Learning to Sequence Movement Primitives from Demonstrations

Simon Manschitz[1,2], Jens Kober[2,3], Michael Gienger[2], and Jan Peters[1]

***Abstract*— We present an approach for learning sequential robot skills through kinesthetic teaching. The demonstrations are represented by a sequence graph. Finding the transitions between consecutive basic movements is treated as classification problem where both Support Vector Machines and Gaussian Mixture Models are evaluated as classifiers. We show how the observed primitive order of all demonstrations can help to improve the movement reproduction by restricting the classification outcome to the currently executed primitive and its possible successors in the graph. The approach is validated with an experiment in which a 7-DOF Barrett WAM robot learns to unscrew a light bulb.**

I. Introduction

Despite the wide use of robots in industry nowadays, their breakthrough in our everyday life is yet to come. One underlying reason is their restriction to a small set of pre-programmed tasks that they are capable to execute very precisely in designated environments. Here, objects can be manipulated by using accurate sensors and well-known (non-)linear controllers. To be applicable more generally, future robots have to learn from observing actions and to generalize observed movements to new situations. Learning from observations is known as imitation learning or learning from demonstrations in robotics [1]. As these approaches can be used as an intuitive programming technique, they are also often referred to as programming by demonstration. The overall concept can be subdivided into different learning schemes depending on the human role in the learning process. Observing and mimicking humans directly is challenging due to the correspondence problem [2] and expensive due to the need of a good measurement system for tracking the movements. We therefore aim for a kinesthetic teaching approach. Here, a human takes the robot by the hand and guides it through the task several times, similar to how parents teach their child a task. Another learning scheme is called reinforcement learning, which allows for a self-exploration of the robot's state space based on the maximization of a reward function. Both learning schemes can also be combined by using imitation learning for bootstrapping the learning process of a reinforcement learning method.

[1]S. Manschitz and J. Peters are with the Institute for Intelligent Autonomous Systems, Technische Universität Darmstadt, 64289 Darmstadt, Germany, manschitz@ias.tu-darmstadt.de, mail@jan-peters.net

[2]S. Manschitz, J. Kober, and M. Gienger are with the Honda Research Institute Europe, 63073 Offenbach, Germany, michael.gienger@honda-ri.de

[3]J. Kober is with the CoR-Lab, Bielefeld University, 33615 Bielefeld, Germany, jkober@cor-lab.uni-bielefeld.de

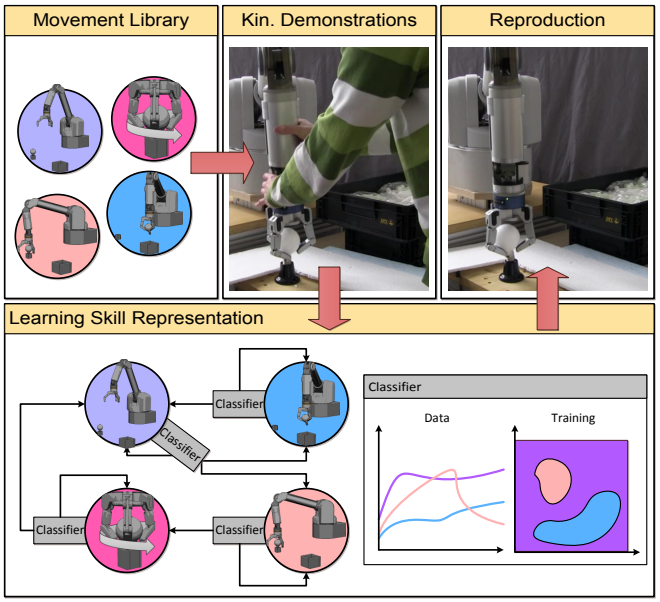


Fig. 1: A 7-DOF WAM arm with a 4-DOF hand has to learn how to unscrew a light bulb from kinesthetic demonstrations. We evaluate our approach with this example in simulation as well as on the real robot.

A. Problem Statement

A sequential skill is the ability to execute basic elementary movements in order to perform a complex task. These movements are often referred to as movement primitives in literature [3], [4]. As we are aiming at learning sequential skills, we assume for simplicity the primitives are given (as they have been previously learned) and we do not have to learn them at this stage. Subdividing a task into smaller parts simplifies the overall problem and introduces a two-level hierarchy, in which the lower-level primitives have to be organized by the upper-level sequencing layer. Among other problems the main question arising when learning sequential skills on the upper-level is: *When to execute which primitive?*

In Sec. II, we present our approach and show one way to answer this question. We use kinesthetic demonstrations as a basis for learning of a skill. The demonstration data is labeled manually and used to create a graph as skill representation (see Fig. 1). The nodes of the graph correspond to movement primitives and the transition conditions are learned by applying machine learning methods. We validate our approach with a set of experiments where a Barrett WAM robot has to unscrew a light bulb. This task requires fine force interaction between the robot and its environment in order to not break the bulb or slip with the fingers

during unscrewing. Also, the sequence of primitives is undetermined beforehand (e.g., the amount of unscrewing repetitions depends on the position of the bulb in its socket). Hence, the task has strong requirements on the generalization capabilities of the algorithm as well as on the accuracy of the whole system.

B. Related Work

The traditional way of modeling skills with a two-level hierarchy is by interpreting the switching behavior as discrete events in a continuous system [5], [6]. Here, an event is often represented as a transition in a directed graph. In [7], an event is added for every observed switch of the demonstration, whereby the transition connects the involved primitives and is labeled with the switching probability. A sequence can then be generated by sampling randomly from the graph. Finite state machines (FSMs) are akin to graphs and can also be used to model transitions between primitives [8].

Graphs can also represent subgoals or constraints of a task [9]–[11]. Such constraints can be used to extract symbolic descriptions which implicitly determine the sequence order. Symbolic approaches can perform sufficiently well for predetermined settings, but lack generality as they rely on predefined assumptions about the tasks. If these assumptions do not apply to the desired task, they are likely to bias the system towards bad decisions.

Instead of modeling the system's policy as an event-based switching behavior, it may be more suitable to treat the overall system as continuous entity. For example, Luksch et al. [12] model the system as a recurrent neural network (RNN) in which primitives can be concurrently activated and are able to inhibit each other. This RNN architecture leads to smooth movements of the robots. The drawback is that their model is hard to learn and the sequence has to be defined by hand. In [13], primitives are encoded as dynamic movement primitives and linked with expected sensory data. Succeeding movements are selected by comparing the current sensor values with the expected ones and choosing the best match. The sequence representation is thus implicit and relies only on the sensor data.

Kinesthetic teaching is a widely used method for learning movements in robotics [14]–[17]. In addition, reinforcement learning allows for self-improvement of the skill and/or the underlying primitives [18]–[20]. It reduces the requirements on the number of necessary demonstrations and makes the robot more independent, but finding the right policies or value and reward functions can be a hard problem.

Most work on sequences of movements concentrates either on segmenting demonstrations into a set of known or unknown movements and/or on learning the individual policies, whereas the sequencing layer is mostly either very simplistic or hand-crafted [7]. In these cases also the switching behavior between movements is either deterministic or not learned at all [21]. The focus of our work instead is on incorporating several demonstrations with varying sequence orders into one graph model and learning the switching behavior between succeeding movements. The most similar work to ours is [8]. Here, a FSM on the sequential level is learned and a k-nearest neighbor classifier is used to learn the switching behavior.

II. Learning Sequential Skills

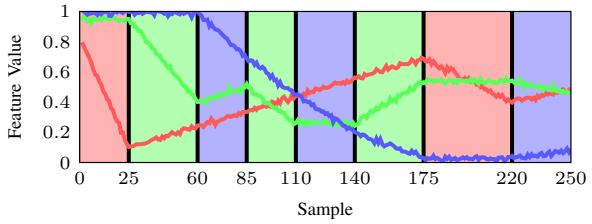
In the following sections we focus on our approach and explain it in more detail. Similar to other work, the switching between primitives is considered to be discrete. Therefore, only one primitive is active at a time and the sequential skill learning algorithm has to decide which primitive to be executed at every time step. The current state of the robot is represented by a set of features, whereby the features are computed from raw sensor values. Finding a mapping between the feature values and executed primitive is essential for the learning algorithm. The straightforward way of applying machine learning methods to this problem would be to train a single classifier with the labeled demonstration data. The skill could then be reproduced by choosing the classification outcome of the current feature values as next executed primitive. However, complex skills involve many different primitives and due to feature ambiguities the classification may yield unsatisfying results. We therefore introduce a graph structure in which each node corresponds to a primitive. The graph is learned from the demonstrations and the classification outcome is restricted to the current node in the graph and its successors as presented in the following.

A. Proposed Approach

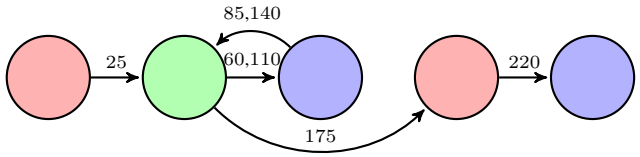
Before going into details about the graph representation and the learning algorithm, an overview of the system is presented first. We assume a predefined set of N primitives denoted as $P = \{p_1, p_2, ..., p_N\}$. In this work a primitive is a dynamical system (DS) with an attractor behavior. Each DS has a goal in task space coordinates that should be reached if the primitive is executed. 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. Primitives may be terminated before their goal is reached, for example if a sensor tells the system that an obstacle is close to the robot. More general, the switching behavior can be triggered based on the state of a feature set denoted as $\boldsymbol{x}$. The features are not global but assigned as output vectors to primitives, leading to one output vector $\boldsymbol{x}_i$ per primitive p_i. A more detailed view on our movement primitive framework can be found in [12]. Please note, however, that our methods are kept general and that they should be applicable to any movement primitive framework and feature set.

Fig. 2 shows the overall flow of our approach based on a simple toy example with only three different primitives. The primitives are indicated by different colors. They are chosen arbitrarily and have no further meaning, but show the essential characteristics of our approach. In general, we assume that M demonstrations have been performed.

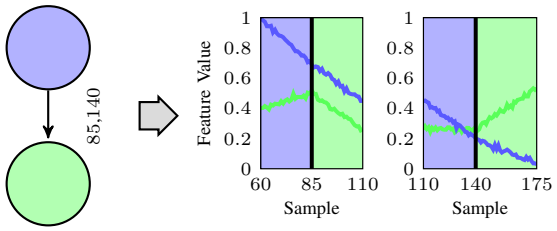
We start with (labeled) sampled data of at least one kinesthetic demonstration (Fig. 2a). Based on the observed sequential ordering of the primitives, the skill then gets represented by a sequence graph in which each node is linked to a primitive (Fig. 2b). The representation will be explained



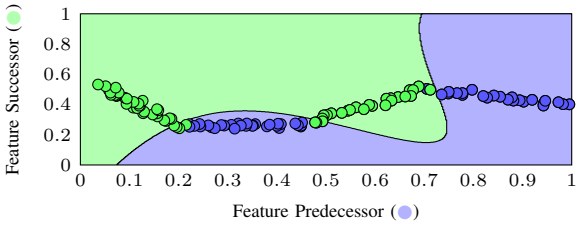
(a) Sampled (labeled) data of one kinesthetic demonstration. The background color indicates the activated primitive, while the plot colors show which feature belongs to which primitive. In this simplified example each primitive has only one associated feature.



(b) Based on the sequential order of the demonstrations, the skill is represented with a sequence graph. The labels correspond to the transitions points (TPs, see left figure). A TP is a point in time at which a switch between primitives occurs.



(c) One classifier is created for each node in the graph. Only the features of the previous primitive and its possible successors are used for training. In this exemplary transition from the upper sequence graph, the red primitive is not involved and hence its feature is not used.



(d) Classification result for a Support Vector Machine. Based on the training data (colored dots) the classifier finds a border separating both classes (background). During reproduction, this border is used to decide either to keep on executing the predecessor primitive or to switch to the successor.

Fig. 2: Overall flow of our framework. First, the labeled data from a set of demonstrations (a) is taken to extract a sequence graph (b). Then, one classifier is created for each transition in the graph based on the linked data of the demonstrations (c, d). The classifiers are then used together with the graph to decide which primitive to execute during reproduction. We propose two different kinds of sequence graphs, as well as two different classifiers.

in detail in the following section, where we also present two different types of sequence graphs, both showing different ways of incorporating the ordering into the representation.

After generating the sequence graph, one classifier is trained for each node in the graph (Fig. 2c). When reproducing the skill, always one node in the graph is considered as active and the corresponding primitive gets executed. The classifier belonging to the node decides at every time step either to continue with the execution of the current primitive or to switch to one of the possible successors in the graph. We evaluate two different types of classifiers: A Gaussian Mixture Model (GMM) and a Support Vector Machine (SVM).

B. Representing Skills with a Sequence Graph

A sequence graph is a directed graph in which each node n_i is linked to a movement primitive. This mapping is not injective which means a primitive can be linked to more than one node. During reproduction, a primitive gets executed if a linked node is considered active. Transitions in the graph lead to succeeding primitives that can be executed if the current primitive 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 Fig. 2a). 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 it is always possible to continue with the execution of the current primitive, the transition $t_{k,k}$ exists for all k.

Before creating a sequence graph, the sequential order S_j for each demonstration is extracted from the sampled data, resulting in one directed acyclic graph with nodes $n_{j,i}$ for each trial. The main step is now to combine these graphs into one representation of the skill, which can be a hard problem as the algorithm has to work solely on the observations. For example, a skill can be shown several times with different sequential orders of the primitives. From the algorithmic point of view it is not clear if the ordering is arbitrary for the 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 and we show two possibilities by investigating two different kinds of sequence graphs: The local graph presumes the ordering to be arbitrary and is not considering it in the representation, while the global graph is trying to construct a more detailed description of the skill based on the ordering of the primitives.

The local sequence graph assigns exactly one node to each executed primitive and hence the number of nodes and primitives is equal. The graph is initialized with one node per primitive and no transitions. For each observed pair of preceding and succeeding primitives 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 Fig. 3.

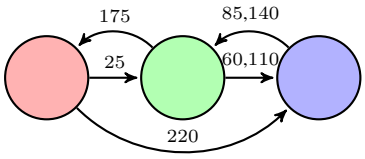


Fig. 3: Local sequence graph for the toy example. Each primitive appears only once in the graph. Thus, there are less nodes and more involved classes for each transition.

The graph contains only three nodes, one for each executed primitive. When reproducing the movement, a switch from the red primitive 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 therefore are the strong requirements on the feature set, as it has to be meaningful enough to allow for a correct classification independent of the current state of the actual skill sequence.

The global sequence graph tries to overcome this issue by constructing a more detailed skill description. One essential characteristic of the global sequence graph is that a node is not only linked to a primitive but can also be considered to be a state of the actual sequence. A primitive can appear multiple times in one representation as depicted in the global sequence graph of the toy example (Fig. 2b). Here, two nodes are linked to the red primitive because the sequence was considered to be in two different states when they were executed. Still, the repeated appearance of the green-blue transitions (see Fig. 2a) is represented by only two nodes as in the local graph. The reason is that consecutive sequences of the same primitives are considered to be a repetition which can be demonstrated and reproduced an arbitrary number of times. Repetitions are also advantageous when describing the task of how to unscrew a light bulb, where you have to repeat the unscrewing movement several times depending on how firm the bulb is in the holder. As the number of repetitions are 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 robot 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 turning off the search for cyclic transitions.

C. Employed Learning Algorithm

As the local sequence graph is created by simply looking at the primitive pairs, we will not go into details here. For creating a global sequence graph three major steps have to be performed:

Algorithm 1 Graph folding

Require: $\boldsymbol{T}$
  $\boldsymbol{S} = \text{getSequenceOrders}(\boldsymbol{T})$;
  $repetition = \text{findRepetition}(\boldsymbol{S})$;
  while $repetition.found$ **do**
    $\boldsymbol{R} = \emptyset$;
    $r = repetition.end - repetition.start + 1$;
    for $i = repetition.start$ **to** $repetition.end$ **do**
      $\text{mergeNodes}(\boldsymbol{S}(i+r), \boldsymbol{S}(i))$;
      $\boldsymbol{R} = \boldsymbol{R} \cup \boldsymbol{S}(i)$;
    $tail = \text{findTail}(\boldsymbol{S}, repetition.end + 1)$;
    $repetition = \text{findRepetition}(\boldsymbol{S})$;
    if !$repetition.found$ **and** $tail.found$ **then**
      for $i = tail.start$ **to** $tail.end$ **do**
        $\text{mergeNodes}(\boldsymbol{S}(i+r), \boldsymbol{S}(i))$;
        $\boldsymbol{R} = \boldsymbol{R} \cup \boldsymbol{S}(i)$;
    $\text{removeNodes}(\boldsymbol{R})$;
    $\boldsymbol{S} = \boldsymbol{S} \setminus \boldsymbol{R}$;

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

The first point is trivial as the acyclic graph represents the primitive orders directly given by the observations. The second point is called *folding* and its pseudo code is shown in Alg. 1. The algorithm starts with the sequential order S with m elements and searches for a repetition of $l = \lfloor m/2 \rfloor$ primitives, meaning longer repetitions are preferred over shorter ones. The method findRepetition starts from the left and compares the primitives 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 $\{n_0, n_{l+1}\} \ldots \{n_l, n_{2l+1}\}$ get merged. If the chains do not match, the starting position is shifted to the right 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. The algorithm terminates if the cycle size is 1, which means no more cycles can be found.

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 the 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, as shown in Fig. 2b. Here, the green-blue repetitions end incompletely with the green primitive.

Algorithm 2 Graph merging

Require: $\boldsymbol{T}_\mathrm{A}$, $\boldsymbol{T}_\mathrm{B}$
  $\boldsymbol{S}_\mathrm{A} = \mathrm{getSequenceOrders}(\boldsymbol{T}_\mathrm{A})$;
  $\boldsymbol{S}_\mathrm{B} = \mathrm{getSequenceOrders}(\boldsymbol{T}_\mathrm{B})$;
  for all $\boldsymbol{s}_\mathrm{B} \in \boldsymbol{S}_\mathrm{B}$ **do**
    $c_\mathrm{max} = 0$;
    for all $\boldsymbol{s}_\mathrm{A} \in \boldsymbol{S}_\mathrm{A}$ **do**
      $c = \sum \mathrm{compare}(\boldsymbol{s}_\mathrm{B}, \boldsymbol{s}_\mathrm{A})$;
      if $c > c_\mathrm{max}$ **then**
        $c_\mathrm{max} = c$;
        $\boldsymbol{s}_\mathrm{A,max} = \boldsymbol{s}_\mathrm{A}$;
    $nodes = 1$;
    for all $i \in \boldsymbol{S}_\mathrm{A,max}$ **do**
      if $nodes \leq c$ **then**
        $\mathrm{mergeNodes}(\boldsymbol{s}_\mathrm{A,max}(i), \boldsymbol{s}_B(i))$;
      else
        $\mathrm{addNode}(\boldsymbol{T}_\mathrm{A}, \boldsymbol{s}_\mathrm{B}(i))$;
      $nodes = nodes + 1$;

The final step of creating a global sequence graph is called *merging*, as several separate graphs are merged into one representation. The algorithm shown in Alg. 2 merges two graphs and thus gets called $M-1$ times for M demonstrations. The goal of the algorithm is to step through the representations simultaneously from left to right, merging equal nodes and introducing branches whenever the nodes differ.

First, the sequence orders are extracted from both graph representations. Here, sequence orders are paths through the graph where only left-to-right transitions are considered. The toy example has two possible orders: red, green, blue and red, green, red, blue. The algorithm considers two nodes as equal if the columns of the corresponding matrices are equal, which means both nodes use the same underlying primitive and have the same input transitions. Before merging the nodes, the algorithm looks for the best match between the sequence orders of both representations. Doing it that way, branches are introduces at the latest possible point in the combined graph. Once branched, both representations are separated and do not get merged together at a later point in the sequence.

After creating the graph representation, the next step is to train the classifiers. Each node has its own classifier which is used if the node is active during reproduction. It decides either to continue with the execution of the current primitive or to switch to a possible successor node. As a node can have more than one outgoing transition, this is a multiclass classification problem with the classes being neighbor nodes in the graph. Due to the graph representation we do not have to learn a overall classification $f(\boldsymbol{x}) = p$ with $p \in P$ and $\boldsymbol{x}$ being the combined output vector of all primitives $\boldsymbol{x} = (\boldsymbol{x}_1^T \ \boldsymbol{x}_2^T \ ... \ \boldsymbol{x}_N^T)^T$, but can restrict the classes c_i of each classifier to a subset $P_i \subseteq P$ and the data vector to the output vectors of the elements in P_i. Restricting the number of classes increases the accuracy of the system as unseen transitions between primitives are prevented. A reduction of the output vector can be seen as intuitive dimensionality reduction, as unimportant features used by uninvolved primitives are not considered for the decision.

Before introducing the classifiers themselves, we show which data is used for the training (see Fig. 2c). 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 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 primitive 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.

We evaluated GMMs and SVMs as classifiers. As both methods are state of the art, we focus on the specifics that are important for our approach. For a deeper insight the interested reader is referred to [22]. When using GMMs, one mixture model is trained per class label. The classification can then be performed by computing the likelihood of a sample given each model and choosing the most likely model as classification result. The number of mixtures within each model is determined using the Bayes Information Criterion (BIC). To prevent one class dominating the others, the number of Gaussians are first computed for each class separately using the BIC and then the maximum is chosen as number of Gaussians for each class.

The behavior of probabilistic generative classifiers is often unpredictable when being faced with unseen data that is in a region of the feature space where absolutely no data has been used for the training. Fig. 4 shows how the GMM approximates the training data of the toy example with Gaussian distributions. Although the data itself is modeled properly, the classification result for data in the white region of the feature space is often unpredictable. We therefore suggest to use SVMs instead of GMMs, as they separate the feature space into hyperplanes and belong to the maximum margin classifiers. Each hyperplane represents one class. Data points are assigned to classes depending on their position in the feature space. We decided to use the freely available *libsvm*

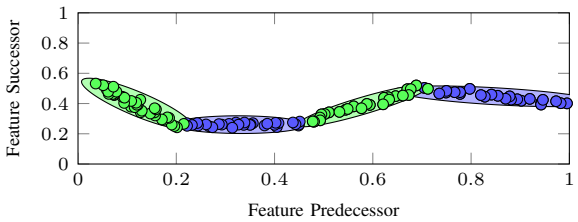


Fig. 4: Classification result of a Gaussian Mixture Model. The GMM approximates the training data (colored dots) with Gaussian distributions (ellipsoids). Despite the correct modeling of the data, the classification result for the white area often is not conclusive.

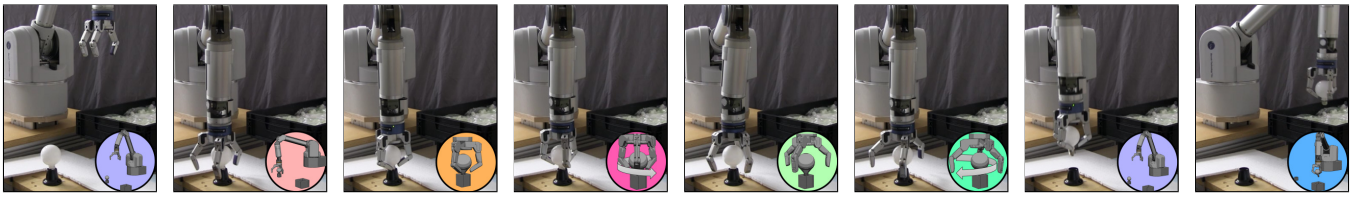


Fig. 5: Illustration of a successful unscrewing sequence. The robot starts in an initial position (●) and first moves towards the bulb (●). Then it repeats the unscrewing movement (●, ●, ●, ●) until the bulb loosens (●) and subsequently, the bulb is put into a bin (●) and the robot returns to its initial position (●).

library [23] as implementation for the SVM and we use radial basis functions as kernels. For the multiclass classification, the standard SVM formulation is used together with the *one-versus-one* concept. Here, for k classes $k(k-1)/2$ binary classifiers are generated. The classification is done for each classifier and the feature vector is assigned to the class that was chosen most frequently.

III. Results

In this section we will present the results of our work. We evaluated our approach both in simulation and with a real 7-DOF Barrett WAM robot with an attached 4-DOF Hand. As a scenario we chose to unscrew a light bulb. In Sec. III-A we outline details of the experiments. The results are then presented in Sec. III-B and discussed in Sec. III-C.

A. Experimental Setup

For the representation of the skill we chose seven different movement primitives. The robot starts in an initial position and first moves towards the bulb. Next, the actual unscrewing movement starts which consists of four different primitives: The hand is closed, rotated counterclockwise, opened and rotated clockwise. These primitives are executed repeatedly until the bulb loosens. Subsequently, the robot puts the bulb into a bin and again returns to its initial position. We chose 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. The detailed task flow is illustrated in Fig. 5.

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 primitive. The unscrewing primitive rotates the closed hand counterclockwise, holds the current horizontal position and applies a force in upward direction to the robot's hand to ensure contact with the bulb. All DOFs that are not controlled by a primitive are handled by the underlying control system.

One feature g is assigned to each primitive. The feature is called goal distance and can be directly derived from the primitive's goal in task space $\boldsymbol{s}_{\text{goal}}$:

$$\boldsymbol{\Delta} = \boldsymbol{s}_{\text{goal}} - \boldsymbol{s} \tag{1}$$

$$g = 1 - \exp(-0.5(\boldsymbol{\Delta}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\Delta})). \tag{2}$$

Here, $\boldsymbol{s} \in \mathbb{R}^n$ is the current state of the robot in task space coordinates and $\boldsymbol{\Sigma}$ is a predefined $n \times n$ diagonal matrix. Eq. (2) depends on the distance between the position of the robot and the primitive's target position. The goal distance has several advantages over using the Euclidean distance as a feature. First, the values are in the range $[0, 1]$ and no further data scaling is necessary. In addition, the feature variation around the robot's goal can be shaped with the parameters of $\boldsymbol{\Sigma}$. For low parameter values, the goal distance starts to decrease only if the robot is already close to its goal.

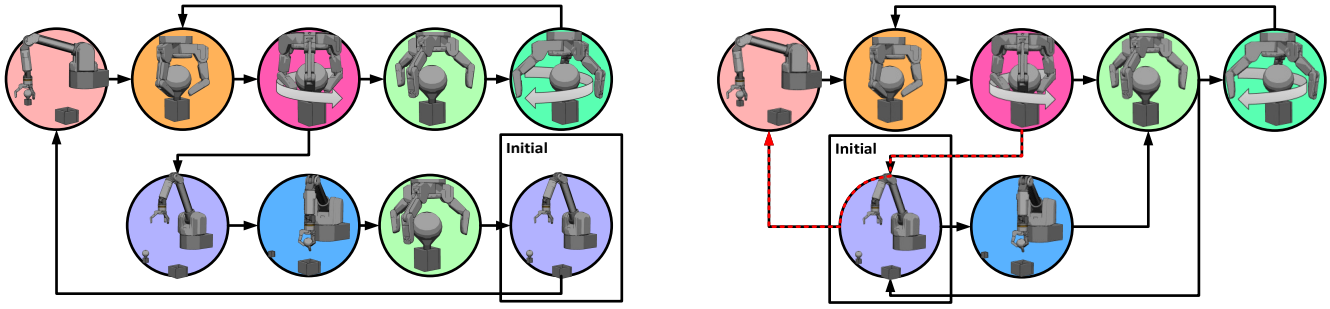
In addition to the goal distances, the velocity of the hand is used as feature to check if the light bulb is loose. To avoid the velocity dominating the other features, it is scaled by subtracting the mean, dividing by the standard deviation and then shifted by 0.5 and clipped to $[0, 1]$. The scaling is done automatically for the complete data used for training the classifiers. Given the goal distances and the velocity of the hand, the overall feature dimension is 8 for seven primitives.

As a kinesthetic teaching is not possible in simulation, we used a teaching method we called automatic demonstration. Here, the primitives are executed in a predefined order using a state machine. For modeling variations in the switching behavior, the transition points are chosen randomly from a certain range. For example when going down to the bulb, the succeeding primitive can be activated if the goal distance of the primitive is in the range $[0, 0.1]$. The exact transition point is chosen randomly from this range every time the primitive starts its execution and hence the transition point could be for example 0.09 and 0.02 for two different demonstrations. The intention of the automatic demonstration was to create a switching behavior which is similar to that of a human teacher.

For the kinesthetic teaching, we activated the gravity compensation mode of the robot and executed the task by guiding its arm. Switches between primitives were indicated by pressing a key every time we considered a movement as complete. We also chose to activate the opening and closing of the hand by pressing a key rather than using compliant fingers in order not to influence the force torque sensor at the wrist. The labeling was then done based on the generated switching points. Examples of a demonstration and successful reproductions can be seen in the attached video.

B. Results

We evaluated our approach with multiple demonstrations along with all possible combinations of classifiers and sequence graphs. For all demonstrations, our approach was able to find the correct sequence graphs of the skill which are shown in Fig. 6. The classification recall for the simulation



(a) Global sequence graph for the light bulb task.

(b) Local sequence graph for the light bulb task.

Fig. 6: Graph representations of the light bulb task. While the global graph gives a complete description of the task, the local graph merges several nodes. The merging creates paths in the representation which were not demonstrated. An example is the sequence marked as red which leads to a misbehavior of the robot if executed.

Demonstration	Ref.	L-GMM	G-GMM	L-SVM	G-SVM
1	92.3	96.1	96.4	96.0	95.8
1+2	95.2	89.9	96.2	93.4	96.7
1+2+3	95.9	89.6	96.1	93.4	96.6
1+2+3+4	96.2	89.1	95.8	93.2	96.6

(a) Classification Recall: Simulation

Demonstration	Ref.	L-GMM	G-GMM	L-SVM	G-SVM
1	56.3	37.5	43.8	81.3	87.5
1+2	81.3	43.8	50.0	93.8	100
1+2+3	62.5	62.5	68.8	93.8	100
1+2+3+4	81.3	62.5	75.0	93.8	100

(b) Reproduction: Simulation

Demonstration	Ref.	L-GMM	G-GMMs	L-SVM	G-SVM
1	91.9	90.0	90.8	90.9	91.4
2	92.7	90.9	91.8	91.0	91.3
3	88.1	90.5	91.7	92.7	92.9
1+2	92.0	86.9	91.4	89.3	91.1
1+3	90.5	86.3	90.2	90.1	92.0
2+3	90.5	87.4	92.9	89.0	92.7
1+2+3	90.5	85.9	91.2	89.4	92.7

(c) Classification Recall: Real Robot

Demonstration	Ref.	L-GMM	G-GMM	L-SVM	G-SVM
1	0.0	53.3	66.7	86.7	93.3
2	0.0	60.0	80.0	86.7	93.3
3	0.0	40.0	40.0	80.0	86.7
1+2	0.0	60.0	80.0	90.0	100
1+3	0.0	53.3	80.0	80.0	100
2+3	0.0	73.3	80.0	80.0	100
1+2+3	0.0	66.7	80.0	90.0	100

(d) Reproduction: Real Robot

TABLE I: Experimental results for the light bulb task. The upper tables show the results for the simulation and the lower ones the results of the experiments with the real robot. For both experiments we tested the classification recall (left tables) as well as the reproduction of the skill (right tables) for SVMs and GMMs both using the global (G-) and local (L-) sequence graph. The results are compared to a reference classifier (Ref.), which is a SVM created without using a sequence graph. For red entries the reproduction failed.

is shown in Table Ia. As reference we trained a SVM without creating a graph representation using the complete labeled data and hence no dimensionality reduction was used.

Table Ib shows the results for the reproduction of the movement. The table outlines the percentage of successfully reproduced transitions between primitives compared to the overall transitions that were necessary to perform the task. If an incorrect movement was chosen or the robot got stuck the transition was marked as faulty. In that case the transition was blocked and triggered manually in the next trial, so that also all succeeding transitions could be tested. Entries marked red indicate dangerous behaviors that could harm the robot or break the light bulb. As our framework allows for incorporating an arbitrary number of demonstrations into one task representation, we were able to test all possible trial combinations. For the teaching, we performed three kinesthetic demonstrations, hence seven different outcomes are possible. Table Ic and Table Id show the classification recall and reproduction results for the experiments on the real robot. The reproduction with the reference classifier failed. Hence, the reproduction rate was set to zero for all entries.

C. Discussion

The benefit of our approach becomes quite obvious when comparing the results of the graph based approaches with the reference classifier. Despite achieving comparable classification recall, the reproduction with the reference classifier works worse in simulation and not at all on the real robot, as indicated by the red entries in Table I.

The presented results also emphasize the advantages of the global over the local sequence graph as well as of the SVMs over the GMMs. No local representation was able to reproduce the task successfully and when using GMMs even dangerous behaviors were possible. The reason is the fusion of the initial state and the return to the initial position after unscrewing the light bulb in the local representation (see Fig. 6). The fusion allows wrong sequences and the potentially dangerous sequence is marked as red in the image. 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. The misclassification comes from the features which are not meaningful enough to be separated

well. While the global representation overcomes this issue by disallowing the incorrect transition, the local representation is not able to encode the skill properly without using additional features. Note that the global representation performs only better if the skill is modeled properly as in our case. When introducing false transitions the system can be biased towards wrong decisions, resulting in a worse performance than by using the more general local representation.

GMMs perform worse than SVMs, but still manage to reproduce most transitions for the global representation if enough demonstrations are available. As the training of each state's GMM is done locally, the models are prone to overfit the data and the behavior for unseen data is uncertain in advance (see Fig. 4). Due to the overfitting GMMs achieve a similar classification recall as SVMs, but the reproduction is not competitive. To avoid the overfitting, we suggest not using the expectation-maximization algorithm for the training of a GMM and recommend training methods which are trying to find large margins (e.g., [24]) or other probabilistic classifiers such as logistic regression. The overall winner is the combination of SVMs and global sequence graph. Only two trials were necessary to perform the skill properly both in simulation and with the real robot. Although the other approaches were not able to execute the overall task completely, they were able to unscrew the light bulb, which was the main goal of the project. The fixed amount of seen unscrewing repetitions were generalized to an arbitrary number and most approaches were able to recognize when the bulb was loose.

IV. Conclusion and Future Work

In this paper, we proposed to use a graph structure for representing sequences of robot movements. Based on this, a sequential manipulation skill was learned by creating a classifier for each node in the graph, which decided either to continue with the execution of the current movement or to switch to another one by taking a transition in the graph. We showed how the observed sequence order of kinesthetic demonstrations can be incorporated into the graph representation. This leads to an intuitive class and dimensionality reduction which is the main benefit of our approach and allows for a reproduction of the skill. We evaluated two different classifiers (SVMs and GMMs) for their skill learning suitability, as well as two different types of sequence graphs. Our approach was validated with an experiment in which the robot unscrews a light bulb, both in simulation and with a real Barrett WAM. Here we showed, that our approach is able to learn the switching behavior from very few demonstrations. Most often two demonstrations were enough to reproduce the demonstrated skill properly.

In future work some simplifications made in this paper will be relaxed. One target is to consider more complex structures within the folding and merging algorithms, e.g., by reuniting branches or looking for common substrings. We also aim at learning skills that require co-articulation and parallel execution of primitives. Therefore we have to synchronize concurrently active primitives which for example control two different end effectors.

References

[1] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robot. Auton. Syst.*, vol. 57, no. 5, pp. 469–483, 2009.

[2] C. Nehaniv and K. Dautenhahn, *Imitation in animals and artifacts*, ch. The Correspondence Problem, pp. 42–61. MIT Press, 2002.

[3] T. Flash and B. Hochner, "Motor primitives in vertebrates and invertebrates," *Current Opinion in Neurobiology*, vol. 15, no. 6, pp. 660 – 666, 2005.

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

[5] J. Peters, "Machine learning of motor skills for robotics," *USC Technical Report*, pp. 05–867, 2005.

[6] V. Pavlovic, J. M. Rehg, and J. Maccormick, "Learning switching linear models of human motion," in *Advances in Neural Information Processing Systems*, vol. 13, MIT Press, 2000.

[7] D. Kulić, C. Ott, D. Lee, J. Ishikawa, and Y. Nakamura, "Incremental learning of full body motion primitives and their sequencing through human motion observation," *Int. J. Rob. Res.*, vol. 31, no. 3, pp. 330–345, 2012.

[8] S. Niekum, S. Chitta, A. Barto, B. Marthi, and S. Osentoski, "Incremental semantically grounded learning from demonstration," in *Robotics Science and Systems*, 2013.

[9] S. Ekvall and D. Kragic, "Learning task models from multiple human demonstrations," in *Int. Symp. Robot and Human Interactive Communication*, 2006.

[10] M. N. Nicolescu and M. J. Mataric, "Natural methods for robot task learning: Instructive demonstrations, generalization and practice," in *Int. Joint Conf. Autonomous Agents and Multi-Agent Systems*, 2003.

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

[12] 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.

[13] P. Pastor, M. Kalakrishnan, L. Righetti, L. Righetti, and S. Schaal, "Towards Associative Skill Memories," in *IEEE-RAS Int. Conf. Humanoid Robots*, 2012.

[14] A. Billard, S. Calinon, and F. Guenter, "Discriminative and adaptive imitation in uni-manual and bi-manual tasks," *Robotics and Autonomous Systems*, vol. 54, no. 5, pp. 370–384, 2006.

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

[16] S. Calinon and A. Billard, "Active teaching in robot programming by demonstration," in *IEEE Int. Symp. on Robot and Human*, 2007.

[17] J. Kober, K. Muelling, O. Kroemer, C. H. Lampert, B. Schölkopf, and J. Peters, "Movement templates for learning of hitting and batting," in *IEEE Int. Conf. Robotics and Automation*, 2010.

[18] C. Daniel, G. Neumann, O. Kroemer, and J. Peters, "Learning sequential motor tasks," in *IEEE Int. Conf. Robotics and Automation*, 2013.

[19] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal, "Skill learning and task outcome prediction for manipulation," in *IEEE. Int. Conf. Robotics and Automation*, 2011.

[20] J. Morimoto and K. Doya, "Reinforcement learning of dynamic motor sequence: Learning to stand up," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 1998.

[21] D. Grollman and O. Jenkins, "Incremental learning of subtasks from unsegmented demonstration," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2010.

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

[23] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 1–27, 2011.

[24] F. Sha and L. K. Saul, "Large margin gaussian mixture modeling for phonetic classification and recognition," in *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, 2006.