Learning Movement Primitive Attractor Goals and Sequential Skills from Kinesthetic Demonstrations

Simon Manschitz[a,b,*], Jens Kober[c], Michael Gienger[b], Jan Peters[a,d]

[a] *Institute for Intelligent Autonomous Systems, Technische Universität Darmstadt, 64289 Darmstadt, Germany*
[b] *Honda Research Institute Europe, 63073 Offenbach, Germany*
[c] *Delft Center for Systems and Control, Delft University of Technology, 2628 CD Delft, The Netherlands*
[d] *Max Planck Institute for Intelligent Systems, 72076 Tübingen, Germany*

Abstract

We present an approach for learning sequential robot skills through kinesthetic teaching. In our work, finding the transitions between consecutive movement primitives is treated as multiclass classification problem. We show how the goal parameters of linear attractor movement primitives can be learned from manually segmented and labelled demonstrations and how the observed movement primitive order can help to improve the movement reproduction. The improvement is achieved by restricting the classification result to the currently activated movement primitive and its possible successors in a graph representation of the sequence, which is also learned from the demonstrations. The approach is validated with three experiments using a Barrett WAM robot.

Keywords: human-robot interaction, kinesthetic teaching, learning from demonstration, programming by demonstration

1. Introduction

Adapting skills to new situations is arguably one of the key elements for robots to become more autonomous. Learning from demonstration (LFD) or imitation learning therefore has received a lot of attention in robotics research in the past years. The goal of LFD is to learn skills based on demonstrations of a teacher [1]. While most work in this domain concentrates on learning single movement skills, sequencing such learned skills in order to perform more sophisticated tasks is still an open research topic. There are two cases where such sequential skills are particularly useful. First, there are tasks which are not representable in a non-sequential way at all. As an example, consider a robot standing in front of a door. Without any additional knowledge, the system does not know whether the robot has to open the door or if the robot just closed it. The reason is that the same state is perceived for both options. This problem is often referred to as perceptual aliasing [29]. Dissolving perceptual aliasing requires either the previous movement history to be encoded in the perceived state or a policy which activates movements based on the history of movements. Such a policy is what we call a sequential skill. Second, even though a task may be representable using a single movement, it may be beneficial to decompose it into smaller (sub-)tasks first. Such a decomposition bounds the complexity of each (sub-)task and the resulting movements are often more intuitive and easier to learn.

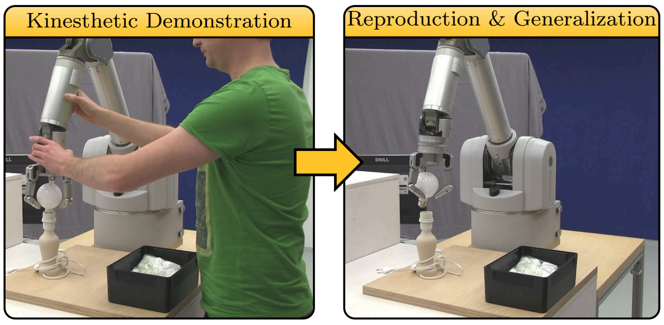


Figure 1: The system is supposed to learn how to unscrew a light bulb from kinesthetic demonstrations. We evaluate our approach on this example using a real seven degrees of freedom (DOF) Barrett WAM robot with a four DOF hand.

We aim at learning sequential skills where the currently activated movement cannot be solely determined from the perceived state, but may also depend on the history of movements. The goal is to learn when to activate each movement, based on kinesthetic demonstrations. Kinesthetic teaching is a widely used teaching method in robotics. Here, a teacher guides a robot through movements by physically moving the robot's arm, similar to parents teaching tasks to their children. (see Figure 1).

*Corresponding author
Email addresses: manschitz@ias.tu-darmstadt.de (Simon Manschitz), j.kober@tudelft.nl (Jens Kober), michael.gienger@honda-ri.de (Michael Gienger), mail@jan-peters.net (Jan Peters)

1.1. Related Work

Single elementary movements are often referred to as movement primitives (MPs) in literature [10, 26]. The traditional way of sequencing MPs was inspired by the subsumption architecture [3], where the behavior of a system is represented by a hierarchy of sub-behaviors. A sequential skill is usually composed by a two-level hierarchy, whereby the lower-level MPs are activated by an upper-level sequencing layer. The sequencing layer is usually modeled as graph structure, finite state machine (FSM) or Petri net and the activation of a MP is interpreted as discrete event in a continuous system [25, 24, 6]. An alternative view is treating the overall system as continuous entity. For example, Luksch et al. [14] model a sequence with a recurrent neural network. In that architecture, MPs can be concurrently active and inhibit each other. Therefore the sequence is defined implicitly. Although this structure leads to very smooth movements, the model is hard to learn and has to be defined mostly by hand.

Most concepts for sequencing MPs concentrate either on segmenting demonstrations into a set of MPs and/or on learning the individual MP parameters [21, 7, 27, 18]. Reproducing a sequence of learned MPs then serves as proof of concept for the segmentation. The actual sequence is not so important here, therefore the MPs are chosen randomly or are the same as in the demonstrations [13, 15]. The transition behavior between MPs is also either deterministic (e.g., the succeeding movement depends only on the previous movement) or not learned at all [11]. For triggering transitions, often subgoals or sequential constraints of a task are used [9, 19]. Sequential constraints (e.g., subtask A has to be executed before subtask B) can also be used to extract symbolic descriptions of tasks [22, 28, 12]. Such a description implicitly determines the MP sequence and is often intuitive. Indeed, symbolic approaches can perform sufficiently well for predetermined settings. However, they lack generality as they rely on predefined assumptions about the tasks. If these assumptions do not fully apply, they are likely to bias the system towards suboptimal decisions. Therefore probabilistic methods have become more popular, as they allow for a better generalization.

In [23], a nearest neighbor classifier is used to decide which MP to activate when the current movement has finished. Butterfield et al. [4] use a hierarchical Dirichlet process hidden Markov model as classification method for determining the next MP based on the sensor information and current MP. Niekum et al. [20] segment a demonstration with a beta process auto-regressive hidden Markov model in a set of MPs and build a FSM on the sequential level. The transition behavior is learned with k-nearest neighbor classification. The focus of our work lies on incorporating several demonstrations with varying MP sequences into one model of a task and learning the transition behavior between succeeding MPs. Basis for learning are the manually segmented and labeled sensor data traces from a set of kinesthetic demonstrations.

1.2. Proposed Approach

In this paper, a MP is a dynamical system (DS) with a linear attractor behavior. A detailed description of the underlying MP framework can be found in [14]. Please note, however, that our methods are kept general and that they should be applicable to arbitrary MP frameworks and feature sets. Each MP has a goal $\boldsymbol{s}_{\mathrm{g}}$ in task space coordinates that should be reached if it is activated. A goal can be a desired position of a robot body, joint angle, force or a combination thereof and can be defined relative between bodies using reference frames. MPs may be terminated before their goal is reached, for example, if a sensor reading indicates to the system that an obstacle is close to the robot. More generally, the transition behavior can be triggered based on the state of a feature set denoted as $\boldsymbol{x}_i$, with i indicating the time step. The features are not global but assigned to MPs, leading to one feature vector $\boldsymbol{x}_i^{(k)}$ per MP p_k. We assume a predefined set of K MPs denoted as $P = \{p_1, p_2, ..., p_K\}$. All parameters of each MP in the library (such as the reference frames) are known, but the attractor goals are not.

Similar to most other approaches, the transition behavior between MPs is considered to be discrete in this paper. Therefore, only one MP is active at a time. At every time step, the system has to decide which MP to activate. A straightforward way of applying machine learning methods to this problem would be training a single classifier with the labeled demonstration data. The skill could be subsequently reproduced by choosing the classification result for the current feature values as next activated MP. Nevertheless, complex skills involve many different MPs and due to perceptual aliasing between the different movements the classification may yield unsatisfying results. As the number of MPs grows, resolving the perceptual aliasing with a better set of hand-crafted features for the classification becomes intractable.

Our proposed approach consists of three stages as depicted in Figure 2. In the first stage, the goal parameters of the individual MPs are learned from the demonstration data (Section 2). In the second stage, a representation of the demonstrated sequences is learned by connecting the observed MPs in a graph (Section 3). Each node in the graph corresponds to a MP and each transition leads to a potentially succeeding MP. In the final stage, the MP transition behavior is learned (Section 4). One classifier is linked to each node in the graph. The task of the classifier is to decide when to transition to a new state in the graph during the reproduction of a skill, resulting in an activation of a different MP. These decisions are made based on the current state of the robot and its environment. The state is based on a set of features which are computed from raw sensor values. The graph structure therefore helps to improve the classification, as the overall classification problem is split into many smaller problems which may be easier to solve. Here, the reduced difficulty is due to the restriction of the possible classification results, which leads to

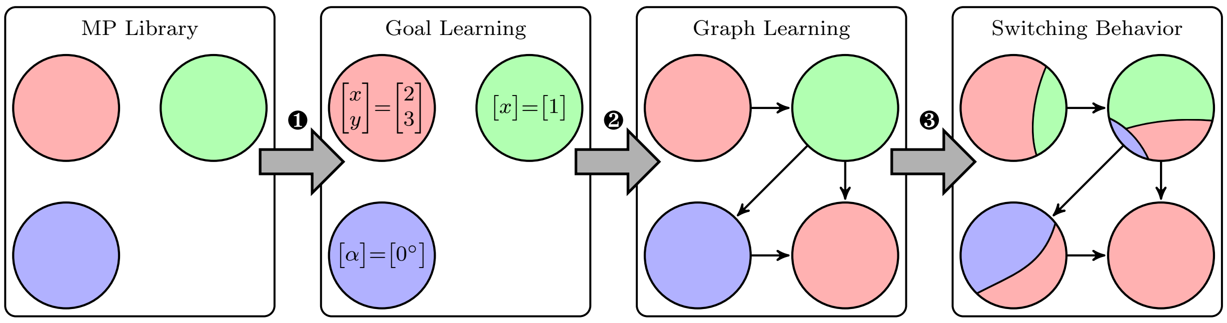


Figure 2: Overview of the learning approach. The approach starts with a predefined movement primitive (MP) library. All parameters of each MP in the library (such as the reference frames) are known, but the attractor goals such as a position x/y or an angle α are not. From the kinesthetic demonstrations, first these goals ❶ are learned. Next, a graph representation of the demonstrated sequences ❷ is learned. The graph is called sequence graph and determines possible successors of a MP during reproduction. Finally, the transition behavior is learned by training one local classifier for each node in the graph ❸.

less perceptual aliasing. An experimental validation of the approach is presented in Section 5, followed by a conclusion and a short outlook on future work in Section 6. Graph and transition behavior learning have been previously presented at two conferences [16, 17]. In this paper, the approach is extended by learning the MP goal parameters and evaluated in substantially more experiments.

2. Learning Movement Primitive Parameters

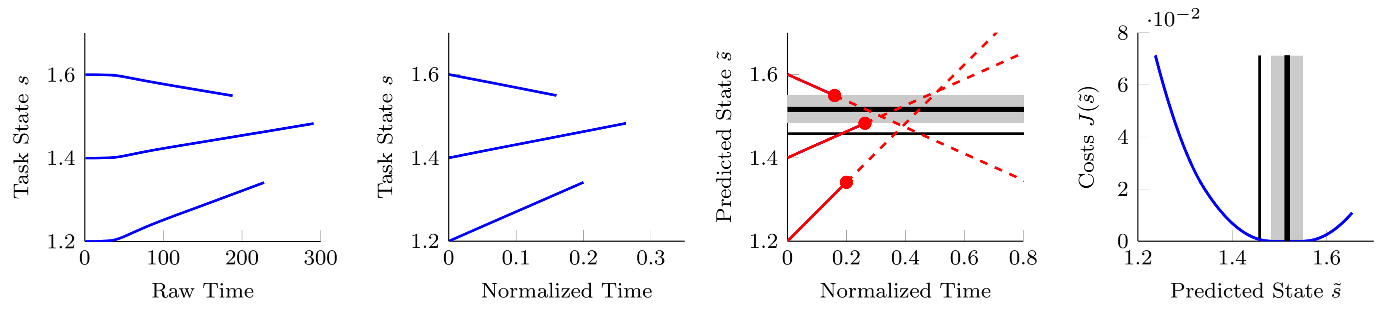
Learning the parameters of MPs directly from the demonstrations is crucial in order to avoid tedious parameter tuning. In our case, the parameters are the attractor goals of each MP. Even when segmenting and labeling demonstrations manually, extracting the goals is often not straightforward. One reason is that MPs do not always reach their goal. For example, in a reaching movement that has to be stopped due to a collision with an obstacle, the robot is usually quite far away from the MP's goal. If the attractor goals are not known beforehand, it is often not clear from the demonstrations whether a MP was stopped as the goal was reached or for any other reason. Hence, a simple solution such as taking the mean of all end points is often insufficient for learning the goals. In general, there are two possible ways of handling incomplete movements.

1. Ignore them for goal learning.
2. Predict the goal regardless of completeness.

The first approach requires a method for detecting incomplete movements, e.g., by clustering the end points of each MP activation and detecting outliers. Clustering usually requires a lot of data. Our goal is to learn from as few demonstrations as possible, as we do not want to overload the demonstrator by requiring tens of demonstrations. Therefore, we take the more sample efficient second approach and predict the goal without detecting and discarding incomplete movements. We argue that even if a MP is stopped prematurely, the movement up to that point still roughly points towards the MP's attractor goal. Our approach utilizes this information as follows. First, all trajectories are approximated by linear functions of time. For each trajectory, the goal is expected to lie on its future path, which is predicted using the linear functions. The goal is subsequently found by arbitrating between all expected goals and finding the best compromise between them. In the following sections, we will first show how a trajectory is approximated and then how the goal is learned. An overview of the goal learning is shown in Figure 3.

2.1. Trajectory Approximation

Before learning the attractor goals, we assume that the demonstrations have been segmented and labelled, resulting in a set of k different MPs. As the goals are learned for each MP separately, the MP index k will be omitted in the following. Each trajectory consists of a set of pairs $\{t_i, \boldsymbol{s}_i\}$, where t_i is the time and $\boldsymbol{s}_i \in \mathbb{R}^{q\times 1}$ is a vector whose elements represent the state of the MP in task space coordinates. Note that task state $\boldsymbol{s}_i$ and feature state $\boldsymbol{x}_i$ are different. The task state represents the state controlled by the system and is used for learning the MP goals. The feature state instead is decoupled from the controller and can represent anything, such as the state of the environment. It will be used for learning the transition behavior in Section 4. Within our system, each dimension of the goal is learned independently. We have observed that one-dimensional goal learning is reflecting the behavior of a human teacher more naturally. As an example, consider a teacher moving a gravity compensated robot end-effector to a desired 3D-position. The teacher will start approaching the goal position. Due to an imperfect motion, not all dimensions of the goal position will be reached at the same time. Instead, the teacher may recognize that the desired x-position is already reached, but y- and z-position are not. Therefore, while trying to reach the desired overall position, the end-effector is moved along the yz-plane while the x-position is kept constant. The resulting trajectory will differ from the desired linear attractor behavior of the robot. For a real demonstration, the teacher also has to control the orientation of the end-effector and the hand of the robot. As it is difficult to reach the desired goal state all at once,



(a) Trajectory (b) Normalization (c) Approximation and Prediction (d) Goal Learning

Figure 3: Goal learning overview. The trajectories (a) are velocity normalized (b) and approximated by linear functions (solid red lines, c). For each trajectory, the goal is expected to lie on its future path (red dashed lines), which is predicted using the linear functions. The goal (thick black line) is then found by minimizing the cost function (d), which arbitrates between all expected goals. Note that the cost function is zero for the entire gray area. Therefore the center of the interval is chosen as goal of the MP. The thin black line shows the mean of the trajectory end points as comparison.

humans seem to concentrate on a few dimensions first. Therefore, demonstrations usually do not really match the attractor behavior of a real robot motion. To compensate for this mismatch, we learn the goal per dimension. In the following, we therefore use a scalar notation for the task space.

We focus on tasks where the velocity of a movement is irrelevant. Therefore, the time does not correspond to the real time axis of the demonstrations, but is computed by normalizing the velocity

$$t_i = t_{i-1} + (s_i - s_{i-1})^2, \quad t_0 = 0, \tag{1}$$

as illustrated in the two left plots in Figure 3. As model for the approximation of a trajectory, a simple linear function is used

$$f(t_i) = at_i + b = \tilde{s}_i. \tag{2}$$

Here, $\tilde{s}_i$ is the predicted state of the MP at time t_i and a and b are the parameters of the model. Although a linear function is a simple model, it is notable here that it matches the demonstrations of single attractor movements quite well, as they can be seen as point-to-point movements in task space. We focus on tasks that have such linear characteristics, e.g., pick-and-place tasks. The assumption of a linear model usually does not apply for the transition between two MPs. During demonstration and reproduction of a sequence, a transition can occur with a non-zero velocity. As a consequence, the start of a MP may be influenced by its predecessor. The resulting trajectory will contain arcs or edges and is an example for a trajectory that cannot be well represented using a straight line. Nevertheless, note that there is no need to approximate the whole trajectory well in our approach. Instead, the line only has to pass through the real goal at some future time point. The method has to find the parameters of Equation (2) to ensure this property.

Therefore, we chose to use weighted least squares regression (WLSR, [5]) for learning the parameters a and b. Compared to least squares regression, WLSR additionally allows to weight the importance of each data point. By weighting the data points at the end of a trajectory stronger than at the beginning, it is possible to minimize the influence of the preceding movement, while still matching the overall trajectory well. The parameters are found by minimizing the weighted sum of distances between the sampled states s_i and predicted states $\tilde{s}_i$ from Equation (2), resulting in the cost function

$$J(a, b) = \sum_{i=1}^{N} g(i)\,(s_i - \tilde{s}_i)^2\,. \tag{3}$$

Here, N is the number of samples and $g(i)$ is a weighting function. We suggest to use $g(i) = i^2/N$, as the quadratic weights minimize the influence of the preceding MP at the start of a trajectory and focus on the data points closer to the goal. Cubic or even larger weights focus on few data points at the end of a trajectory and therefore become sensitive to noise. Minimizing the error function (3) is straightforward (see [5]).

2.2. Goal Learning

We assume that a MP has been active M times, resulting in M trajectories approximated by linear functions f_j with $j = \{1, \ldots, M\}$. Figure 3(c-d) shows an overview of our goal learning approach. As already mentioned, the basic idea is that for each trajectory j, the goal is expected to lie on its future path. The future path is predicted using the linear function f_j, which allows us to formulate the expected goal in terms of the slope parameter a_j and the predicted state u_j at the final time of the trajectory

$$u_j = f_j(t_N^{(j)}). \tag{4}$$

If the slope is positive, the expected goal is equal or greater than u_j. If it is negative, the expected goal is equal or less than u_j. For finding the best compromise between all

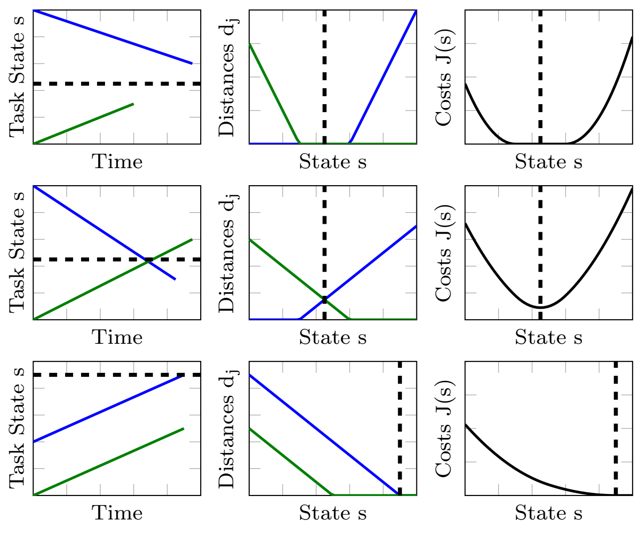


Figure 4: Three goal learning cases and their resulting distance and cost functions. In general, trajectories may converge (top), diverge (center) or point in the same direction (bottom). The black dashed lines show the goal of the MP.

trajectories, we construct a cost function which penalizes deviations from each expected goal. As a first step, we define the distance between a state s and the expected goal of a trajectory j as

$$d_j(s) = \begin{cases} 0, & \text{if } a_j > 0 \text{ and } s \geq u_j, \\ 0, & \text{if } a_j < 0 \text{ and } s \leq u_j, \\ \left|s - u_j\right|, & \text{otherwise.} \end{cases} \tag{5}$$

Then, the attractor goal of one particular dimension of a MP can be defined as the point s_g where the squared sum of distances becomes minimal

$$J(s) = \sum_{j=1}^{M} d_j^2(s), \tag{6}$$

$$s_\mathrm{g} = \min_s J(s). \tag{7}$$

Figure 4 shows some trajectories and their resulting distance and cost functions as an example. For finding the solution s_g, we first calculate the derivative of the cost function (6). Due to Equation (5), the cost function is non-differentiable at each threshold u_j. Therefore, the derivative has to be computed for each interval $[u_j, u_{j+1}]$ separately and is given by

$$\frac{d}{ds} J(s) = 2 \sum_{j \in D} (s - u_j). \tag{8}$$

Here, D is the set of functions for which Equation (5) is non-zero. For the intervals, we assumed that the thresholds have been sorted in ascending or descending order. Setting the derivative equal to zero and rearranging for s results in

$$s_\mathrm{g} = \frac{1}{n_D} \sum_{j \in D} u_j, \tag{9}$$

where n_D is the number of elements in D. The solution s_g may lie outside of the interval. In that case, it is clipped to the closest interval border. If there exists an interval for which D is empty, the error will become zero and hence any value in this interval is a possible solution. In that case, the center of the interval is taken as goal. The final equation therefore is

$$s_\mathrm{g} = \begin{cases} (u_j + u_{j+1})/2 & \text{if } n_D = 0, \\ u_j & \text{if } n_D \neq 0,\ s_\mathrm{g} < u_j, \\ u_{j+1} & \text{if } n_D \neq 0,\ s_\mathrm{g} > u_{j+1}, \\ s_\mathrm{g} & \text{otherwise.} \end{cases} \tag{10}$$

The goal s_g is computed for every interval $[u_j, u_{j+1}]$ and subsequently inserted into Equation (6). The goal resulting in the lowest value of this cost function is then chosen as final goal of the MP.

Due to the quadratic dependency of Equation (6) on the expected goals, overshoots may shift the goal away from the desired value. However, our experience is that if a teacher recognizes that the goal was not hit accurately, he/she usually corrects his/her mistake, so that in the end the real goal is approximately reached. If a MP is stopped prematurely, overshoots also do not lead to problems. Additionally, the trajectory approximation with WLSR leads to some robustness against overshoots.

3. Learning Graph Representations

In the previous section, the parameters of the individual MPs have been acquired. In this section, we propose an approach for learning a graph representation of the demonstrated sequences which we call sequence graph. In a sequence graph, every node is linked to a MP. During reproduction, the graph determines which MP may be activated next. A missing transition in the graph might prevent an activation of the correct MP, while too many outgoing transitions might lead to the activation of a wrong MP due to perceptual aliasing. It is therefore crucial to find a good structure for a given set of demonstrations.

Figure 5 shows an overview of the graph learning based on a simple toy example with only three different MPs, that will be used throughout this section. The MPs are indicated by different colors. They are chosen arbitrarily and have no further meaning, but show the essential characteristics of our approach. First, we perform at least one kinesthetic demonstration. In general, we assume that M demonstrations have been collected. For each demonstration, we get a labeled (background colors in Figure 5a) set of features (black lines). The features are only used for learning the transition behavior and will be explained in Section 4. For learning a sequence graph, only the observed MP sequence is used, as shown in Figure 5b. The graph representation will be explained in detail in the following section, where we also present two different types of sequence graphs, both showing different ways of incorporating the sequences into the representation.

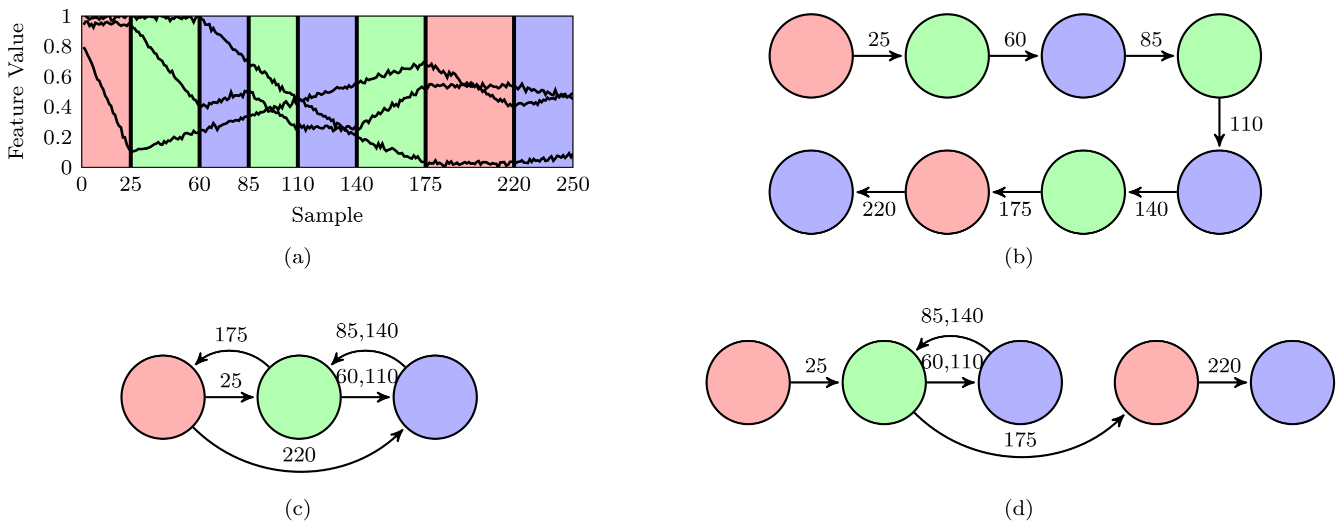


Figure 5: Overview of the graph learning. First, the labeled data from a set of demonstrations (a) is taken to extract the MP sequence (b). This sequence can then be used to generate a sequence graph. We investigate two different sequence graph types. A compact local sequence graph (c) and a more sophisticated global sequence graph (d). The numbers on the transitions correspond to the transitions points (TPs, see upper left figure). A TP is a point in time at which a transition between MPs occurs.

A sequence graph is a directed graph in which each node n_i is linked to a MP. This mapping is not injective which means a MP can be linked to more than one node. During reproduction, a MP is activated if a linked node is considered active. Transitions in the graph lead to succeeding MPs that can be activated if the current MP has finished. A transition $t_{k,l}$ is connecting the node n_k with n_l. Each transition is linked with the corresponding transition points (TP) at which it was observed during the demonstration (black vertical lines in Figure 5a). As the same transition can be observed multiple times, multiple TPs are possible.

Having m nodes in a graph, we use a $m \times m$ transition matrix $\boldsymbol{T}$ with elements $t_{k,l}$ to describe one sequence graph. As a MP is usually activated for more than one time step, the transition $t_{k,k}$ exists for all k. We start with one directed acyclic graph with nodes $n_{j,i}$ for each trial (see Figure 5b), which contains the observed MP sequence. The main step is now to combine multiple of these graphs into one representation of the skill, which can be a hard problem as the algorithm has to work solely based on the observations. As an example, consider the task of baking a cake. Here, it does not matter if milk or eggs are put in the bowl first. Still, the task may be demonstrated one time with the sequence milk-eggs and one time with the sequence eggs-milk. From an algorithmic point of view it is often not clear if a sequence is arbitrary for a skill or if the differences can be linked to some traceable sensor events. Hence, there are different ways of building the graph structure for a skill. We show two possibilities by investigating two different kinds of sequence graphs. The local graph presumes a sequence to be arbitrary and is not considering it in the representation, while the global graph is trying to construct a more detailed skill description.

3.1. Local Sequence Graph

The local sequence graph assigns exactly one node to each activated MP and hence the number of nodes and MPs is equal. The graph is initialized with one node per MP and without transitions. For each observed pair of MPs a transition is added to the graph. As only pairs and no history are considered, it is irrelevant at which point in the sequence a transition occurs. The corresponding graph for the toy example is shown in Figure 5c.

The graph contains only three nodes, one for each activated MP. When reproducing the movement, a transition from the red MP to the blue one is always possible at this level of the hierarchy and it is up to the classifier to prevent such incorrect transitions. The major drawback of this representation is the strong requirement on the feature set, as it has to be sufficiently meaningful to allow for a correct classification independent of the history of activated MPs.

3.2. Global Sequence Graph

The global sequence graph attempts to overcome this issue by constructing a more detailed skill description. One essential characteristic of the global sequence graph is that there is no one to one mapping between MPs and graph states. Instead, a MP can appear multiple times in one representation as depicted in the global sequence graph of the toy example (Figure 5d). Here, two nodes are linked to the red MP because the sequence was considered to be in two different states when they were activated. The repeated appearance of the green-blue transitions (see Figure 5a) is represented by only two nodes as in the local graph. The reason is that consecutive sequences of the same MPs are considered to be a repetition which can be demonstrated and reproduced an arbitrary number of times. Repetitions are also advantageous when describing tasks with

Algorithm 1 Graph Folding

```
Require: $\boldsymbol{T}$
 1: $repetition$ = findRepetition($\boldsymbol{T}$);
 2: while $repetition.found$ do
 3:   $\boldsymbol{R} = \emptyset$;
 4:   $repetition.l = repetition.end - repetition.start + 1$;
 5:   for $i = repetition.start$ to $repetition.end$ do
 6:     mergeNodes($\boldsymbol{T}(i + repetition.l), \boldsymbol{T}(i)$);
 7:     $\boldsymbol{R} = \boldsymbol{R} \cup \boldsymbol{T}(i)$;
 8:   $repetition$ = findRepetition($\boldsymbol{T}$);
 9:   if $!repetition.found$ then
10:     $tail$ = findTail($\boldsymbol{T}, repetition.end + 1$);
11:     if $tail.found$ then // Found incomplete cycle
12:       for $i = tail.start$ to $tail.end$ do
13:         mergeNodes($\boldsymbol{T}(i + r), \boldsymbol{T}(i)$);
14:         $\boldsymbol{R} = \boldsymbol{R} \cup \boldsymbol{T}(i)$;
15:   $\boldsymbol{T} = \boldsymbol{T} \setminus \boldsymbol{R}$; // Remove merged nodes from graph
```

Algorithm 2 Graph Merging

```
Require: $\boldsymbol{T}_{\mathrm{A}}$, $\boldsymbol{T}_{\mathrm{B}}$
 1: $\boldsymbol{U}_{\mathrm{A}}$ = getUniquePaths($\boldsymbol{T}_{\mathrm{A}}$);
 2: $\boldsymbol{U}_{\mathrm{B}}$ = getUniquePaths($\boldsymbol{T}_{\mathrm{B}}$);
 3: for all $\boldsymbol{u}_{\mathrm{B}} \in \boldsymbol{U}_{\mathrm{B}}$ do // Iterate over paths
 4:   $c_{\max} = 0$;
 5:   for all $\boldsymbol{u}_{\mathrm{A}} \in \boldsymbol{U}_{\mathrm{A}}$ do
 6:     $c$ = compare($\boldsymbol{u}_{\mathrm{A}}, \boldsymbol{u}_{\mathrm{B}}$); // Nr. matching nodes
 7:     if $c > c_{\max}$ then
 8:       $c_{\max} = c$;
 9:       $\boldsymbol{n}_{\mathrm{B,max}} = \boldsymbol{u}_{\mathrm{B}}(1, \ldots, c)$; // First $c$ nodes
10:       $\boldsymbol{n}_{\mathrm{A,max}} = \boldsymbol{u}_{\mathrm{A}}(1, \ldots, c)$;
11: for all $n_{\mathrm{A}} \in \boldsymbol{T}_{\mathrm{A}}, n_{\mathrm{B}} \in \boldsymbol{T}_{\mathrm{B}}$ do // Iterate over all nodes
12:   if $n_{\mathrm{B}} \in \boldsymbol{n}_{\mathrm{B,max}}$ then // Nodes match
13:     mergeNodes($n_{\mathrm{A}}, n_{\mathrm{B}}$);
14:   else // Node of graph B has to be added to A
15:     addNode($\boldsymbol{T}_{\mathrm{A}}, n_{\mathrm{B}}$);
```

repetitive characteristics, such as unscrewing a light bulb. Here, the unscrewing movement has to be repeated several times depending on how firm the bulb is in the holder. As the number of repetitions is not fixed for each single demonstration, the algorithm has to conclude that different numbers of repetitions of the same behavior appeared in the demonstrations and incorporate this information into the final representation of the task.

Note that even if a skill requires a fixed number of repetitions, both presented sequence graphs will contain a cycle in the representation. The system is then only able to reproduce the movement properly if the classifier would find the transition leading out of the cycle after the correct number of repetitions. While an improvement is not possible here for the local graph, a fixed number of repetitions can be modeled with the global graph by skipping the search for cyclic transitions.

3.3. Graph Construction

The local sequence graph is created by adding one node for each observed MP and one transition for every observed MP pair. If a MP pair is observed multiple times, only one transition is added to the graph. For creating a global sequence graph, three steps have to be performed.

1. Create one acyclic graph $\boldsymbol{T}_j$ for each demonstration.
2. Replace repetitions of MPs with cyclic transitions.
3. Combine updated graphs to one global representation $\boldsymbol{T}$ of the skill.

The first step is straightforward as the acyclic graph represents the MP sequence directly observed in the demonstrations. We call the second point *folding* and its pseudo code is shown in Algorithm 1. The algorithm starts by calling the method findRepetition, which is searching for repetitions of length $l = \lfloor m/2 \rfloor$ in a graph $\boldsymbol{T}$ with m nodes. The method starts by comparing the MPs of the nodes $\{n_0, n_1, \ldots, n_l\}$ with $\{n_{l+1}, \ldots, n_{2l+1}\}$. If both node chains match, the node pairs $\{n_0, n_{l+1}\} \ldots \{n_l, n_{2l+1}\}$ are returned. If the chains do not match, the indices are incremented by one and the method starts from the beginning with n_1 as starting point. The shifting is done until the end of the list is reached. Next, l is decremented by one and all previous steps are repeated. Thus, longer repetitions are preferred over shorter ones. The method terminates if the cycle size is one, which means no more cycles can be found.

If a repetition is found, the corresponding nodes are merged to a single node. When merging two nodes n_A and n_B, the input and output transitions of node n_B become input and output transitions of n_A. If an equal transition already exists for n_A, only the associated TPs are added to the existing transition. Note that a cyclic transition is introduced when merging the nodes n_0 and n_{l+1}, as this leads to the input transition $t_{l,l+1}$ being rerouted to $t_{l,0}$. After each iteration of the algorithm, the nodes of the latter chain are not connected to the rest of the graph anymore and can be removed from the representation. To allow escaping a cycle not only at the end of a repetition, the algorithm also searches for an incomplete cycle after a found repetition. This tail is considered to be part of the cycle and is also merged into the cyclic structure (Algorithm 1, lines 11-15). The toy example also contains an incomplete cycle, as the green-blue repetitions end incompletely with the green MP.

We call the final step of creating a global sequence graph *merging*, as several separate graphs are merged into one representation. Algorithm 2 merges two graphs and thus gets called $M - 1$ times for M demonstrations. The algorithm steps through the graphs simultaneously, starting at the initial nodes, merging equal nodes and introducing branches whenever nodes differ. The algorithm starts by extracting the unique paths from both graphs. A unique path is a path which starts with a node that has no input transition and ends with a node that has either no output

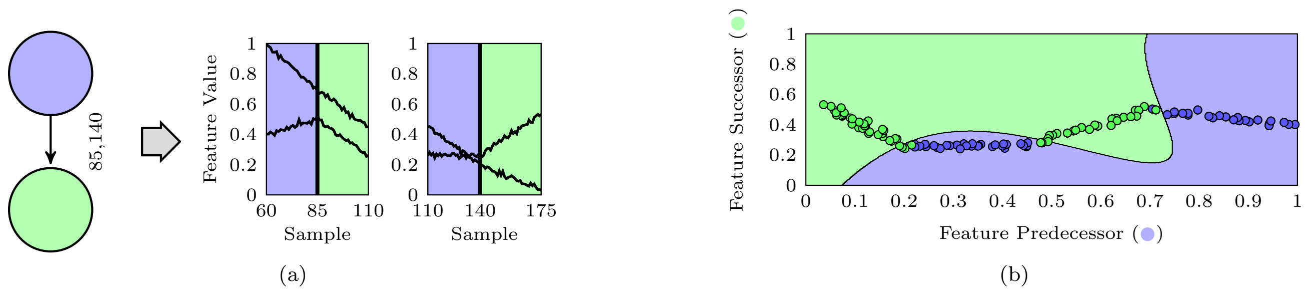


Figure 6: One classifier is created for each node in the graph. The training is performed by using the data around the transitions between the connected MPs (a). The data is projected into feature space and the classifier learns a separating border between the MPs, as indicated by the background color in (b). As soon as the current feature state crosses such a border during reproduction, a transition in the graph is triggered, leading to a switch to a different classifier.

transition or only output transitions to nodes that were already visited. The toy example has two unique paths, red, green, blue and red, green, red, blue (see Figure 5d). Next, the algorithm compares the paths of both graphs with each other from left to right, searching for the longest equal subpath. Two nodes are considered as equal if the columns of the corresponding transition matrices are equal, which means both nodes use the same underlying MP and have the same input transitions. Finally, the nodes of the longest equal subpath are merged, whereas all other nodes of graph $\boldsymbol{T}_{\mathrm{B}}$ are added to $\boldsymbol{T}_{\mathrm{A}}$. By searching for the longest subpath, branches are introduced at the latest possible point in the combined graph. Once branched, both branches are separated and do not get merged together at a later point in the sequence.

4. Learning the Transition Behavior

After creating the graph representation, the next step is to train the classifiers — one for each node in the graph. If a node is active during reproduction, its associated classifier decides when to transition to a possible successor node. This multiclass classification problem has the active node and all of its neighbor nodes in the graph as classes. Due to the graph representation, we do not have to learn an overall classification function $f(\boldsymbol{x}) = p$ with $p \in P$ and $\boldsymbol{x}$ being the combined feature vector of all MPs $\boldsymbol{x} = (\boldsymbol{x}^{(1)}; \boldsymbol{x}^{(2)}; \ldots; \boldsymbol{x}^{(K)})^{\mathrm{T}}$, but can restrict the classes c of each classifier to a subset $P_c \subseteq P$ and the data vector to the feature vectors of the elements in P_c. Restricting the number of classes often increases the accuracy of the system as transitions not observed in the training data are prevented. A reduction of the feature vector can be seen as an implicit dimensionality reduction where unimportant features used by uninvolved MPs are no longer considered for the decision.

Figure 6a depicts the data used for training a classifier as an example. After the demonstrations, each transition in the acyclic graph is linked to one TP in the sampled data. During the merging and folding process of the global sequence graph or the pair search for the local graph transitions are merged together, resulting in potentially multiple TPs for each transition. For each TP, the data points between the previous and next TP in the overall data are taken from the training and labeled with the MP that was active during that time. As all transitions have the same predecessor for one classifier, the first part of the data will always have the same labels, while the second part may differ depending on the successor node of the transition.

The classifiers learn from the labeled training data how to separate the MPs in feature space (Figure 6b). During the reproduction of the MP, the current feature state of the robot is tracked and as soon as it crosses a border, the corresponding successor will be activated, leading to a transition in the graph and a switch to another classifier. Any classifier is applicable to our method. We evaluated support vector machines (SVMs, [8]), logistic regression (LR, [2]), kernel logistic regression (KLR, [8]), import vector machines (IVMs, a certain type of sparse kernel logistic regression, [30]), and Gaussian mixture models (GMMs, [2]).

5. Evaluations and Experiments

For evaluating our approach, we perform three different experiments. In Section 5.1, we evaluate the goal learning on a task where a robot has to move an object in its workspace. In Section 5.2, we evaluate the overall performance of the system, including the two sequence graph representations and different classifiers. In the experiment, a robot has to unscrew a light bulb. In Section 5.3, the system has to learn to grasp different objects. Additionally, an error recovery strategy for unsuccessful grasps is demonstrated. With the third experiment, we evaluate the system performance on a more complex feature set. All experiments are evaluated using a real seven degrees of freedom (DOF) Barrett WAM robot with an attached four DOF hand. For the first and third experiment, also some simulation results are presented.

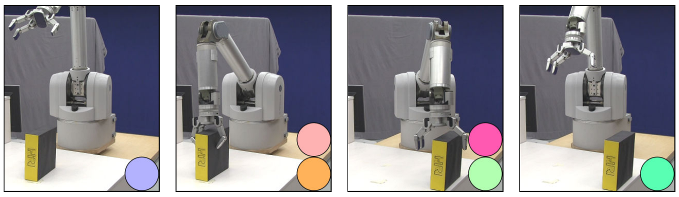


Figure 7: In the first experiment, the robot has to move an object to a certain position. The robot starts in an initial position (●), moves to the object (●) and grasps it (●). Subsequently, it moves the object to another position (●), opens its hand (●) and moves to the final position (●).

5.1. Moving an Object

The first experiment evaluates the goal learning algorithm we describe in Section 2. Therefore, we chose a rather simple sequence of movements, which does not differ between single demonstrations. The robot starts in an initial position, moves to the object and grasps it. Subsequently, it moves the object to another position, opens the hand and returns to the initial position, as illustrated in Figure 7. As the sequence always is the same and does not include any repetitions or specific patterns of movements, the local and global sequence graph algorithms return the same structure. A MP may control the position and/or orientation of the end effector as well as the joint angles of the fingers. If an entity is not controlled by a MP, the movement results from the null space criteria (e.g., joint limit avoidance) of the underlying task space controller. Six different MPs have been defined to perform the task, as shown in Table 1.

Before the experiments on the real robot are presented, the goal learning is evaluated in simulation first. As kinesthetic teaching is not possible in simulation, we predefine the demonstrated sequence with a state machine and the transition behavior between MPs using thresholds. For realism and variation, Gaussian noise is added to the thresholds. Every time a new MP is activated, new thresholds are computed. The goal of each MP k is set to desired values $\tilde{\boldsymbol{s}}_{\mathrm{g}}^{(k)}$ and perturbed with additive noise $\mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I})$ for each demonstration. The intention of this experiment is to evaluate the robustness and accuracy of the goal learning by disturbing the transition behavior and MP goals. We perform eight demonstrations with a fixed σ and learn the goals $\boldsymbol{s}_{\mathrm{g}}^{(k)}$ of every MP k after each demonstration with the data of all demonstrations that have been performed up to this point. For each learning instance, an error is computed

MP	Position	Orientation	Fingers	Next MP
●	Initial	-	-	●
●	A	Fixed	Open	●
●	Hold	Fixed	Closed	●
●	B	Fixed	Closed	●
●	Hold	Fixed	Open	●
●	Final	-	-	-

Table 1: MPs for the object movement experiments.

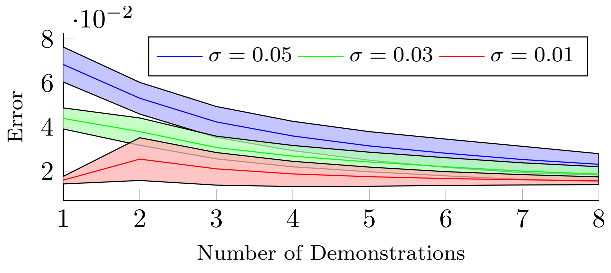


Figure 8: Goal learning error for the simulated toy example. The error degrades when demonstrating a task multiple times. The background color shows the hull of one standard deviation.

according to

$$e = \frac{1}{N} \sum_{k=1}^{N} \left\| \tilde{\boldsymbol{s}}_{\mathrm{g}}^{(k)} - \boldsymbol{s}_{\mathrm{g}}^{(k)} \right\|_1 , \tag{11}$$

which is the mean of the ℓ_1-norm of the difference between predefined and computed goal of all N MPs. The experiment is repeated with three different values for σ 20 times, so that in total 480 demonstrations are performed. We summarize the results for all learning instances according to the number of demonstrations they have been trained with and compute the mean and standard deviations of the errors. The results are plotted in Figure 8. In general, the error decreases slightly for more demonstrations, but is always consistent with the amount of noise added to the system.

For the experiments with the real robot, we perform three kinesthetic demonstrations. Transitions between MPs are indicated by pressing a key every time we consider a movement as complete. Opening and closing of the hand is also activated by pressing a key. The labeling is performed based on the indicated transitions. Next, the transition behavior is learned with SVMs as classifier and the task is reproduced on the real robot.

One feature $x_1^{(k)}$ is assigned to each MP k according to the equation

$$\boldsymbol{\Delta} = \boldsymbol{s}_{\mathrm{g}}^{(k)} - \boldsymbol{s}^{(k)}, \tag{12}$$

$$x_1^{(k)} = 1 - \exp(-0.5(\boldsymbol{\Delta}^{\mathrm{T}} \boldsymbol{\Sigma}_k^{-1} \boldsymbol{\Delta})). \tag{13}$$

Here, $\boldsymbol{s}^{(k)} \in \mathbb{R}^{q \times 1}$ is the current state of the robot in task space coordinates and $\boldsymbol{\Sigma}_k$ is a $q \times q$ diagonal matrix with positive parameters. Note that q can be different for each MP. Equation (13) depends on the absolute difference between the state of the robot and the MP's goal position. Hence, the feature can be seen as progress indicator and is called goal distance [14]. It is intrinsically in the range $[0, 1]$ and makes further data scaling superfluous. In addition, the variation of the feature around the MP goal can be shaped with the parameters of $\boldsymbol{\Sigma}_k$. To get expressive features, we learn the parameters by minimizing the variance of the feature values while constraining the min and max values to be as close to zero and one as possible.

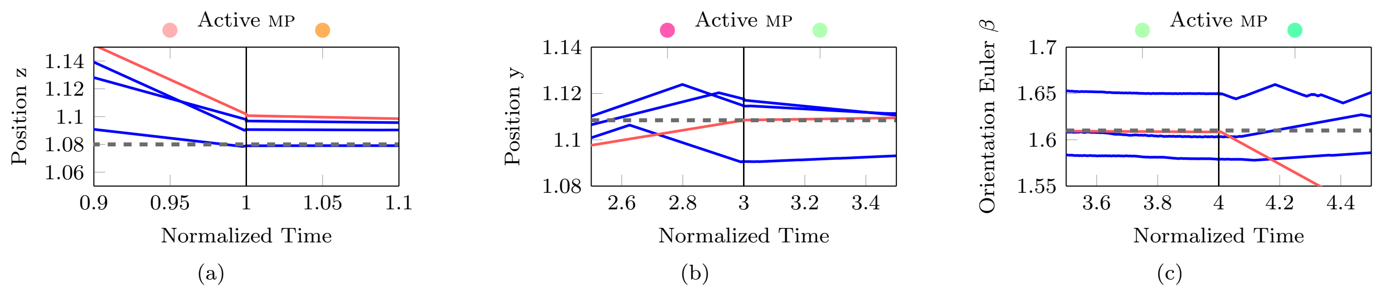


Figure 9: Comparison of trajectories from kinesthetic demonstrations (blue) and reproduction (red). The dashed lines show the learned MP attractor goals. All trajectories are aligned in time, so that each MP activation takes the same (normalized) time. The learned goals and the transition behavior are consistent with the demonstrations.

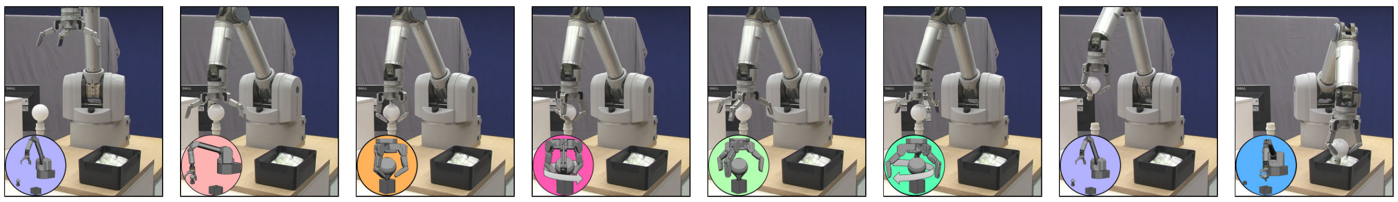


Figure 10: Illustration of a successful unscrewing sequence. The robot starts in an initial position (🟣) and first moves towards the bulb (🩷). Then it repeats the unscrewing movement (🟠, 🔴, 🟢, 🟩) until the bulb loosens (🟣) and subsequently, the bulb is put into a bin (🔵) and the robot returns to its initial position (🟣).

Figure 9 shows the resulting trajectories for some selected MP transitions. The learned goals are consistent with the directions of the movements. If trajectories are constant or diverge slightly, the goal is averaging over the trajectories (Figures 9b,c). If all trajectories point in the same direction as it is the case for Figure 9a, the goal lies in this direction as well, without conflicting with the trajectories.

The first example also illustrates the difference between the goal state of a MP and the state at which a new MP is activated. A new MP is activated as soon as the classification border is crossed, which is the case when the feature state reaches the value of the first transition. Note that such an early transitioning strategy is only triggered if it was also demonstrated. Therefore, behaviors such as opening or closing a gripper prematurely should not occur unintendedly.

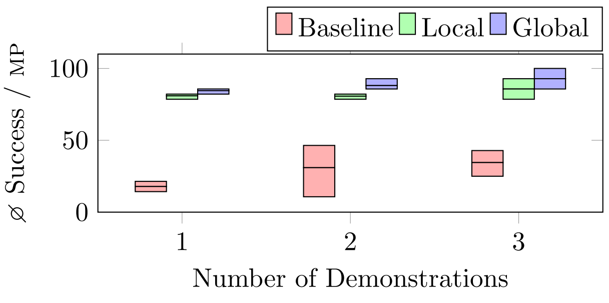
5.2. Unscrewing a Light Bulb

In the second experiment, the system has to learn how to unscrew a light bulb. The focus of this experiment is on evaluating the overall performance of the learning system, including the goal learning, the graph representations and the transition behavior. For the representation of the skill, we choose seven different MPs, shown in Table 2.

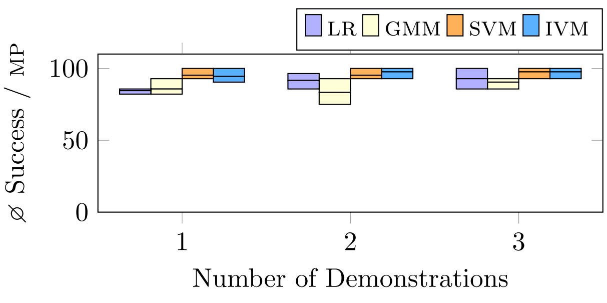
The detailed task flow is illustrated in Figure 10. We choose to unscrew the light bulb by caging it. Here, the robot encloses the bulb with its hand and grasps it below the point with the largest diameter. For positioning the robot, the end effector coordinates defined relative to the light bulb holder are set. When opening, closing or rotating the hand, either the three DOFs of the fingers or the angle of the wrist joint are controlled by the MP. The unscrewing MP (rotating the closed hand counterclockwise) additionally applies a force in upward direction to the robot's hand to ensure contact with the bulb. Again, the goal distance feature is assigned to each MP. The goal distance of the 🩷-MP can be used to detect if the light bulb is still in the holder. As a force is applied in upward direction during unscrewing, this force leads to an acceleration of the robot's arm as soon as the light bulb gets loose. As a consequence, the arm moves away from the MP's goal, resulting in an increasing value of the goal distance. The system has to learn that an increase of this goal distance leads to an immediate stopping of the unscrewing MP and a transition to the branch in the graph that puts the light bulb into the bin. As the light bulb is not represented in the feature set and the unscrewing stopping criterion depends implicitly on the height of the end-effector, a slipped light bulb can not be detected. As no slip happened during our experiments, we did not integrate the state of the light bulb into the feature set.

MP	X	Y	Z	Orientation	Fingers	Next MP
🟣		Initial Position		Initial	Hold	🩷, 🔵
🩷		Light Bulb		Hold	Spread	🟠
🟠		Light Bulb		Hold	Closed	🔴
🔴	Bulb	Bulb	Force	Rot. Wrist	Closed	🟢, 🩷
🟢		Light Bulb		Hold	Spread	🟩, 🟣
🟩		Light Bulb		Rot. Back	Spread	🔴
🔵		Garbage		Hold	Closed	🟢

Table 2: MPs for the light bulb experiments.

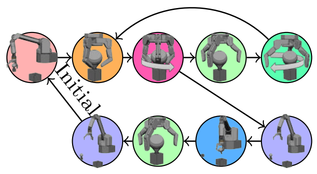


(a) Graph Comparison

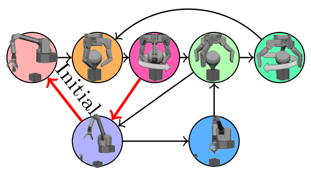


(b) Classifier Comparison

Figure 11: Experimental results for the light bulb task. The left plot shows the comparison of the three different graph structures: A fully connected graph used as baseline and the local and global sequence graph. All graphs were trained with Logistic Regression (LR) as classifiers. The right plot shows the evaluation of different classifiers: LR, Gaussian Mixture Models (GMMs), Support Vector Machines (SVMs), and Import Vector Machines (IVMs). Here, the global sequence graph has been used. The bars in both plots indicate the minimum, average and maximum success rate of each MP transition during reproduction of the task.



(a) Global Sequence Graph



(b) Local Sequence Graph

Figure 12: Graph representations of the light bulb task. Compared to the global graph, the local graph merges several nodes. The merging creates paths in the graph which were not demonstrated. An example is the sequence marked as red which leads to a misbehavior of the robot if reproduced.

We perform three kinesthetic demonstrations and vary the position of the light bulb holder for each demonstration. For all following experiments, the system is trained separately with the data of each single demonstration, all pairs of demonstrations and all demonstrations. Every time the system is trained, the task is reproduced and the success rate of each MP transition is evaluated. A transition is considered successful, if the system activates at the correct state the correct successor. Incorrect, premature or too late transitions are considered failures. For unsuccessful transitions, we restart the movement, trigger the transition manually and continue with the reproduction from there.

We first evaluate the graph representations by comparing the local and global sequence graph with a baseline graph, which has one node for each MP and is fully connected. Hence, the system is allowed to transition to any MP at every point in time. All graph types are trained with LR as classifiers. The reproduction results are shown in Figure 11a. Both presented graph representations are clearly better suited than the baseline graph. Due to the reduced number of outgoing transitions for each node, the effect of perceptual aliasing gets reduced, which in turn improves the performance of the classifiers. This effect is also the reason why the global sequence graph slightly outperforms the local sequence graph. Both graphs are shown in Figure 12. The local sequence graph contains paths which have not been demonstrated and lead to misbehavior if reproduced. An example is the red path in the figure. Here, the robot returns to its initial position with the bulb in its hand and immediately goes back to the bulb holder while opening its hand instead of going to the bin.

In a second set of experiments, we evaluate different classifiers. In addition to LR, we evaluate GMMs, SVMs, and IVMs (see Section 4). All classifiers are trained with the global sequence graph, as this was the overall winner of the first experiments. The reproduction results for the different classifiers are shown in Figure 11b. The results indicate only a slightly better performance of the kernel methods compared to LR and GMMs. When being trained with all demonstrations, the average success rate of SVMs and IVMs is 97.6%, while LR reaches only 92.9%. The main reason for failing is the unscrewing movement, where the system sometimes fails to generalize from the demonstrations. When the light bulb gets loose at the beginning of the unscrewing movement during demonstration, the system is expected to be in a similar state when the light bulb gets loose during reproduction. If both states are different, the system sometimes fails to trigger the transition to the successor MP properly. This effect is reduced if more demonstrations are performed, as different wrist orientations are observed for each transition and therefore this feature becomes irrelevant for the decision. In general, a feature selection method may be helpful for further improving the performance.

5.3. Grasping Objects with Error Recovery

In a third experiment, the system has to learn to grasp and lift three cylinders with different lengths using different grasps. The intention is to evaluate a more complex feature set, which also represents the state of the environment, positions and joint angles instead of the MP goal distance features. Table 3 summarizes all features for this

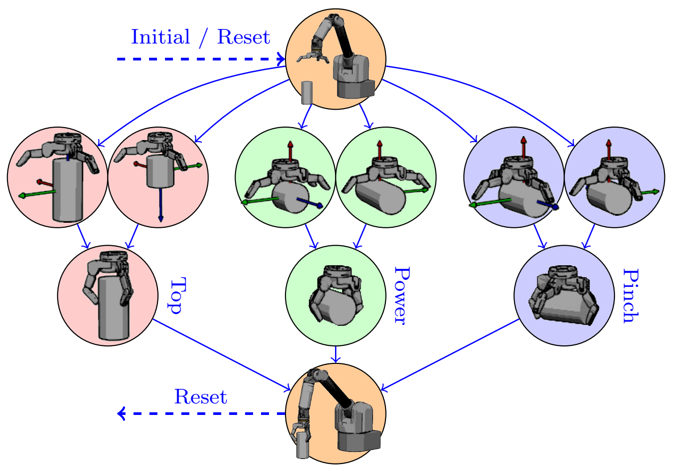


Figure 13: Task flow of the grasping task. The robot starts in an initial position (top). Depending on the size of the cylinder and its orientation, the system approaches the cylinder with different orientations and finger configurations. Subsequently, the cylinder is grasped and lifted. If the contact with the cylinder is lost during lifting, the system has to re-start the task.

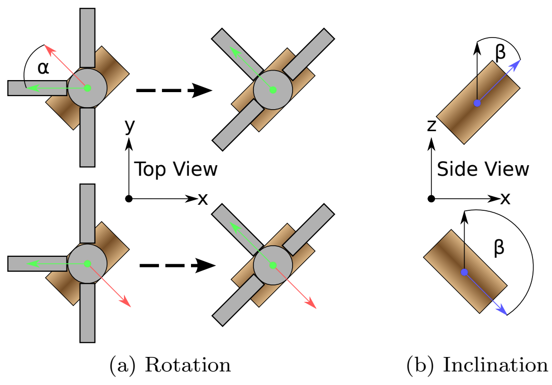


Figure 14: The rotation feature (a) is the angle between the shown axes of the lying cylinder (red) and the hand (green). For avoiding unnecessary rotations of the end-effector, we define two pre-grasp MPs that approach the cylinder with different orientations (top and bottom). Depending on the angle α, the MP that is closer to its target orientation should be activated. The feature is shown for the pinch grasp, but is used for the power grasp in the same manner. The inclination feature (b) is the angle between the negative gravity vector (black) and the shown cylinder axis (blue). It indicates if the cylinder is standing ($\beta = 0$), lying or standing upside down ($\beta = \pi$).

task. The position of the end-effector is relative to the cylinder. Inclination and rotation are angles that represent the orientation of the cylinder. Both features are illustrated in Figure 14. Orientation and position of the cylinder are measured using a six DOF magnetic field tracking sensor system with precisions of approximately 1 cm and 0.15°. The strain gauges measure the tension in each finger. Together with the finger joint angles, they are used for detecting successful grasps or a slipped cylinder.

The task is demonstrated as follows (see Figure 13). The end-effector starts in an initial position and is first moved to a pre-grasp position close to the cylinder. Subsequently, the cylinder is grasped and lifted. Depending on the length of the cylinder and its orientation, different grasps and pre-grasps are used to solve the task. If the cylinder is standing, it is grasped from the top. If it standing upside down, it is grasped at its bottom. We demonstrate the task with three different cylinder lengths, 8, 16, and 24 cm. If the cylinder is lying and has a length of 8 or 24 cm, it is grasped using the power grasp. If the 16 cm cylinder is lying, it is grasped using the pinch grasp. Pinch and power grasp can be performed with two different wrist angles as shown in Figure 14. During the final lifting step, the

Feature	Dimension
Cylinder Length	1
Cylinder Inclination	1
Cylinder Rotation	1
End-Effector Position	3
Strain Gauges	3
Finger Joint Angles	4

Table 3: Feature set for grasping task.

cylinder may slip. In that case, the lifting is immediately stopped. The end-effector is moved to the initial position and the task is demonstrated from the beginning. Note that even though this error recovery strategy is demonstrated to the robot, the system has no explicit notation of an error when reproducing the task. Instead, a slipped cylinder is supposed to lead to a MP transition which is treated just as any other MP transition. The immediate triggering of the error recovery is also a typical example for a MP that is stopped before reaching its goal state when reproducing the task.

Similar to the previous experiments, we train the system incrementally after each demonstration. Each training is followed by an evaluation of the reproduction. A successful reproduction of the tasks requires grasping and lifting the cylinder, as well as successfully detecting a slipped cylinder. Figure 15 shows the success rate of the reproduction for all 17 demonstrations. We evaluated the system with the local sequence graph and LR in simulation and SVMS on the real robot. The results indicate that the system is able to learn the task completely, even though more demonstrations have to be performed compared to the light bulb task. The increased number of required demonstrations is not surprising, as the resulting graph has six outgoing transitions for the initial node. The transitions for the lying cylinder depend non-linearly on the cylinder length. The logistic regression models fail to cover this non-linearity. As a result, usually the power grasp is performed when the cylinder is lying. Even though this might also lead to a successful grasp, it is not the behavior that was demonstrated for the medium-length cylinder and is therefore considered

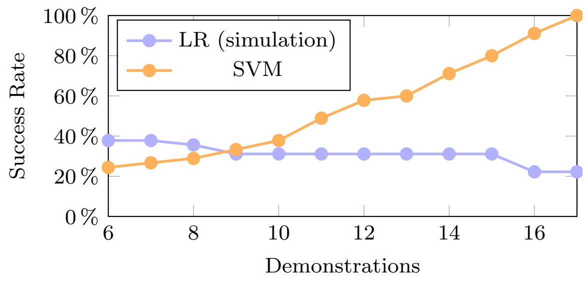


Figure 15: Reproduction results for the grasping task. While an accuracy of 100% can be achieved using SVMs, the logistic regression (LR) models fail to cover the non-linear dependency on the length of the cylinder.

a failure. The experiment shows that the system works well when using positions or joint angles instead of the MP goal distances as features. It also shows that it is easy to integrate arbitrary other features into our framework, which can for example describe the state of the environment. The only requirement of our system is, that the features are meaningful enough, so that successive MPs can be separated well in feature space.

6. Conclusion and Future Work

In this paper, we proposed a method for learning to sequence single movements from kinesthetic demonstrations in order to reproduce a complex task. We showed how the parameters of the single linear attractor movements as well as the transition behavior between the movements can be learned. Learning the transition behavior is formulated as classification problem. We showed how MP sequences observed from kinesthetic demonstrations can be incorporated into one graph representation we call sequence graph. A sequence graph allows to split the overall classification problem into many smaller problems, as it is possible to learn the transition behavior between MPs locally for each node in the graph. We presented two different types of sequence graphs and evaluated four different classifiers. The approach was validated in three experiments using a real seven DOF Barrett WAM robot with a four DOF hand. The results show that our system is able to learn the transition behavior from very few demonstrations.

In future work we want to investigate how an optimal sequence graph can be found, considering not only the MP sequences but also the actual data sampled from the demonstrations. Additionally, we plan to use co-activated MPs and a two-arm setup. Therefore we need methods for synchronizing concurrently active movements.

Acknowledgment

We thank Manuel Mühlig for his help at setting up the magnetic field tracking system and his assistance during the experimental evaluation of the approach.

[1] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robot. Auton. Syst.*, 57(5):469–483, 2009.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., 2006.

[3] R.A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, Mar 1986.

[4] J. Butterfield, S. Osentoski, G. Jay, and O.C. Jenkins. Learning from demonstration using a multi-valued function regressor for time-series data. In *IEEE-RAS Int. Conf. Humanoid Robots*, 2010.

[5] R. J. Carroll and D. Ruppert. *Transformation and Weighting in Regression*. Chapman & Hall, Ltd., 1988.

[6] Guoting Chang and Dana Kulić. Robot task learning from demonstration using petri nets. In *IEEE Int. Symp. Robot and Human Interactive Communication*, 2013.

[7] S. Chiappa and J. Peters. Movement extraction by detecting dynamics switches and repetitions. In *Advances in Neural Information Processing Systems*, 2010.

[8] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[9] S. Ekvall and D. Kragic. Learning task models from multiple human demonstrations. In *IEEE Int. Symposium Robot and Human Interactive Communication*, pages 358–363, 2006.

[10] Tamar Flash and Binyamin Hochner. Motor primitives in vertebrates and invertebrates. *Current Opinion in Neurobiology*, 15(6):660 – 666, 2005.

[11] D.H. Grollman and O.C. Jenkins. Incremental learning of subtasks from unsegmented demonstration. In *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2010.

[12] B. Hayes and B. Scassellati. Discovering task constraints through observation and active learning. In *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2014.

[13] Dana Kulić, Christian Ott, Dongheui Lee, Junichi Ishikawa, and Yoshihiko Nakamura. Incremental learning of full body motion primitives and their sequencing through human motion observation. *Int. J. Rob. Res.*, 31(3):330–345, 2012.

[14] T. Luksch, M. Gienger, M. Muehlig, and T. Yoshiike. Adaptive movement sequences and predictive decisions based on hierarchical dynamical systems. In *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2012.

[15] G.J. Maeda, M. Ewerton, R. Lioutikov, H.B. Amor, J. Peters, and G. Neumann. Learning interaction for collaborative tasks with probabilistic movement primitives. In *Int. Conf. Humanoid Robots*, 2014.

[16] S. Manschitz, J. Kober, M. Gienger, and J. Peters. Learning to sequence movement primitives from demonstrations. In *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2014.

[17] S. Manschitz, J. Kober, M. Gienger, and J. Peters. Learning to unscrew a light bulb from demonstrations. In *ISR/ROBOTIK 2014*, 2014.

[18] Bernard Michini. *Bayesian Nonparametric Reward Learning from Demonstration*. PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, Cambridge MA, August 2013.

[19] Monica N. Nicolescu and Maja J. Mataric. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *Int. Joint Conf. Autonomous Agents and Multi-Agent Systems*, 2003.

[20] Scott Niekum, Sachin Chitta, Andrew Barto, Bhaskara Marthi, and Sarah Osentoski. Incremental semantically grounded learning from demonstration. In *Robotics Science and Systems*, 2013.

[21] A. L. Pais, Keisuke Umezawa, Yoshihiko Nakamura, and A. Billard. Learning robot skills through motion segmentation and constraints extraction. HRI Workshop on Collaborative Manipulation, 2013.

[22] M. Pardowitz, R. Zollner, and R. Dillmann. Learning sequential constraints of tasks from user demonstrations. In *IEEE-RAS Int. Conf. Humanoid Robots*, 2005.

[23] P. Pastor, M. Kalakrishnan, L. Righetti, L. Righetti, and

S. Schaal. Towards associative skill memories. In *IEEE-RAS Int. Conf. Humanoid Robots*, 2012.

[24] Vladimir Pavlovic, James M. Rehg, and John Maccormick. Learning switching linear models of human motion. In *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000.

[25] Jan Peters. Machine learning of motor skills for robotics. *USC Technical Report*, pages 05–867, 2005.

[26] S. Schaal, S. Kotosaka, and D. Sternad. Nonlinear dynamical systems as movement primitives. In *IEEE-RAS Int. Conf. Humanoid Robots*, 2000.

[27] F. Stulp, L. Herlant, A. Hoarau, and G. Raiola. Simultaneous on-line discovery and improvement of robotic skill options. In *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2014.

[28] M. Tenorth and M. Beetz. A unified representation for reasoning about robot actions, processes, and their effects on objects. In *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2012.

[29] Steven D. Whitehead and Dana H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7(1):45–83, 1991.

[30] Ji Zhu and Trevor Hastie. Kernel logistic regression and the import vector machine. In *Journal of Computational and Graphical Statistics*, pages 1081–1088. MIT Press, 2001.

Simon Manschitz studied Information and Systems Technology and received his bachelor's and master's degree from TU Darmstadt, Germany. Since 2014, he works in a joint project of the Intelligent Autonomous Systems group of the TU Darmstadt and the Honda Research Institute Europe in Offenbach, Germany, as a PhD student.

Jens Kober is an assistant professor at the Delft Center for Systems and Control, TU Delft, The Netherlands. He worked as a postdoctoral scholar jointly at the CoR-Lab, Bielefeld University, Germany and at the Honda Research Institute Europe, Germany. He received his PhD in 2012 from TU Darmstadt, Germany. From 2007-2012 he was working at the Department Schölkopf, MPI for Intelligent Systems, Germany. He has been a visiting research student at the Advanced Telecommunication Research (ATR) Center, Japan and an intern at Disney Research Pittsburgh, USA.

Michael Gienger received the diploma degree in Mechanical Engineering from the Technical University of Munich, Germany, in 1998. From 1998 to 2003, he was research assistant at the Institute of Applied Mechanics of the TUM, addressing issues in design and realization of biped robots. He received his Ph.D. degree with a dissertation on "Design and Realization of a Biped Walking Robot". After this, Michael Gienger joined the Honda Research Institute Europe in Germany in 2003. Currently he works as a principal scientist in the field of robotics. His research interests include mechatronics, robotics, control systems, imitation learning and cognitive systems. He also serves as a scientific coordinator for the Research Institute for Cognition and Robotics (CoR-Lab) of the Bielefeld University.

Jan Peters is a full professor (W3) for Intelligent Autonomous Systems at the Computer Science Department of the Technische Universität Darmstadt and at the same time a senior research scientist and group leader at the Max-Planck Institute for Intelligent Systems, where he heads the interdepartmental Robot Learning Group. Jan Peters has received the Dick Volz Best 2007 US PhD Thesis Runner Up Award, the Robotics: Science & Systems - Early Career Spotlight, the INNS Young Investigator Award, and the IEEE Robotics & Automation Society's Early Career Award. Jan Peters has studied Computer Science, Electrical, Mechanical and Control Engineering at TU Munich and FernUni Hagen in Germany, at the National University of Singapore (NUS) and the University of Southern California (USC). He has received four Master's degrees in these disciplines as well as a Computer Science PhD from USC. Jan Peters has performed research in Germany at DLR, TU Munich and the Max Planck Institute for Biological Cybernetics (in addition to the institutions above), in Japan at the Advanced Telecommunication Research Center (ATR), at USC and at both NUS and Siemens Advanced Engineering in Singapore.