ISR/ROBOTIK 2014

Learning to Unscrew a Light Bulb from Demonstrations

Simon Manschitz, Technische Universität Darmstadt, Germany
Jens Kober, Bielefeld University, Germany
Michael Gienger, Honda Research Institute Europe, Offenbach, Germany
Jan Peters, Technische Universität Darmstadt, Germany
E-Mail: manschitz@ias.tu-darmstadt.de **Web**: www.ias.tu-darmstadt.de

Abstract

In this paper we show a way of learning how to sequence predefined basic movements in order to reproduce a previously demonstrated skill. Connections between subsequently executed movements are learned and represented in a graph structure. Learning the switching behavior between connected movements is treated as classification problem. Due to the graph, the overall accuracy of the system can be improved by restricting the possible outcomes of the classification to connected movements. We show how the graph representation can be learned from the observations and evaluate Support Vector Machines as classifier. The approach is evaluated with an experiment in which a 7-DOF Barrett WAM robot learns to unscrew a light bulb.

1 Introduction

Programming by demonstration (PBD) is an intuitive way of programming a robot. Here, a teacher demonstrates a skill to a robot and from observing these demonstrations the robot learns how to perform the skill by itself. As a consequence, there is no need to program a robot in the old-fashioned way where the programmer has to be an expert in the robotics domain. The demonstrations allow for creating a connection between perception and action. Due to this connection it is possible for a robot to learn how to react to its environment. Applying this learning method can therefore lead to more flexible robots and this is why PBD has been a hot topic in robotic research over the past years. Other names for PBD found in the literature are imitation learning or learning from demonstrations [1].

One possibility for a teacher to demonstrate a skill is to perform the task by himself. In that case, the robot needs a measurement system for tracking the movements of the teacher. Such systems are expensive and a mapping between the measured positions of the human body and the robot's joints has to be found, which is complicated due to the correspondence problem [2]. We therefore perform the demonstrations kinesthetically. Here, the teacher guides the robot through the movement by taking its arm, similar to how parents teach their child a task.

1.1 Problem Statement

Solving a complex task by executing basic movements subsequently sounds very intuitive to us, as this divide and conquer strategy is a typical problem solving approach of humans. Basic movements are often referred to as movement primitives (MPs) in literature [3, 4]. As we aim at sequencing MPs, we assume for simplicity they are given and we do not have to learn them at this stage (as they have been previously learned). The main question tackled by this work therefore is: *When to execute which primitive?* The goal is to find an answer to this question from the data sampled by kinesthetic demonstrations, without further knowledge about the actual task.

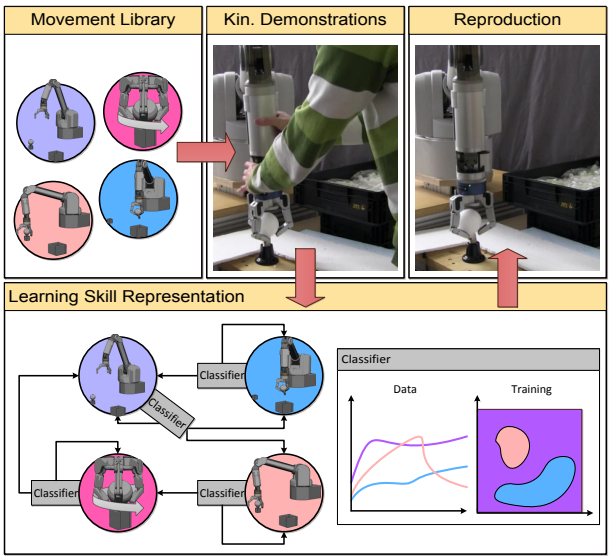

Figure 1: A 7-DOF WAM arm with a 4-DOF hand has to learn how to unscrew a light bulb from kinesthetic demonstrations. We evaluate our approach with this example in simulation as well as on the real robot.

An overview of the approach is shown in **Figure 1**. The demonstration data is labeled manually, which means it is clear at every point in time which MP was active during the demonstrations. Based on this data, a graph is created as representation of the skill. The nodes of the graph correspond to MPs and the transition conditions are learned by applying machine learning methods. The approach is validated with an experiment 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 MPs is undetermined

beforehand as 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.

1.2 Related Work

Work on organizing and sequencing sets of MPs can be roughly categorized into two categories, both having different views on the learned system. The traditional view on such a system is interpreting the switching behavior as discrete events in a continuous system [5, 6]. Here, only one MP is active at the same time and the connections between the MPs are often graph based. In [7], an event is added for every observed switch of the demonstration, whereby the transition connects the involved MPs and is labeled with the switching probability. A sequence can then be generated by sampling randomly from the graph. Finite state machines are akin to graphs and can also be used to model transitions between MPs [8].

The second possible view on the system is to see it as completely continuous entity. Here, MPs can be concurrently active, meaning that each MP can decide for itself whether it should be active or not. Additionally, MPs usually can be gradually active. The resulting behavior is a superposition of all (partially) active MPs and the sequence is therefore mostly implicitly defined. For example, the authors of [9] model the system as a recurrent neural network (RNN) in which MPs 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 [10], MPs are encoded as Dynamic Movement Primitives (DMPs) 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.

Although much effort has been put into learning single movements of various complexity (e.g., [11, 12, 13]), research on sequences of movements is usually restricted to very simple skills. For such skills, the switching behavior is mostly either hand-crafted or the order of movements is always the same (e.g., [14, 15]). Closest to our work is the work presented in [16]. Here, a task is represented with a FSM that is learned from the demonstrations. The switching behavior is also treated as classification problem, but the classification is not performed at every time step. Due to that, their approach is not suitable for tasks that require finer decisions at a smaller scale. Other authors use graphs to represent sub-goals or constraints of a task. In that case, a transition is triggered if a goal is reached or all constraints are fulfilled [17, 18, 19]. While such approaches can lead to simple models, their flexibility is usually limited.

2 Learning Sequential Skills

Next, we focus on our approach and explain it in detail. In Section 2.1, the graph representation is introduced and formalized. Section 2.2 shows how a graph can be learned from the demonstration data. The final step then is to learn the switching behavior between connected MPs in the graph, which is the topic of Section 2.3.

We denote a MP as p_i and the set of MPs as $P = \{p_1, p_2, ..., p_n\}$. In this work a MP is a dynamical system (DS) with an attractor behavior. Each DS has a goal in task space coordinates that should be reached if the MP 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. The feature set is denoted as $\boldsymbol{x} \in \mathbb{R}^n$. The features are not global but assigned as output vectors to MPs, leading to one output vector $\boldsymbol{x}_i$ per MP p_i.

In our approach, the switching behavior is considered to be discrete. Therefore only one MP is active at the same time and it is clear from the labels of the demonstration data which MP was active at each point in time. **Figure 2** shows such exemplary demonstration data for a simple toy example. The example will be used throughout this section and consists of three different MPs, indicated by different colors. The MPs are chosen arbitrarily and have no further meaning, but show the essential characteristics of our approach. The labels are depicted by the background color. Black vertical lines indicate a switch between MPs and are referred to as transition points (TPs) in the following. The feature dimension is three, whereas every MP has one associated feature.

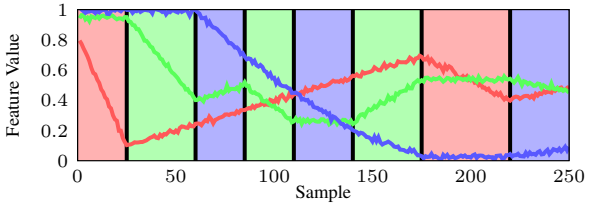

Figure 2: Sampled (labeled) data of one demonstration. The background color indicates the activated MP, while the plot colors show which feature belongs to which MP. In this simplified example each MP has only one associated feature.

For every point in time, the feature vector determines the current state of the robot. The learning algorithm has to find a mapping between the state of the robot and the MP to be executed in order to sequence the MPs properly. 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 MP. However, complex skills involve many different MPs and due to feature ambiguities the classification may yield unsatisfying results. We therefore introduce a graph structure in which each node corresponds to a MP. 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.

2.1 Representing Skills with a Sequence Graph

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 gets executed if a linked node is considered active. Transitions in the graph lead to succeeding MPs that can be executed 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 TP at which it was observed during the demonstration. As the same transition can be observed multiple times, multiple TPs are possible.

Having m nodes in the graph, we use a $m \times m$ transition matrix $\boldsymbol{T}$ to describe one sequence graph. As it is always possible to continue with the execution of the current MP, the transition $t_{k,k}$ exists for all k.

Before creating a sequence graph, the sequential order S_i for each demonstration is extracted from the sampled data, resulting in one directed acyclic graph with nodes n_i for each trial. The main step now is 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 MPs. From the algorithmic point of view it is not always clear if the ordering is arbitrary for the skill or if the differences can be linked to some traceable sensor events. Hence, a heuristic is needed which interprets the ordering in cases where multiple options are possible. We show one possible heuristic by investigating a graph structure we call global sequence graph (GSG).

The GSG for the toy example is shown in **Figure 3**. One essential characteristic of a GSG is that a node is not only linked to a MP but can also be considered to be a state of the actual sequence. A MP can therefore appear multiple times in one representation as depicted in the figure. Here, two nodes are linked to the red MP because the sequence was considered to be in two different states when they were executed.

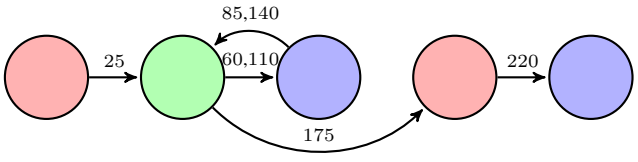

Figure 3: Based on the sequential order of the demonstrations, the skill is represented with a sequence graph. The transitions are linked with the corresponding data points of the transitions.

The repeated appearances of the green-blue transitions (see Figure 2) instead are represented by one node per MP and two transitions, creating a cycle in the graph. The reason is that consecutive executions of the same sequence of MPs are considered to be a repetition of the same movement. Such a movement can be demonstrated an arbitrary number of times, but corresponds to one state of the sequence. As soon as a repetition is interrupted by switching to a different MP, the system is considered to be in a new state. Therefore, a transition to a new node linked to the red MP is added instead of a transition leading back to the existing one.

Repetitions are also advantageous when describing the task of unscrewing a light bulb, where the unscrewing movement has to be repeated 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.

2.2 Learning the Skill Representation

For creating a GSG three major steps have to be performed:

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

The first point is trivial as the acyclic graph represents the MP orders directly given by the observations. Thus, the sequential orders can be taken directly from the main diagonal of $\boldsymbol{T}$. The second point is called *folding* and its pseudo code is shown in **Algorithm 1**. The algorithm starts with the sequential order S with n elements and searches for a repetition of $l = \lfloor n/2 \rfloor$ MPs, meaning longer repetitions are preferred over shorter ones. The method $\mathrm{findRepetition}$ starts from the left and compares 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 $\{n_0, n_{l+1}\} \ldots \{n_l, n_{2l+1}\}$ 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 bend 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.

Algorithm 1 Graph folding

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

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. The toy example also contains an incomplete cycle, as the green-blue repetitions end incompletely with the green MP (see Figure 2). Without considering incomplete repetitions, the GSG would contain two nodes for the green MP, both having incoming transitions from the first blue MP. In that case, the decision whether to escape the cycle had to be made when the blue MP is executed. During reproduction of the skill, the sensor events necessary for leaving or staying in the cycle might not be available at the time. As a consequence, the system may get stuck in the loop or is not entering the loop at all.

The final step of creating a GSG is to combine the graphs of each demonstration to one overall representation of the demonstrated skill. This step is called *merging*, as several separate graphs are merged into one graph. **Algorithm 2** merges two graphs and thus gets called $n-1$ times for n demonstrations. The algorithm works as follows. The goal is to step through both graphs simultaneously from left to right, merging equal nodes and introducing a branch as soon as two nodes differ. Two nodes are considered as equal if the columns of the corresponding matrices are equal, which means both nodes use the same underlying MP and have the same input transitions. Two graphs are compared with each other by extracting their sequence orders first. A sequence order is a path through a 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 then algorithm looks for the best match between the sequence orders of both representations. At the point where the best matching orders differ, a branch is introduced.

Algorithm 2 Graph merging

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

2.3 Learning the Switching Behavior

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 MP 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 MPs $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)^T$, 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 MPs are prevented. A reduction of the output vector can be seen as intuitive dimensionality reduction, as unimportant features used by uninvolved MPs are not considered for the decision. Note that our approach is not requiring an assignment of features to MPs in general. However, our opinion is that such local feature sets allow for a more flexible control of the features used for a decision, due to the dimensionality reduction.

Before introducing the classifiers themselves, we show which data is used for the training (see **Figure 4**). 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 GSG transitions are merged together, resulting in potentially multiple TPs for a 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.

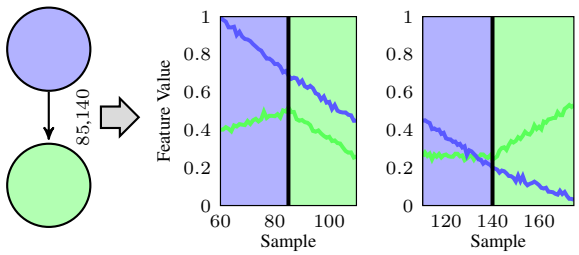

Figure 4: One classifier is created for each node in the graph. Only the features of the previous MP and its possible successors are used for training. In this exemplary transition from the upper sequence graph, the red MP is not involved and hence its feature is not used.

For the classification, Support Vector Machines (SVMs) are used. As this classifier is 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 [20]. SVMs are trying to separate the feature space into hyperplanes and belong to the maximum margin classifiers. Each hyperplane represents one class and data points are assigned to classes depending on their position in the feature space. We decided to use the freely available *libsvm* library [21] as implementation for the SVM and we use radial basis functions as kernels:

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \exp(-\gamma \|\boldsymbol{x}_n - \boldsymbol{x}_m\|^2), \gamma > 0 \qquad (1)$$

For the multiclass classification, the standard SVM formulation is used together with the *one-versus-one* concept. Here, for c classes $c(c-1)/2$ 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. The classification result for the exemplary transition of the toy example is depicted in **Figure 5**. The SVM is able to separate the feature space into two different areas as indicated by the different background colors.

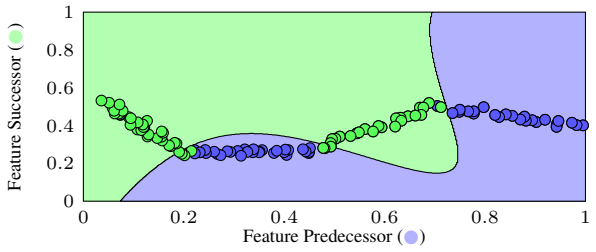

Figure 5: Classification result for the SVM. Every sample from the training data is plotted with colored dots in the feature space. The colors again show the label of each data point. The classifier finds a border separating both classes (background). As soon as the current feature vector enters the green region during reproduction, the preceding blue MP is stopped and the succeeding green MP gets executed. The system then switches to the classifier linked to the successor node in the graph.

3 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 Section 3.1 we outline details of the experiments. The results are then presented and discussed in Section 3.2.

3.1 Experimental Setup

For the representation of the skill we chose 7 different MPs. The detailed task flow is illustrated in **Figure 6**. 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. When unscrewing the bulb (rotating the closed hand counterclockwise), a force in upward direction is applied to the robot's hand to ensure contact with the bulb.

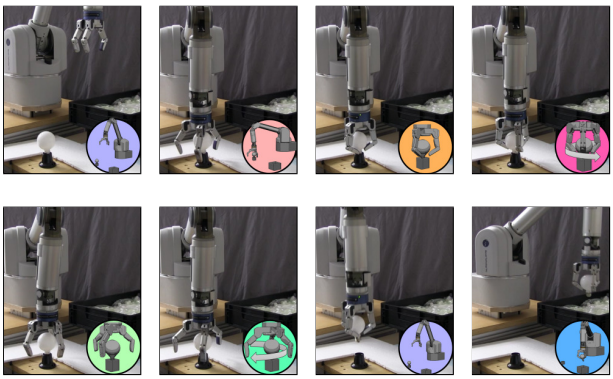

Figure 6: Illustration of a successful unscrewing sequence. The robot starts in an initial position (●) and first moves towards the bulb (●). Then it repeats the unscrewing movement (●, ●, ●, ●) until the bulb loosens (●) and subsequently, the bulb is put into a bin (●) and the robot returns to its initial position (●).

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

$$\boldsymbol{\Delta} = \boldsymbol{x}_{\text{goal}} - \boldsymbol{x} \qquad (2)$$

$$g = 1 - \exp(-0.5(\boldsymbol{\Delta}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\Delta})). \qquad (3)$$

Here, $\boldsymbol{x} \in \mathbb{R}^n$ is the current state of the robot in task space coordinates and $\boldsymbol{\Sigma}$ is a manually defined $n \times n$ diagonal matrix. Equation (3) is dependent on the distance between the position of the robot and the MP'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 7 MPs.

As a kinesthetic teaching is not possible in simulation, demonstrations were performed by executing the MPs in a predefined order using a state machine. For modeling variations in the switching behavior, the transition points were chosen randomly from a certain range. For example when going down to the bulb, the succeeding MP could be activated if the goal distance of the MP was in the range [0, 0.1]. The exact transition point was chosen randomly from this range every time the MP started its execution and hence possible transition points were for example 0.02 and 0.09 for two different demonstrations. The intention of this 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 MPs 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.

3.2 Results and Discussion

We evaluated our approach with data from up to 3 demonstrations. For all demonstrations, our approach was able to find the correct GSG of the skill which is shown in **Figure 7**.

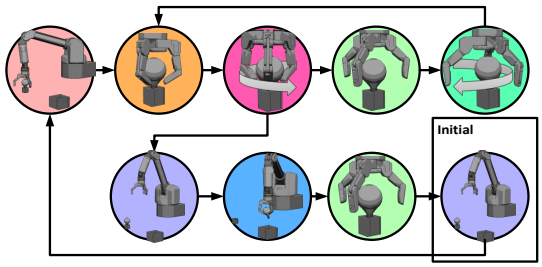

Figure 7: Graph representations of the light bulb task.

Figure 8 shows the classification recall of the simulation and the experiments with the real Barrett WAM robot. As reference a single SVM was used, trained with the complete data of the demonstrations (no sequence graph was used). The classifier was solely active during the entire reproduction and could choose every MP at every point in time. Therefore all MPs were involved in the decision and no dimensionality reduction could be used.

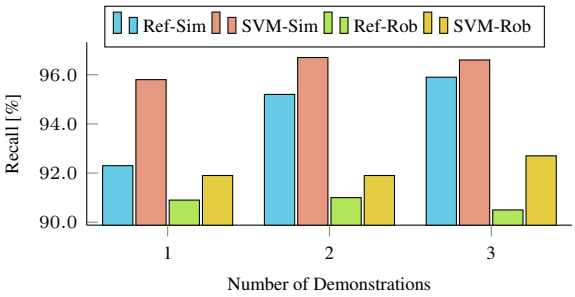

Figure 8: Classification recall in % based on 1–3 demonstrations for simulation (-Sim) and the experiments with the real Barrett WAM robot (-Rob). The reference classifier (Ref-) is a SVM which was created without using a sequence graph.

The results for the reproduction of the movements are depicted in **Figure 9**. The figure outlines the percentage of successfully reproduced transitions between MPs 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. First tests showed that the reference classifier did not provide useful results for the experiments on the real robot. In order to not harm the robot we therefore skipped it for the evaluation.

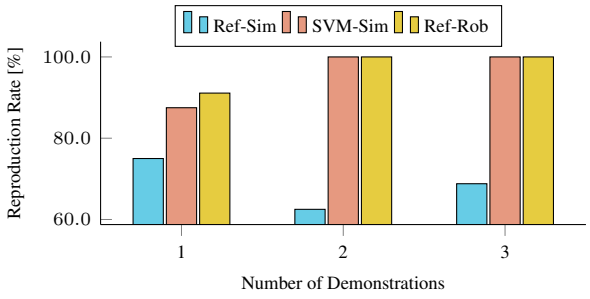

Figure 9: Reproduction results in % of different demonstrations. Shown are the successfully performed MP switches compared to the overall switches. The reference classifier did not produce any useful results and is therefore not shown here.

The presented results show, that our system is able to reproduce the demonstrated manipulation skill. Only two trials were necessary to perform the skill properly both in simulation and with the real robot. The fixed amount of seen unscrewing repetitions were generalized to an arbitrary number and the system was able to recognize when the bulb was loose. It is notable here that of course not every demonstration works for 100% as indicated by the results. Instead, the success also relies on the teacher. For example, attention has to be paid on the velocity of the robot's hand. If the arm of the robot is moved too fast when the light bulb gets loose, the actual velocity reached during reproduction is not matching the expectations. The reason is that the expected velocity is learned from the demonstrations, while the reached velocity is

a consequence of the applied force in upward direction during reproduction. This constant force is part of the MP and predefined in our system. The potential misbehavior therefore is not a classification error but caused by the general drawback that occurs with predefined MPs.

4 Conclusion and Future Work

In this paper, we showed how a sequential manipulation skill can be learned from kinesthetic demonstrations.
A skill is represented in a graph structure and the switching behavior between basic movements is treated as classification problem. For the classification, SVMs were used. We showed how the observed sequence order of the demonstrations can be incorporated into the graph structure, leading to an intuitive class and dimensionality reduction of the classifiers. Our approach was validated with an experiment in which the robot unscrews a light bulb, both in simulation and with a real Barrett WAM robot.
In future work some simplifications made in this paper will be relaxed. We aim at learning more complex skills that require co-articulation and parallel execution of MPs. Therefore we have to synchronize concurrently active MPs which for example control two different end effectors. Additionally, we plan to use a segmentation algorithm instead of labeling the data manually.

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.

[6] V. Pavlovic, J. M. Rehg, and J. Maccormick, "Learning switching linear models of human motion," pp. 981–987, 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] L. Riano and T. M. McGinnity, "Automatically composing and parameterizing skills by evolving finite state automata," *Robot. Auton. Syst.*, vol. 60, no. 4, pp. 639–650, 2012.

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

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

[11] S. Calinon, F. D'halluin, E. Sauser, D. Caldwell, and A. Billard, "Learning and reproduction of gestures by imitation," *Robotics Automation Magazine, IEEE*, vol. 17, no. 2, pp. 44–54, 2010.

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

[13] J. Ernesti, L. Righetti, M. Do, T. Asfour, and S. Schaal, "Encoding of periodic and their transient motions by a single dynamic movement primitive," in *IEEE-RAS Int. Conf. Humanoid Robots*, 2012.

[14] 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.

[15] D. Forte, A. Gams, J. Morimoto, and A. Ude, "On-line motion synthesis and adaptation using a trajectory database," *Robotics and Autonomous Systems*, vol. 60, no. 10, pp. 1327 – 1339, 2012.

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

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

[18] 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.

[19] U. Thomas, B. Finkemeyer, T. Kröger, and F. M. Wahl, "Error-tolerant execution of complex robot tasks based on skill primitives," in *IEEE/RSJ Int. Conf. Robotics and Automation*, 2003.

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

[21] 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.