boldsymbol{s}}_i')).$$

The rows of the matrix $\boldsymbol{\lambda}$ and the vector $\bar{\boldsymbol{r}}$ are then updated accordingly, and a new value function is computed. This policy iteration process is repeated until the value function and the policy converge. The resulting policy is given by

$$\pi(\mathcal{M}|\tilde{\boldsymbol{s}}) = \frac{\sum_{i=1}^{m} \varphi(\mathcal{M}, \mathcal{M}_i)\, \phi(\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{s}}_i)}{\sum_{k=1}^{m} \phi(\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{s}}_k)},$$

where $\varphi(\mathcal{M}, \mathcal{M}_i)$ is a Kroenecker delta function. The policy has a similar form to the multi-class classifier policies that are commonly used by imitation learning approaches.

IV. Evaluations

The proposed method was evaluated on a bimanual grasping task. The robot consists of two Kuka light-weight robot arms, and two five-fingered DLR hands [7].The robot's arms were controlled using cartesian impedance control. The object was tracked using a marker-based Optitrak system. The box is too large to grasp with a single robot hand.

A. Setup and Model Learning

The robot was given two demonstrations of a bimanual grasping task using kinaesthetic teaching. In the first demonstration, the robot's right hand made contact with the box first. In the second demonstration, the left hand made the first contact with the object. In both demonstrations, the object was subsequently grasped with both hands and lifted up from the table. The trajectories were sampled at 10 Hz. When the object was placed at the same position and orientation, the robot performed the task by replaying the demonstrated trajectory. However, the robot failed to grasp the box if it was placed at a different location on the table. The trajectories were segmented into five phases, as illustrated in Fig. 3, using the method described in Section II.

For learning the multi-phase model, the state $\boldsymbol{s}$ includes the position and orientation of the box, the positions of the robot's hands, and the contact features. The joints of the robot's fingers have encoders and torque sensors. Rather than using the joint angles and torques directly, principal component analysis was used to reduce the dimensionality of the data from 60 to eight dimensions, i.e., two dimensions for positions and two dimensions for torques for each hand. The state transition distributions learned the change in state rather than the absolute state, except for the contact features. The actions $\boldsymbol{a}$ were defined by the change in the desired trajectory. The robot used an impedance controller to follow the desired trajectory. The features for the phase transition distributions include the distances between the box and the finger tips, the distance between the box and the table, and the joint angle and torque data. The contact features were also used for computing the phase initiation distribution. The proposed entry and exit version of the STARHMM allowed the robot to share initiation data for phase transitions $2 \rightarrow 3$ and $5 \rightarrow 3$, and termination data for $1 \rightarrow 2$ and $1 \rightarrow 5$.

B. Learning Motor Primitives for Phase Transitions

Using the method described in Section III, the robot learned a library of five motor primitives corresponding to the five phase transitions observed in the demonstrations. Three different approaches for learning DMP libraries were evaluated for comparison. The first approach used only imitation learning. The goal states were set to the origins of the task frames, except for the $3 \rightarrow 4$ phase transition, for which the goal states were set to 10 cm above the task frame. The second approach learned the DMP parameters using the policy search method described in Section III. The third approach also used reinforcement learning, but the stochasticity of the model was removed by using the MAP state transitions and keeping the phase fixed. In this manner, the model provided the robot with a deterministic reward. For both model-based reinforcement learning approaches, the bound was set to $\epsilon = 0.5$, and the goal states $\boldsymbol{g}$ were learned using 10 policy updates of 50 episodes. The weights $\boldsymbol{w}$ were learned using 10 policy updates of 100 episodes each.

The robot executed the motor primitives according to the phase transition sequences observed in the demonstrations, i.e., $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ and $1 \rightarrow 5 \rightarrow 3 \rightarrow 4$. Each sequence of motor primitives was executed 25 times using each of the three approaches, giving a total of 450 motor primitive executions. At the start of each sequence, the box was placed at a random position and orientation on the table in a 50 cm $\times$50 cm region within the robot's workspace. The successes and failures of the motor primitives were labelled by hand. A transition to phases 2 or 5 was considered successful if the robot's right or left hand made contact with the object. A transition to phase 3 was considered a success if the hands were touching opposite sides of the box, and the box was in contact with the table. A transition to phase 4 was considered a success if the robot succeeded in lifting the box from the table with both hands for more than 15 seconds.

The results of the experiment are shown in Fig. 4. Both reinforcement learning approaches succeeded in lifting the box in more than 90% of the trials. Using imitation learning, without additional reinforcement learning resulted in a success rate of only 38%. These motor primitives often resulted in a single finger tip making light contact with the object, which lead to delicate grasps. In comparison, the motor primitives learned using reinforcement learning pushed the box a few centimeters, which resulted in the object becoming aligned with the hand. The motor primitives for transitioning from phase 3 to 4 tended to apply less horizontal force to

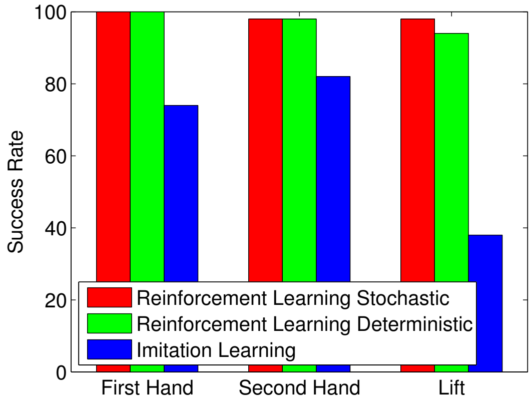

Fig. 4. The success rates of the motor primitive evaluations. The robot performed the sequences of three motor primitives (first hand, second hand, and then lift) as observed in the human demonstrations. The different colored bars indicate different approaches to learning the motor primitives.

the object at the end of the motion than at the start. This may be a result of the demonstrations, which also tended to apply less force once the object was lifted from the table.

The results of the experiment show that the robot can learn to transition between phases more reliably by optimizing the motor primitives according to the learned phase transition model. The multi-phase model captured a sufficient amount of detail to allow the robot to learn the required motor primitives. Determining the number of phases is still an open problem, which could be addressed by using a suitable prior [6], [14], [29], or by first learning the coarse phase structure using simulation data. The forward model could also be improved by explicitly enforcing each phase's contact constraints, or using Gaussian processes with phase-specific hyperparameters to model the state transition distributions.

The robot's ability to generalize the motor primitive between different objects was evaluated by attempting to grasp six novel objects (shown in Fig. 5) using the $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ phase sequence. Two out of three grasps failed for the bottle, as it tended to topple rather than slide across the table. Two of the grasps of the green box also failed as a result of low-friction contacts between metal and metal. The grasps of the other objects all succeeded. The results show that the learned motor primitives are not applicable to objects with distinct dynamics or friction properties, but they do generalize between objects with different shapes.

C. Learning Policies for Sequencing Motor Primitives

The goal of the second experiment was to learn high-level policies for sequencing motor primitives. Two task motor primitives were added to the library: moving both hands to the left 10 cm and raising both hands by 10 cm. Unlike the motor primitives for transitioning between phases, these task motor primitives can be executed from any phase. Executing a task motor primitive marked the end of a trial, which was modeled by a special absorbing state. Thus, the robot had to learn a high-level policy for reaching a suitable phase and then executing the correct task motor primitives.

The high-level policy was learned as described in Section III-C. The phase estimates $\tilde{\rho}$ were computed using the model's phase transition distribution $p(\rho_t | \boldsymbol{s}_t, \rho_{t-1})$ and the trajectory from the previous motor primitive. The kernel function $\phi\left(\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{s}}_i\right)$ was computed using the 3D position of the box and the positions of the hands relative to the box. For this evaluation, the robot always selected the most likely next motor primitive $\arg\max_{\mathcal{M}} \pi(\mathcal{M} | \tilde{\boldsymbol{s}})$. The robot computed 137 prototype samples $\tilde{\boldsymbol{s}}_i$ based on 20 start states.

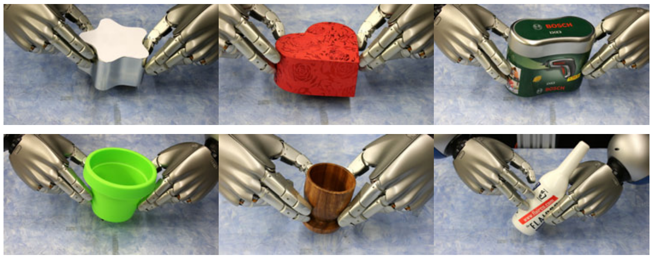

Fig. 5. The figure shows the ability of the the learned DMPs to adapt to the geometry of different objects. Two out of three grasps of the green box and the bottle failed. The other fourteen grasps lead to successful lifts.

For the first task, the robot was given a reward for the final height of the box and a penalty for the left-right deviation of the box from the center of the table. The discount factor was set to $\gamma = 0.99$. The task was executed 20 times on the robot. The box was placed on the left side of the table for ten of the trials, and on the right side for the other ten trials. In all of the trials, the robot grasped the box with both hands and successfully lifted it off of the table. When the box was placed on the left side of the table, the robot always approached the box with the left hand first, as shown in Fig. 6. When the box was placed on the right side of the table, the robot approached the box first with the right hand in nine of the ten trials. In this manner, the robot tended to push the box towards the center of the table before lifting it. In two of the trials, the robot failed to detect the $3 \rightarrow 4$ phase transition. These errors seems to be a result of the robot using the back of the thumb to hold the box, which pushed the fingers together rather than apart. This problem could be addressed by representing the forces in task-space.

For the second task, the robot was given a reward for quickly moving the box to the left, and a penalty for the height of the box. The discount factor was set to $\gamma = 0.95$. The task was again executed 20 times on the robot with the box placed at different locations on the table. In all of the trials, the robot placed its right hand on the box and then moved both hands to the left. The robot thus learned to exploit a two-handed motor primitive to perform a one-handed manipulation.

The results of the experiment demonstrated that the proposed value function approach is suitable for creating medium-length sequences of DMPs. The first task showed that the robot was able to reconstruct the original sequences of phase transitions and then execute the task motor primitive. The robot additionally learned that it could exploit the DMPs to push the box towards the center of the table for a higher reward. The second task showed that the robot could learn to create new sequences by reusing DMPs from the demonstrated task. The experiments also showed that the robot could use the model of the phases to determine the effects of applying the task motor primitives in each phase.

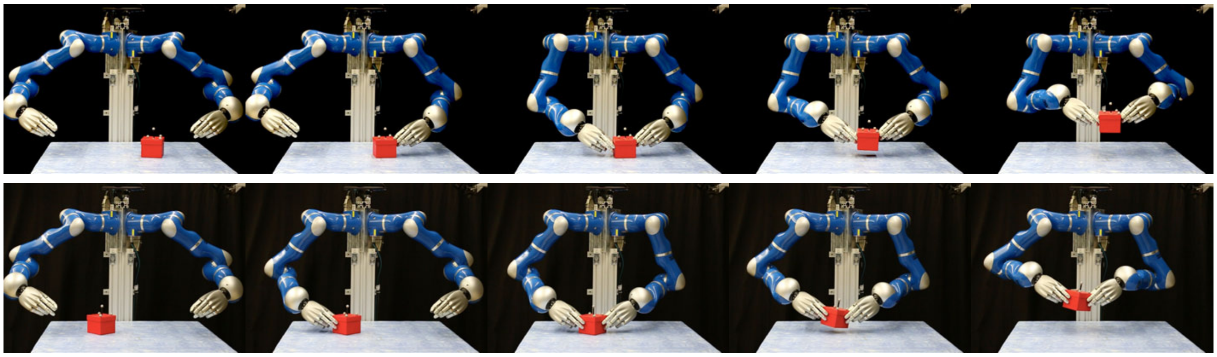

Fig. 6. The images show two sequences of the bimanual grasping task. In the top row, the box was placed towards the left, and the high-level controller approached the box first with the left hand. In the bottom row, the robot chose to approach with the right hand first, as the box was located on the right.

V. Conclusion

We proposed a method for learning hierarchical manipulation skills that exploit the phase structure of tasks. The robot first uses a STARHMM to extract the phase structure, which defines a set of subtasks. The robot then learns a library of DMPs for performing these subtasks using imitation and reinforcement learning. Finally, the robot learns a high-level policy for selecting motor primitives using policy iteration.

The proposed approach was successfully evaluated on a bimanual grasping task. The experiments showed that the learned DMPs reliably transition between different phases, generalize between objects with different shapes, and can be reused between tasks with similar phase transitions.

VI. Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7-ICT-2013-10) under grant agreements 610878 (3rdHand) and 610967 (TACMAN).

References

[1] S. Andrews and P.G. Kry. Goal directed multi-finger manipulation: Control policies and analysis. *Computers and Graphics*, 2013.

[2] D. Barber. Expectation correction for smoothed inference in switching linear dynamical systems. *JMLR*, 2006.

[3] J. Barry, L. P. Kaelbling, and T. Lozano-Pérez. A hierarchical approach to manipulation with diverse actions. In *ICRA*, 2013.

[4] L. E. Baum. An equality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 1972.

[5] J. Bohg, A. Morales, T. Asfour, and D. Kragic. Data-driven grasp synthesis - a survey. *TRo*, 2014.

[6] J. Butterfield, S. Osentoski, G. Jay, and O.C. Jenkins. Learning from demonstration using a multi-valued function regressor for time-series data. In *Humanoids*, 2010.

[7] Z. Chen, N. Y. Lii, T. Wimboeck, S. Fan, M. Jin, C. Borst, and H. Liu. Experimental study on impedance control for the five-finger dexterous robot hand dlr-hit ii. In *IROS*, 2010.

[8] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[9] C. Daniel, G. Neumann, O. Kroemer, and J. Peters. Learning sequential motor tasks. In *ICRA*, 2013.

[10] T. Debus, P. E. Dupont, and R. D. Howe. Contact state estimation using multiple model estimation and hidden markov models. In *ISER*, 2002.

[11] M. P. Deisenroth, G. Neumann, and J. Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2013.

[12] J. R. Flanagan, M. C. Bowman, and R. S. Johansson. Control strategies in object manipulation tasks. *Curr Opin Neurobiol*, 2006.

[13] O. Fuentes, R.P.N. Rao, and M. Van Wie. Hierarchical learning of reactive behaviors in an autonomous mobile robot. In *International Conference on Systems, Man and Cybernetics*, 1995.

[14] D.H. Grollman and O.C. Jenkins. Incremental learning of subtasks from unsegmented demonstration. In *IROS*, 2010.

[15] S. Hart and R. Grupen. Learning generalizable control programs. *TAMD*, 2011.

[16] K. Hauser and V. Ng-Thow-Hing. Randomized multi-modal motion planning for a humanoid robot manipulation task. *IJRR*, 2011.

[17] A. Ijspeert, J. Nakanishi, and S. Schaal. learning attractor landscapes for learning motor primitives. In *NIPS*, 2003.

[18] J. A. Ijspeert, J. Nakanishi, and S. Schaal. movement imitation with nonlinear dynamical systems in humanoid robots. In *ICRA*, 2002.

[19] R. S. Johansson and J. R. Flanagan. Coding and use of tactile signals from the fingertips in object manipulation tasks. *Nat Rev Neurosci*, 2009.

[20] J. Kober and J. Peters. Learning elementary movements jointly with a higher level task. In *IROS*, 2011.

[21] G.D. Konidaris, S.R. Kuindersma, R.A. Grupen, and A.G. Barto. Robot learning from demonstration by constructing skill trees. *IJRR*, 2012.

[22] M. Koval, N. Pollard, and S. Srinivasa. Pre- and post-contact policy decomposition for planar contact manipulation under uncertainty. In *R:SS*, 2014.

[23] O. Kroemer and J. Peters. A non-parametric approach to dynamic programming. In *NIPS*, 2011.

[24] O. Kroemer and J. Peters. Predicting object interactions from contact distributions. In *IROS*, 2014.

[25] O. Kroemer, H. van Hoof, G. Neumann, and J. Peters. Learning to predict phases of manipulation tasks as hidden states. In *ICRA*, 2014.

[26] S. Levine, N. Wagener, and P. Abbeel. Learning contact-rich manipulation skills with guided policy search. In *ICRA*, 2015.

[27] F. Meier, E. Theodorou, F. Stulp, and S. Schaal. Movement segmentation using a primitive library. In *IROS*, 2011.

[28] J. Mugan and B. Kuipers. Autonomous learning of high-level states and actions in continuous environments. *TAMD*, 2012.

[29] S. Niekum, S. Chitta, B. Marthi, S. Osentoski, and A. G. Barto. Incremental semantically grounded learning from demonstration. In *R:SS*, 2013.

[30] P. Pastor, M. Kalakrishnan, L. Righetti, and S. Schaal. Towards associative skill memories. In *Humanoids*, 2012.

[31] J. Peters, K. Muelling, and Y. Altun. Relative entropy policy search. In *AAAI*, 2010.

[32] J. Romano, K. Hsiao, G. Niemeyer, S. Chitta, and K. J. Kuchenbecker. Human-inspired robotic grasp control with tactile sensing. *TRo*, 2011.

[33] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. learning movement primitives. In *ISRR*, 2004.

[34] V. Soni and S. Singh. Reinforcement learning of hierarchical skills on the sony aibo robot. In *ICDL*, 2006.

[35] F. Stulp, E. Theodorou, and S. Schaal. Reinforcement learning with sequences of motion primitives for robust manipulation. *TRo*, 2012.

[36] R. S. Sutton and A. G. Barto. *Reinforcement Learning an Introduction*. The MIT Press, 2000.

[37] C. K. Tham. Reinforcement learning of multiple tasks using a hierarchical CMAC architecture. *Robotics and Autonomous Systems*, 1995.

[38] M. Waechter, S. Schulz, T. Asfour, E. Aksoy, F. Woergoetter, and R. Dillmann. Action sequence reproduction based on automatic segmentation and object-action complexes. In *Humanoids*, 2013.