Learning Sequential Skills for Robot Manipulation Tasks

Lernen von sequentiellen Fähigkeiten für Roboter-Manipulationsaufgaben

Master-Thesis von B.Sc. Simon Manschitz

Januar 2014

TECHNISCHE UNIVERSITÄT DARMSTADT

Department of Computer Science
Intelligent Autonomous Systems

Learning Sequential Skills for Robot Manipulation Tasks
Lernen von sequentiellen Fähigkeiten für Roboter-Manipulationsaufgaben

Vorgelegte Master-Thesis von B.Sc. Simon Manschitz

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Dr.-Ing. Michael Gienger
3. Gutachten: Dr.-Ing. Jens Kober

Tag der Einreichung:

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 30. Januar 2014

(Simon Manschitz)

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 Hidden Markov Models are evaluated as classifiers. We show how the observed primitive order of all trials can help to improve the classification results by choosing the feature vector depending on the current primitive and its possible successors in the graph. The approach is validated with an experiment in which a seven degree of freedom Barrett WAM robot learns to unscrew a light bulb.

Contents

1 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. In these environments, 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 [Argall et al., 2009]. 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 [Nehaniv and Dautenhahn, 2002] and expensive due to the need of a good measurement system for tracking the movements. We therefore employ 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.

1.1 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 (MPs) in literature [Flash and Hochner, 2005, Schaal et al., 2000]. As we are aiming at learning the 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 the 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 that arises when learning sequential skills on the upper-level is:

1. When to stop the execution of the current movement?
2. Which primitive to execute next?

These two questions can be treated as one single problem, leading to the question:

- When to execute which primitive?

In Section 2, we present our approach and show how this question can be answered. We use kinesthetic demonstrations to learn a skill. The demonstration data is labeled manually and used

to create a graph as skill representation (see Figure 1.1). 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 experiments where a Barrett whole arm manipulator (WAM) robot with seven degrees of freedom (DoF) 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 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.

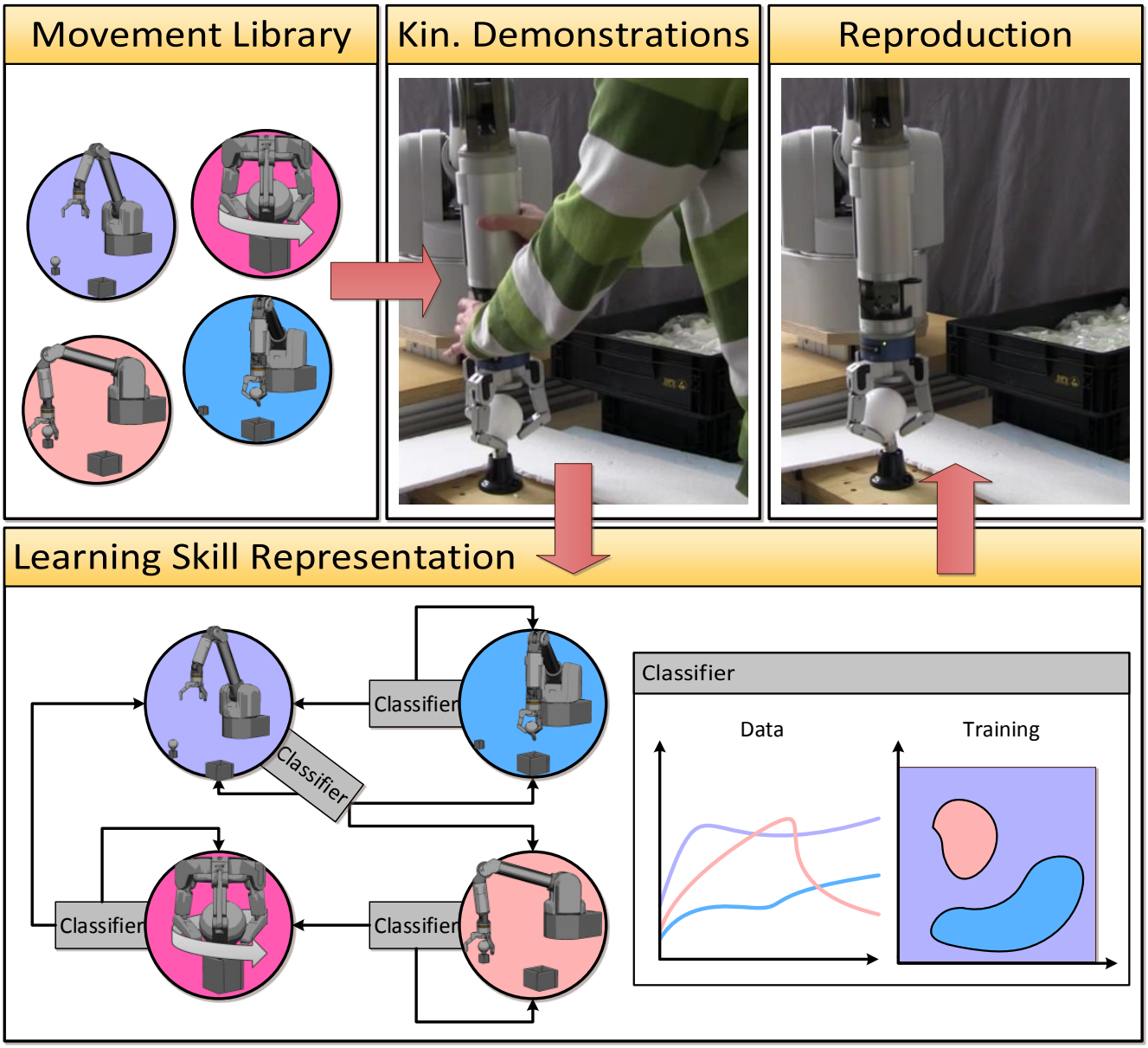


Figure 1.1.: A 7-DoF WAM arm with a 4-DoF hand has to learn how to unscrew a light bulb from kinesthetic demonstrations. We evaluate our approach with this example in simulation as well as on the real robot.

1.2 Related Work

When performing manipulation tasks, the human brain seems to learn a connection between an action and its corresponding sensor signals [Flanagan et al., 2006]. The learned model can be used for a comparison of expected and actual sensor signals when reproducing the movement. This comparison enables a monitoring of the movement progress as well as for error detection. The remarkable object manipulation abilities of humans comes from their capabilities of adapting the models over time and to generalize it to new situations. Instead of learning one complex model, the task is separated into smaller subgoals such as contact events. In recent years, robot researchers tried to transfer these abilities to the robotic domain. Therefore, a lot of effort has been put into segmenting a demonstrated skill into smaller parts as well as on building models of basic movements.

Dynamic movement primitives (DMP) are arguably the most prominent models for basic movements [Degallier and Ijspeert, 2010]. A DMP consists of a set of differential equations that encode an attractor behavior. A brief summary of the equations can be found in Appendix A.1 The models are robust to perturbations and and therefore widely used (e.g., Forte et al. [2012], Muelling et al. [2010], Ude et al. [2010]). Also, many extensions of the original formulation exist, such as encoding periodic motions [Ernesti et al., 2012] or joining consecutive primitives with a non-zero velocity [Nemec and Ude, 2012]. Other prominent models used for MPs are Hidden Markov Models [Kulic et al., 2007] or Gaussian Mixture Models [Calinon et al., 2007].

Segmenting a skill into smaller parts boils down to the question of finding the transition points between consecutively executed primitives. In [Pais et al., 2013], a task is performed several times and the variance of the parameters over time and trials is measured. The variance is used as an indicator how important a certain parameter is for a certain part of the task. A task is segmented into several parts by looking at changes of the control mode (position based versus force based) and changes of the reference frame (which object is involved) or variables of interest. In [Kulic et al., 2008] and [Takano and Nakamura, 2006], a primitive is considered to be a data segment of fixed length in time. As this might produce good results for a predefined skill, the resulting primitives are usually not generic enough to be applied to other tasks. Additionally, the data has to be aligned in time before the segmentation to reduce the time-variations over different demonstrations. Another possibility of finding transition points is by looking at the velocity profile of sensor signals. In [Meier et al., 2012], a zero-crossing of a velocity is considered to be a candidate of a primitive transition.

Many segmentation approaches additionally use a library of MPs. The authors of [Meier et al., 2011] and [Meier et al., 2012] use a predefined library where the MPs are taught in isolation. The segmentation than can be performed by finding the best matching sequence of known MPs. Teaching MPs in isolation can also cause problems. A demonstrated skill is usually a smooth movement. Therefore continuous primitives are blended together in a demonstration and the

actual sensor values at the beginning and end of a primitive can differ from the expected ones. This blending of primitives is a general problem for segmentation algorithms and using primitives learned in isolation can even aggravate the problem.

As soon as a sequence is segmented into MPs, these movements can be used to reproduce the demonstrated skill. The drawback here is, that only the exact sequence can be reproduced. More complex skills require a non-deterministic order of the sequence, e.g., repeating a movement until a certain sensor event happens or choosing a succeeding movement dependent on the state of the environment. The representation of the skill therefore has to incorporate such task knowledge into its model.

The traditional way of modeling skills with a two-level hierarchy is by interpreting the switching behavior as discrete events in a continuous system as illustrated in Figure 1.2 (left) [Pavlovic et al., 2000, Peters, 2005]. Here, an event is often represented as a transition in a directed graph. In [Kulić et al., 2012], 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.

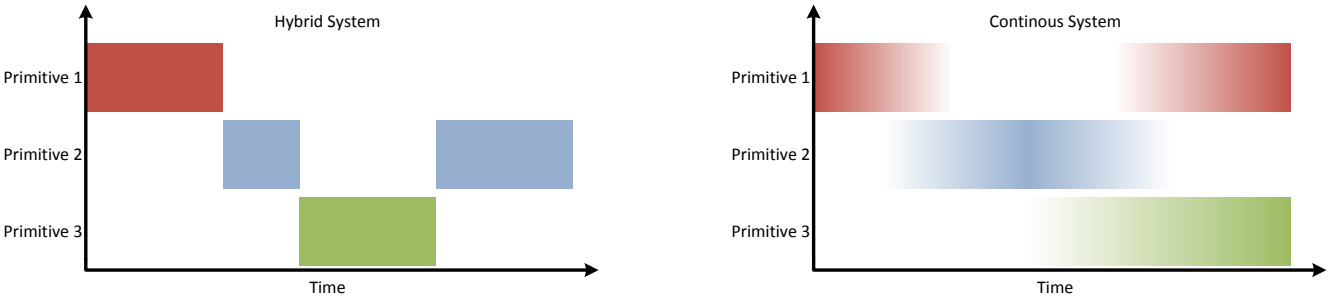


Figure 1.2.: Different views of a sequence of primitives. The left figure shows a hybrid system, in which switches between primitives are seen as discrete events in a continuous system. The right figure shows a continuous system. Here, primitives can be concurrently (and gradually) active. The behavior of the system is a superposition of all primitives.

Graphs can also represent subgoals or constraints of a task [Ekvall and Kragic, 2006, Nicolescu and Mataric, 2003, Pardowitz et al., 2005]. 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, the authors of [Luksch et al., 2012] 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 [Pastor et al., 2012], primitives 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. In contrast to that, the authors of [Niekum et al., 2013] use a finite state machine as explicit representation of a skill. The switching behavior between the different primitives that are linked to these states is learned using a classifier.

Learning directly from humans by observing their movements can be quite tedious because of the required complex tracking systems and other difficulties such as the correspondence problem [Nehaniv and Dautenhahn, 2002]. Kinesthetic teaching is therefore a widely used alternative in movement learning [Billard et al., 2006, Calinon and Billard, 2007, Calinon et al., 2007, Kober et al., 2010]. In addition, reinforcement learning allows for self-improvement of the skill and/or the underlying primitives [Daniel et al., 2013, Morimoto and Doya, 1998, Pastor et al., 2011]. 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.

2 Learning sequential skills

In the following sections we will focus on our approach and explain it in more detail. Similar to other work, the system is also modeled on two levels of abstraction. The upper level represents the sequence of primitives and can be seen as the set of states the robot has to pass through in order to perform the task. As this representation models the skill itself it is independent of the reproduction and has to be learned from the demonstrations.

On the lower level, sensor values and other features represent the current state of the sequence. As a sequential skill is the ability to guide the robot through a desired sequence of movements, it is necessary to connect this lower level with the upper one in a way that it becomes possible to classify the current state into the overall representation of the skill.

The straightforward way of applying machine learning methods to the sequencing of primitives would be to train one overall classifier with the data sampled from the kinesthetic demonstrations. As we assume that the data is labeled, the training could be performed with state of the art methods. A reproduction then could be achieved by performing the classification based on the current features of the system at every point in time. Although this solution sounds simple and might work for very basic skills, it has some major drawbacks that come up with an increasing complexity of a skill. Usually the feature set as well as the number of required primitives grow with the complexity. As a result, the danger of misclassifications also increases as the classifier has to choose between more classes. In addition, the classification will take more time which might violate the real time constraints of the system.

Our approach therefore tries to reduce the number of involved classes and feature dimensions. In our case, the upper layer of the two-level hierarchy is represented by a sequence graph in which nodes correspond to movement primitives. The state of the sequence (or lower layer) consists of the current node in the graph and a set of features which are based on the raw sensor values. Compared to other approaches, the feature set is not fixed but also depends on the current node in the graph. We show how this reduction of the feature set can help to improve the overall reproduction result.

Instead of one single overall classifier, a different classifier is used for each node in the graph. During reproduction, the classifier belonging to the current node in the graph decides whether to keep on executing the current primitive or to switch to one of the possible successors in the graph. We evaluate two different types of classifiers: A generative and a discriminative classifier. As generative classifier, we evaluate Hidden Markov Models (HMMs) and as discriminative classifier Support Vector Machine (SVMs). Using a sequence graph in conjunction with a set of classifiers splits the overall classification problem into many smaller problems. These problems are easier

to solve and we will show that the reduction of the complexity makes the learning of a skill manageable.

2.1 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 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. The feature set is denoted as $\boldsymbol{x} \in \mathbb{R}^n$. The features are not global but assigned as output vectors to primitives, leading to one output vector $\boldsymbol{x}_i$ per primitive p_i.

Figure 2.1 shows the overall flow of our approach based on a simple example with only 3 different primitives. The primitives are illustrated by using different colors. They are chosen arbitrarily and have no further meaning, but show the essential characteristics of our approach.

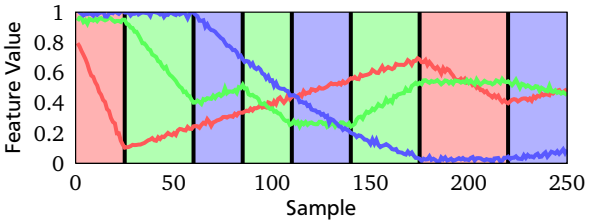
We start with (labeled) sampled data of at least one kinesthetic demonstration (Figure 2.1a). Based on the observed sequential ordering of the primitives, the skill then gets represented by a sequence graph in which every node is linked to a primitive (Figure 2.1b). The presentation will be explained in detail in the following section, where we also present two different types of sequence graphs, both showing different ways of incorporating the ordering into the representation.

After generating the sequence graph, one classifier is trained for each node in the graph (Figure 2.1c). When reproducing the skill, always one node in the graph is then 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.

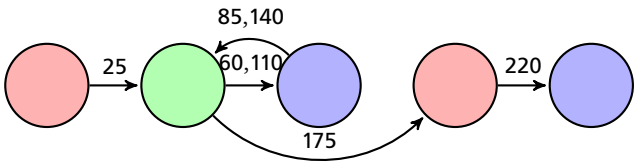
In Section 2.2 the sequence graph representation is formalized. Here, also two different types of graphs are presented. The learning algorithms for these graphs are then explained in detail in Section 2.3. In Section 2.4 it is shown how the switching behavior between the connected primitives is learned.

2.2 Representing Skills with a Sequence Graph

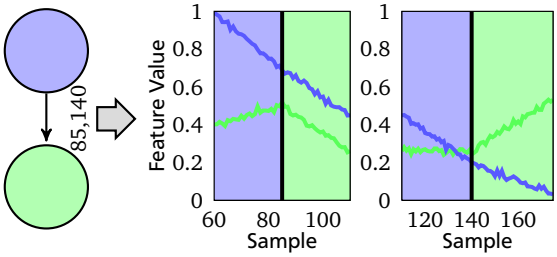
A sequence graph is a directed graph in which each node n_i is linked to a movement primitive. This mapping is not injective which means a primitive can be linked to more than one node. During reproduction, a primitive gets executed if a linked node is considered active. Transitions in the graph lead to primitives that can be executed next, whereby the switching behavior is learned by a classifier. A transition $t_{k,l}$ is connecting the node n_k with n_l. Each transition is linked



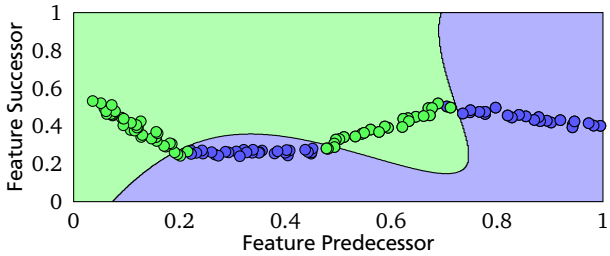
(a) Sampled (labeled) data of one kinesthetic demonstration. The background color indicates the activated primitive, while the plot colors show which feature belongs to which primitive. In this simplified example each primitive has only one associated feature.



(b) Based on the sequential order of the demonstrations, the skill is represented with a sequence graph. The transitions are linked with the corresponding data points of the transitions.



(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.

Figure 2.1.: 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.

with the corresponding data points at which it was observed during the demonstration (black vertical lines in Figure 2.1a). As the same transition can be observed multiple times, multiple data points are possible.

Having m nodes in the graph, we use a $m \times m$ transition matrix $\boldsymbol{T}$ to describe one sequence graph. Note that it is possible to continue with the execution of the current primitive at each time step. As a result, 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 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.

2.2.1 Local Sequence Graph

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 n nodes 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 Figure 2.2.

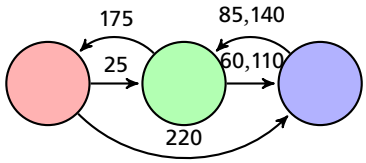


Figure 2.2.: Local sequence graph for the toy example. Each primitive appears only once in the graph. Thus, there are less nodes and more involved classes for each transition.

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.

2.2.2 Global Sequence Graph

The global sequence graph tries to overcome the strong requirements on the feature set of the local sequence graph 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 (Figure 2.3). Here, two nodes are linked to the red primitive because the sequence was considered to be in two different states when they were executed.

When comparing the primitive ordering (Figure 2.1a) with the sequence graph, it is notable that the repeated appearance of the green-blue transition is represented by only two nodes in

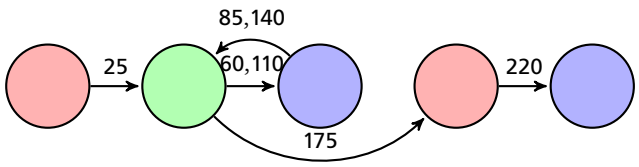


Figure 2.3.: Global sequence graph for the toy example. Compared to the local sequence graph, the global graph has more nodes, but less outgoing transitions per node.

the graph which are connected by a cyclic path. 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.

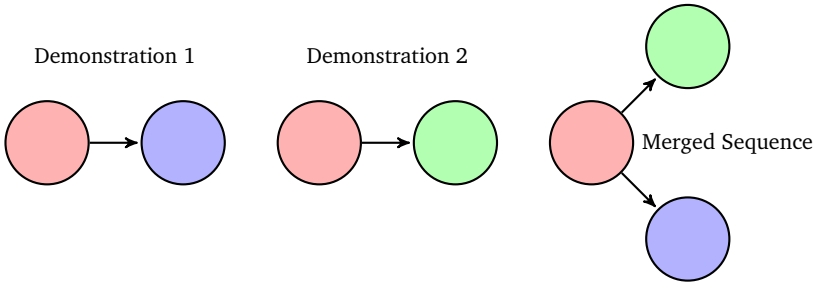


Figure 2.4.: The global sequence graph considers the sequence order when merging multiple demonstrations into one representation. The orders are compared with each other and branches are introduced if nodes differ.

In addition to interpreting repetitions of primitives as one state of a sequence, the global sequence graph is able to consider different sequence orders of multiple demonstrations. The orders are compared with each other and branches are introduced into the graph structure if nodes differ. An illustration of this merging step is shown in Figure 2.4. As a primitive can be linked to more than one node in a global sequence graph, the graph has in general more nodes than the local sequence graph for the same kill. The resulting representation is flatter and hence the nodes have less outgoing transitions. The classification therefore becomes easier as less classes are involved in the decisions.

2.3 Sequence Graph Learning

In this section we discuss how the presented representations are learned as well as how the classifiers are trained and which data is taken for the training.

2.3.1 Local Sequence Graph

The local sequence graph is created by simply looking at the primitive pairs. The starting point is the sequential order of the primitives from the demonstrations. As the number of nodes is equal to the number of primitives, the graph is initialized with n nodes.

Next, the algorithm steps through the order and for each switch of a primitive a corresponding transition is added to the graph, leading from the node of the preceding primitive to the node of the succeeding primitive. If a transition with the same predecessor and successor already exists in the graph, only the corresponding data point is added to the existing transition.

Multiple demonstrations are processed one after the other and the starting point for each demonstration is the generated graph of the previous demonstration. The algorithm is summarized in Algorithm 1. Here, $\boldsymbol{T}_{\text{all}}$ is the set of all demonstrations and $\boldsymbol{G}$ is the generated sequence graph which is the output of the algorithm.

Algorithm 1 Local sequence graph generation

Require: $\boldsymbol{T}_{\text{all}}$
  $\boldsymbol{G}$ = initializeGraph(T_{all}); {Initialize with n nodes}
  for all $T \in \boldsymbol{T}_{\text{all}}$ **do**
    $\boldsymbol{S}$ = getSequenceOrders($\boldsymbol{T}$);
    $\boldsymbol{E}$ = getTransitions($\boldsymbol{S}$);
    for all $e \in \boldsymbol{E}$ **do**
      if hasTransition($\boldsymbol{G}, e.predecessor, e.successor$) **then**
        e_{existing} = getTransition($\boldsymbol{G}, e.predecessor, e.successor$);
        addTransitionPoint($e_{\text{existing}}, e.transitionPoint$);
      else
        addTransition($\boldsymbol{G}, e.predecessor, e.successor, e.transitionPoint$);
      end if
    end for
  end for
  return $\boldsymbol{G}$;

2.3.2 Global Sequence Graph

For creating a global sequence graph three major steps have to be performed:

1. Create one acyclic graph $\boldsymbol{T}_i$ for each demonstration.
2. Replace repetitions of primitives 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 primitive 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 2. The algorithm starts with the sequential order S with n elements and searches for a repetition of $l = \lfloor n/2 \rfloor$ 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 $\{n_{l+1}, \ldots, n_{2l+1}\}$.

Algorithm 2 Graph folding

```
Require: T
  S = getSequenceOrders(T);
  repetition = findRepetition(S);
  while repetition.found do
    M = ∅;
    m = repetition.end − repetition.start + 1;
    for i = repetition.start to repetition.end do
      mergeNodes(S(i + m), S(i));
      M = M ∪ S(i);
    end for
    tail = findTail(S, repetition.end + 1);
    repetition = findRepetition(S);
    if !repetition.found and tail.found then
      for i = tail.start to tail.end do
        mergeNodes(S(i + m), S(i));
        M = M ∪ S(i);
      end for
    end if
    removeNodes(M);
    S = S \ M;
  end while
```

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 transition points 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.

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 Figure 2.1b. Here, the green-blue repetitions end incompletely with the green primitive.

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 Algorithm 3 merges two graphs and thus gets called $n-1$ times for n 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.

Algorithm 3 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**
        $c_\mathrm{max} = c$;
        $\boldsymbol{s}_\mathrm{A,max} = \boldsymbol{s}_\mathrm{A}$;
      end if
    end for
    $nodes = 1$;
    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
        $\mathrm{addNode}(\boldsymbol{T}_\mathrm{A}, \boldsymbol{s}_\mathrm{B}(i))$;
      end if
      $nodes = nodes + 1$;
    end for
  end for

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 introduced 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.

2.3.3 Characteristics of the Representations

Note that if a skill always requires the same number of repetitions of a sequence, both presented sequence graphs will introduce a cycle in the representation. The system is then only able to reproduce the movement properly if the classifier finds 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 repetitions of primitives.

Although the global sequence graph has advantages over the local sequence graph due to the more detailed representation, it also introduces some drawbacks. Figure 2.5 depicts an exemplary sequence order in which two possible repetitions of (sub-)sequences exist.

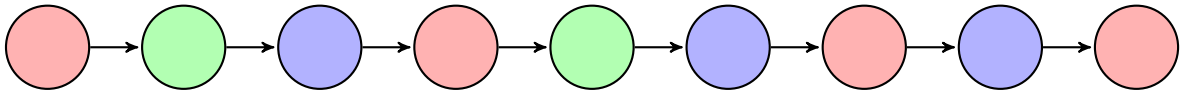
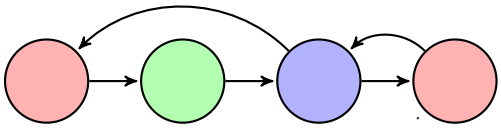


Figure 2.5.: Exemplary sequence order in which multiple repetitions of (sub-)sequences occur. Without additional knowledge about the task it is not clear how these cycles should be represented in the graph.

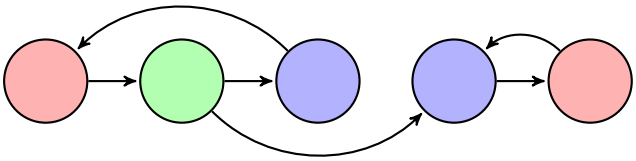
In the beginning, the repetition *red-green-blue* can be found, followed by the repetition *blue-red* in the end, both appearing two times. Without additional knowledge about the skill there are many possibilities to build the global sequence graph. For example, it is not clear if the second appearance of the *blue* primitive is belonging to the first repetition, the second one or both. Depending on such an assumption about the skill the resulting graph may differ. Figure 2.6 depicts four possible sequence graphs for the exemplary sequence order.

As an assumption has to be made here, we chose to search from left to right and require a full repetition of a (sub-)sequence in order to create a cycle in the graph as illustrated in Figure 2.6c. In the following section it is explained how the search is performed and in which cases a cycle will be introduced.

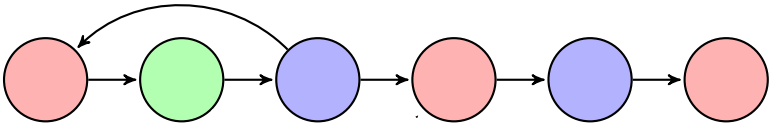
One characteristic of the local sequence graph is that it is not able to represent partial orders. For example when baking a cake, it is irrelevant whether the eggs or the flour are put into the bowl first. If both possibilities are demonstrated, the local sequence graph will introduce a bilateral transition between both primitives. As a result, features are needed which prevent the system from taking this transition more than one time. The global sequence graph instead would introduce two different branches and after deciding for the eggs or flour first it would choose the other one as successor.



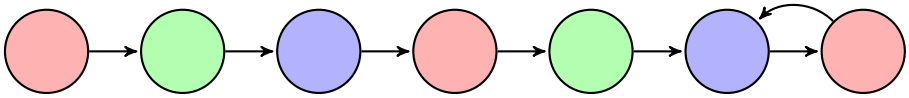
(a) Two cycles are introduced. It is possible to get back to the first cycle when being in the second one.



(b) Two cycles are introduced. It is not possible to get back to the first cycle once the transition leaving the cycle was taken.



(c) Only one cycle is introduced. Here, the blue primitive belonging to both repetitions is assigned solely to the first repetition. As a result, the remaining order *red-blue-red* is not considered to be a repetition anymore and thus no second cycle is introduced.



(d) Only one cycle is introduced. Here, the blue primitive belonging to both repetitions is assigned solely to the second repetition. As a result, the starting order *red-green-blue-red-green* is not considered to be a repetition anymore and thus no second cycle is introduced.

Figure 2.6.: Possible sequence graphs for the exemplary sequence order of Figure 2.5. The graphs differ significantly depending on their metric of finding the repetitions in the sequence order.

2.4 Learning the Switching Behavior

After creating the graph representation, the next step is to learn the switching behavior between the primitives. In our case, this is treated as classification problem. We therefore present in this section how the classifiers are trained. 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 $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)^T$, 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 primitives are prevented. A reduction of the output vector

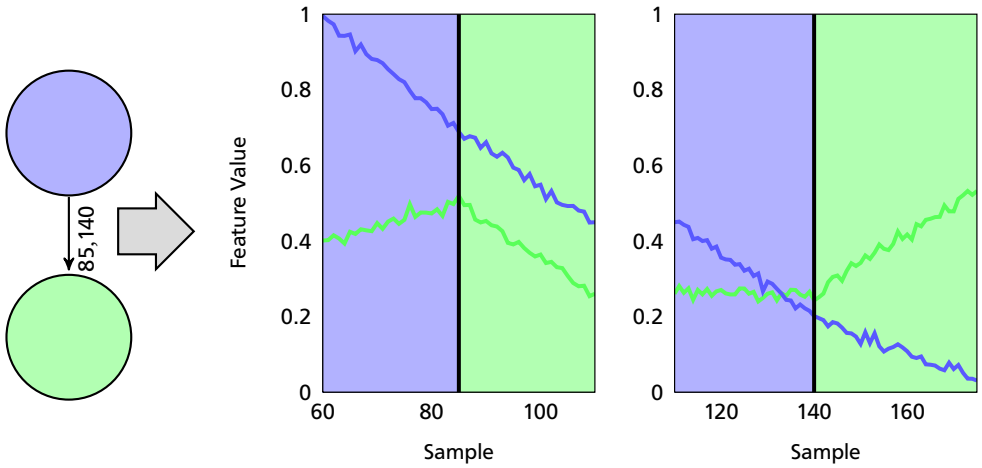


Figure 2.7.: For each observed transition point the data between the start of the preceding primitive and the end of the succeeding primitive is chosen as training data. Only the features of the nodes that are linked to each other in the graph are considered for the classification problem. In this exemplary transition of the toy example, the red primitive is not connected to the blue primitive and hence its feature is not used.

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 Figure 2.7). After the demonstrations, each transition in the acyclic graph is linked to one transition point (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 transition point 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.

Next, a rough overview on the two used classifiers is given, starting with HMMs and then introducing SVMs. As these 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 [Bishop, 2006]

2.4.1 Hidden Markov Models

HMMs model a sequence of features as a set of m hidden states $Z = \{z_1, z_2, \ldots, z_m\}$ with observable outputs (emissions) that depend on probability distributions. The sequence is taken into account by introducing transition probabilities between the states. Thus, the current state of the system depends on the features as well as on the last state. We perform the classification by introducing one state for each class. A state is modeled by a Gaussian Mixture Model (GMM)

and the number of Gaussians is determined by 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 state.

The transitions can be represented by a $m \times m$ matrix $\boldsymbol{T}$ with element $T(i,j)$ being the probability of switching to state j when being in state i. As the demonstration data is labeled, the probabilities can be estimated directly from the observations $\{l_1, l_2, \ldots, l_n\}$, $l_i \in Z$:

$$T(i,j) = \frac{\sum_{k=1}^{n} \mathrm{b}(l_{k-1}, z_i)\, \mathrm{b}(l_k, z_j)}{\sum_{k=0}^{n-1} \mathrm{b}(l_k, z_i)} \tag{2.1}$$

Here, b is a Boolean helper function which simply checks two labels for equality:

$$\mathrm{b}(l_i, l_j) = \begin{cases} 1 & \text{if } l_i = l_j \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

The numerator in (2.1) is equal to the number of transitions to state z_j when being in state z_i. The denominator is equal to the number of times the state z_i has been observed and can be seen as scaling factor ensuring that each row of the matrix sums up to one, thus representing a probability distribution.

Each GMM is learned separately with the expectation-maximization algorithm. For the classification, the Viterbi algorithm is used to get the most probable state path for the input sequence and the class with the last state is chosen as classification result. The behavior of probabilistic

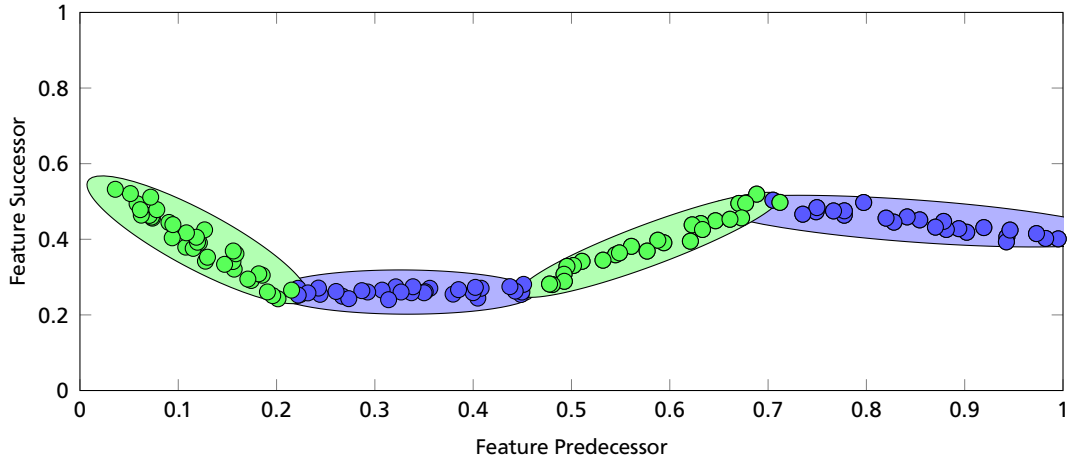


Figure 2.8.: Classification result of a Hidden Markov Model. The HMM approximates the training data (colored dots) with Gaussian distributions (ellipsoids). Despite the correct modeling of the data, the classification result for the white area often is not conclusive.

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. Figure 2.8 shows how the HMM 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.

2.4.2 Support Vector Machines

We chose SVMs as discriminative classifiers. 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 [Chang and Lin, 2011] as implementation for the SVM and we use radial basis functions as kernels:

$$K(\boldsymbol{x}_n, \boldsymbol{x}_m) = \exp(-\gamma \|\boldsymbol{x}_n - \boldsymbol{x}_m\|^2), \gamma > 0 \tag{2.3}$$

The detailed SVM equations can be found in Appendix A.3. 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.

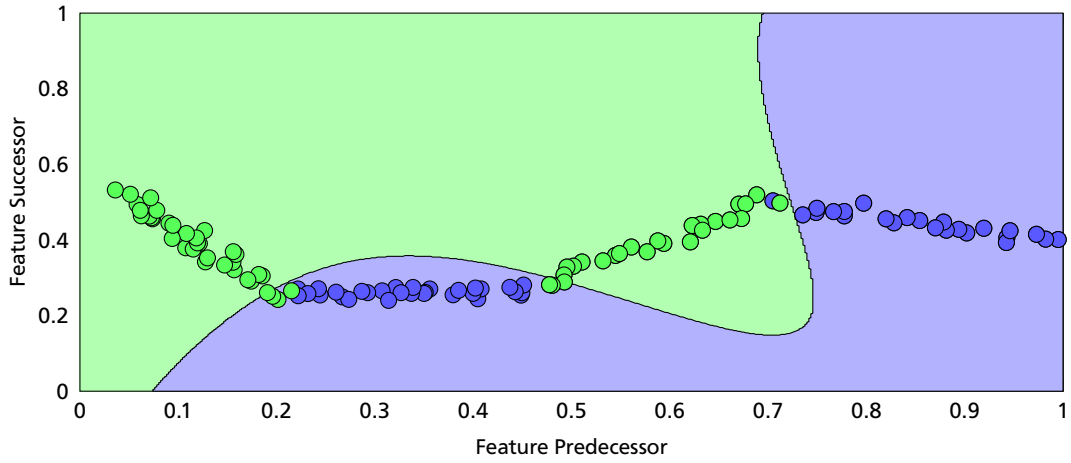


Figure 2.9.: Classification result of a SVM trained with data of the toy example. In contrast to a HMM, the SVM is finds a clear border between both classes.

The classification result for the toy example is depicted in Figure 2.9. The SVM is able to separate the feature space into two different areas as indicated by the different background colors. We will show in the succeeding section that this clear distinction indeed leads to better results when reproducing the skill, despite both classifiers having similar misclassification rates for the training data.

3 Experiments

In this section we present the results of our work. We evaluated our approach both in simulation and with a real Barrett WAM robot. As a scenario we chose to unscrew a light bulb. In Section 3.1 we outline the details of the experiments. The results are then presented in Section 3.2 and discussed in Section 4.1.

3.1 Setup

The Barrett WAM robot has seven DoF and its joints are torque controlled. The controller is model based. The attached hand has four DoF. Each finger has two joints, whereby the joints of each finger are kinematically coupled to each other. Therefore each finger has one DoF. The remaining DoF can be used to spread two fingers of the hand. Strain gauges in the fingers can be used to measure contacts between the hand and the environment. Additionally, one six-dimensional force torque sensor is at the wrist of the robot's hand.

For the representation of the skill we chose seven different movement primitives as depicted in Figure 3.1. 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. The detailed task flow is illustrated in Figure 3.2. 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.

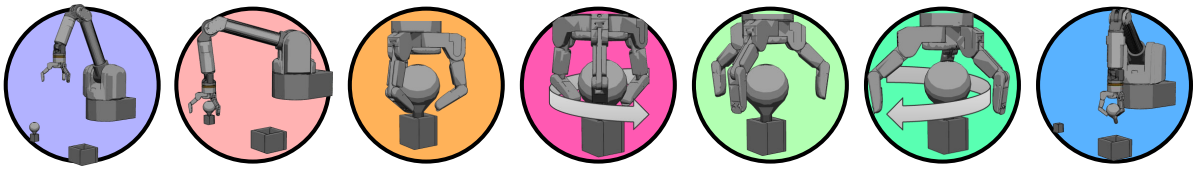


Figure 3.1.: Seven primitives were used to unscrew the light bulb. The movements performed by the primitives are going to the initial position (●), going to the light bulb (●), closing the fingers (●), rotating the hand counterclockwise (●), opening the fingers (●), rotating the hand clockwise (●), and going to the bin (●).

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 $\boldsymbol{x}_{\text{goal}}$:

$$\boldsymbol{\Delta} = \boldsymbol{x}_{\text{goal}} - \boldsymbol{x} \tag{3.1}$$

$$g = 1 - e^{-0.5\boldsymbol{\Delta}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\Delta}}. \tag{3.2}$$

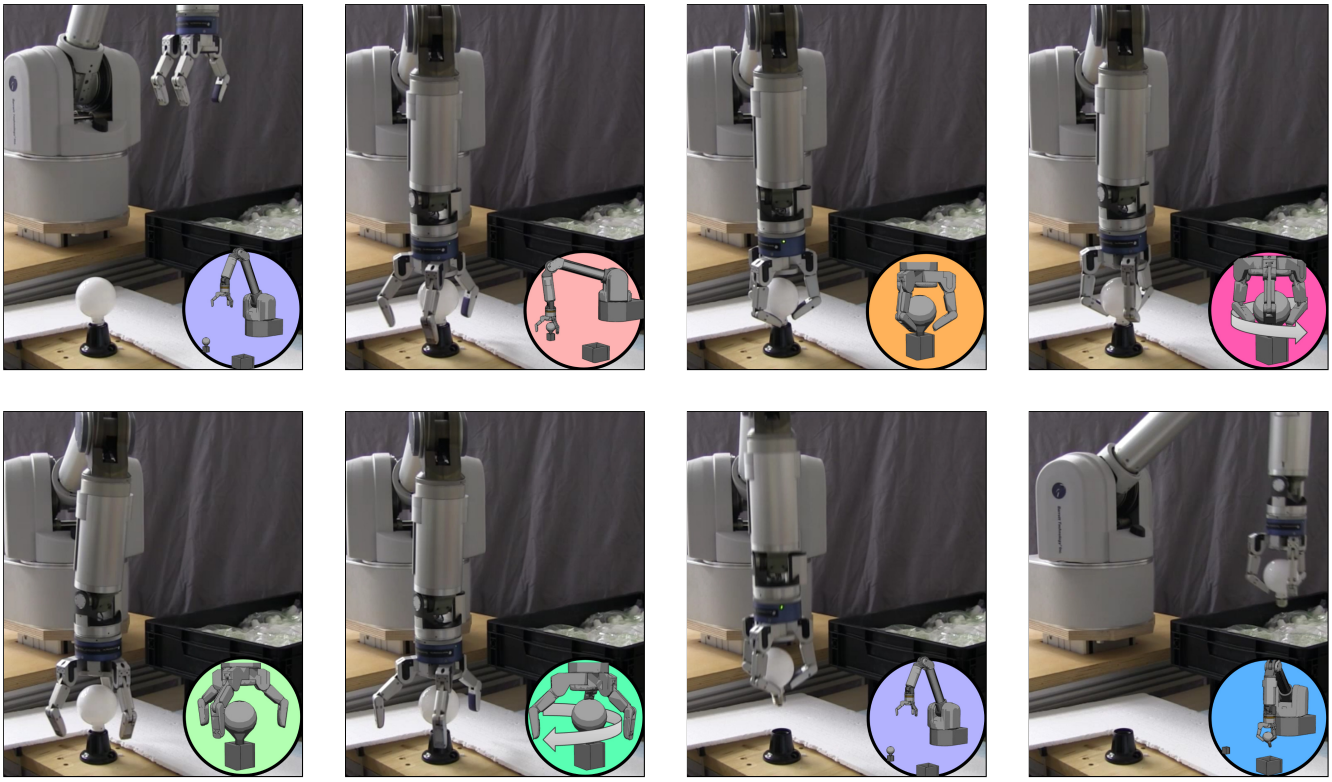


Figure 3.2.: Illustration of a successful unscrewing sequence. The robot starts in an initial position (●) and first moves towards the bulb (●). Then it repeats the unscrewing movement (●, ●, ●, ●) until the bulb loosens (●) and subsequently, the bulb is put into a bin (●) and the robot returns to its initial position (●).

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 (3.2) is dependent 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 v 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 using the mean $\bar{v}$ and standard deviation σ_v with the equation

$$\tilde{v} = \frac{v - \bar{v}}{\sigma_v} + 0.5 \tag{3.3}$$

and then 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 primitives.

3.1.1 Automatic Demonstration

We first validated our approach in simulation before testing the movements with the real Barrett WAM robot. Figure 3.3 shows the simulation environment that was used for the experiments. The simulation framework is a commercial software and uses the Vortex physics engine. The engine computes the contact forces between dynamic objects in the environment and hence it was possible to simulate the interaction between the robot and the light bulb. To ensure comparability between the results of simulation and real experiments we recreated the real setup within the simulation framework. The necessary unscrewing repetitions were varied by changing the position of the light bulb in the holder for each demonstration. Recreating the scenario within the simulation allowed us to use the same underlying primitives in both simulation and for the real experiments. The simulation was therefore also used as an easy rapid-prototyping method to ensure that the primitives were defined properly and that they were sufficient for the execution of the skill.

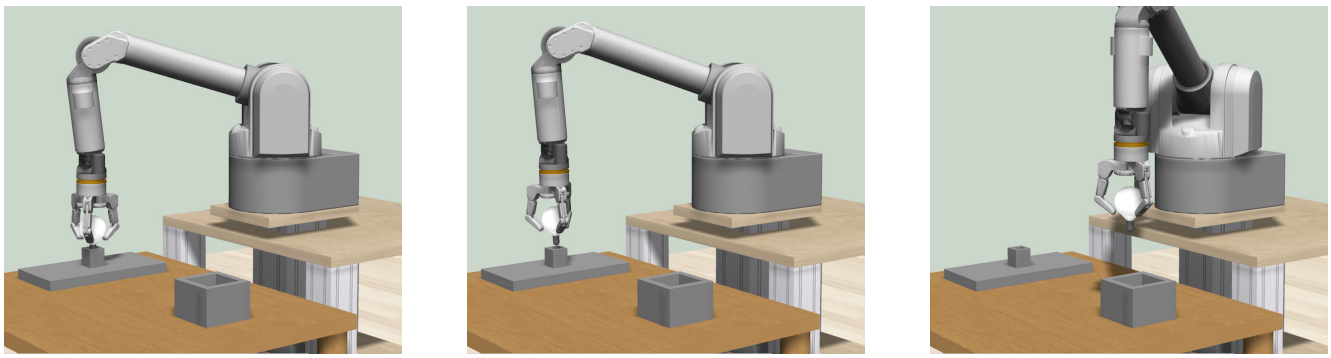


Figure 3.3.: The real setup was recreated within the simulation framework. The teaching was done with automatic demonstrations in which primitives are executed in a predefined order.

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.

3.1.2 Kinesthetic Demonstration

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 (see Figure 3.4).

Figure 3.4.: Kinesthetic demonstration with a Barrett WAM. The skill of unscrewing a light bulb was taught to the robot by guiding its arm through the movement. Transitions between primitives were indicated by pressing a key. Also, the closing and opening of the hand was activated by pressing a key in order not to influence the force torque sensor at the wrist of the robot.

Although only the transitions were indicated by pressing keys, the labeling could be performed automatically after each demonstration. The reason is the definition of the skill, which is deterministic in our case as both start and end of the sequence are the same for every trial. In the beginning, the robot has to go down to the bulb and in the end it has to put the loose bulb into the bin. In between the primitives responsible for unscrewing the bulb are executed arbitrary times in a predefined order. Thus, a conclusion about the number of repetitions can be drawn by dividing the difference of total and fixed primitives by the number of repetitive primitives. Algorithm 4 depicts how the labeling was performed in detail. Note that this is not an automatic data labeling algorithm in a sense that it labels data by choosing the best matching primitive from a library. Instead, it is a fixed and manually defined algorithm which reduces the effort for the teacher only for this skill.

Algorithm 4 Labeling of a kinesthetic demonstration

Require: $\boldsymbol{X}, \boldsymbol{t}$ {Data and vector with transition points}
  $primitives = \text{length}(\boldsymbol{t}) + 2$;
  $datapoints = \text{length}(\boldsymbol{X})$;
  $repetitions = (primitives - 7)\%4$;
  $\boldsymbol{labels} = \text{vector}(datapoints)$;
  {Label start of sequence}
  $\boldsymbol{labels}(1 : t(1) - 1) = \textit{Go to light bulb}$;
  {Label arbitrary repetitions}
  for $idx = 0$ **to** $repetitions - 1$ **do**
    $\boldsymbol{labels}(t(4 * idx + 1) : t(4 * idx + 2) - 1) = \textit{Close hand}$;
    $\boldsymbol{labels}(t(4 * idx + 2) : t(4 * idx + 3) - 1) = \textit{Rotate counterclockwise}$;
    $\boldsymbol{labels}(t(4 * idx + 3) : t(4 * idx + 4) - 1) = \textit{Open hand}$;
    $\boldsymbol{labels}(t(4 * idx + 4) : t(4 * idx + 5) - 1) = \textit{Rotate clockwise}$;
  end for
  {Label end of sequence}
  $\boldsymbol{labels}(t(end - 5) : t(end - 4) - 1) = \textit{Grasp}$;
  $\boldsymbol{labels}(t(end - 4) : t(end - 3) - 1) = \textit{Rotate counterclockwise}$;
  $\boldsymbol{labels}(t(end - 3) : t(end - 2) - 1) = \textit{Go to initial position}$;
  $\boldsymbol{labels}(t(end - 2) : t(end - 1) - 1) = \textit{Go to bin}$;
  $\boldsymbol{labels}(t(end - 1) : t(end) - 1) = \textit{Open hand}$;
  $\boldsymbol{labels}(t(end) : end) = \textit{Go to initial position}$;
  return $\boldsymbol{labels}$;

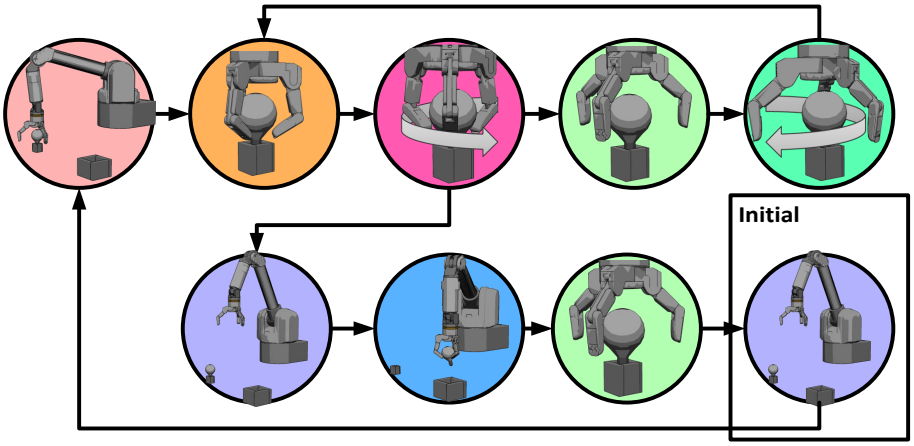
3.2 Results

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 Figure 3.5.

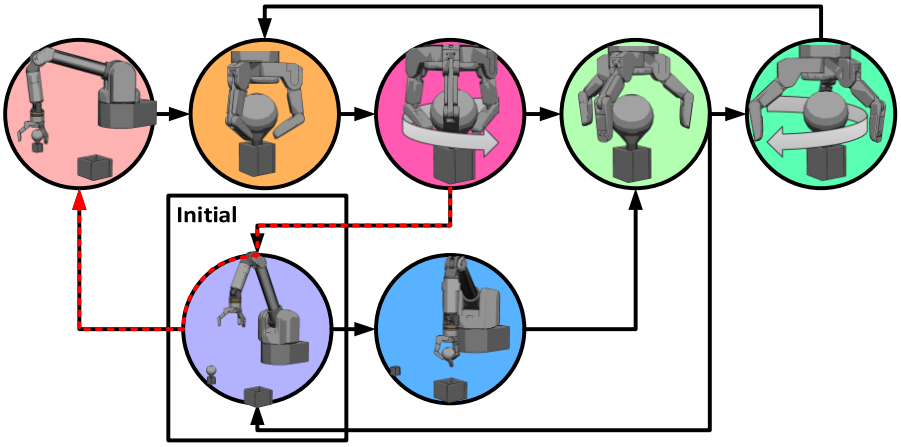
Figure 3.6 shows the classification recall for the simulation. As reference we trained a SVM without creating a graph representation using the complete labeled data, hence no dimensionality reduction was used.

Figure 3.7 shows the results for the reproduction of the movement. The table outlines the percentages 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 all succeeding transitions could be tested. Dashed entries 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 3 kinesthetic demonstrations, hence 7 different outcomes are possible. Figure 3.8 and



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



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

Figure 3.5.: 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 presentation which were not demonstrated. An example is the sequence marked as red which leads to a misbehavior of the robot if executed.

Figure 3.9 show the classification recall and reproduction results for the experiments on the real robot. Here, the same reference classifier was used and again all combinations of SVMs, HMMs and both sequence graph types were evaluated. First tests showed that the reference classifier did not provide useful results. In order to not harm the robot we therefore skipped it for the evaluation. The results of the experiments are discussed in Section 4.1.

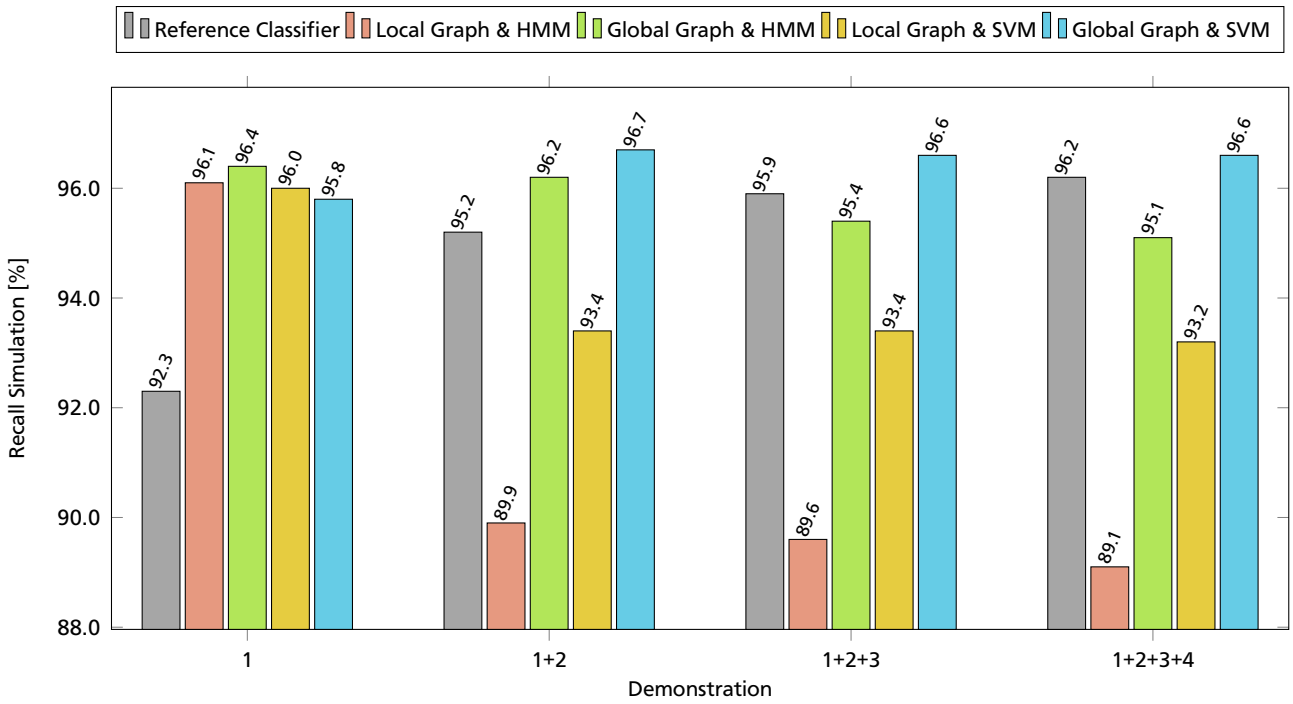


Figure 3.6.: Classification recall in percent based on 4 different demonstrations, using SVMs, HMMs together with the global respective local sequence graph. The reference classifier is a SVM which was created without using a sequence graph.

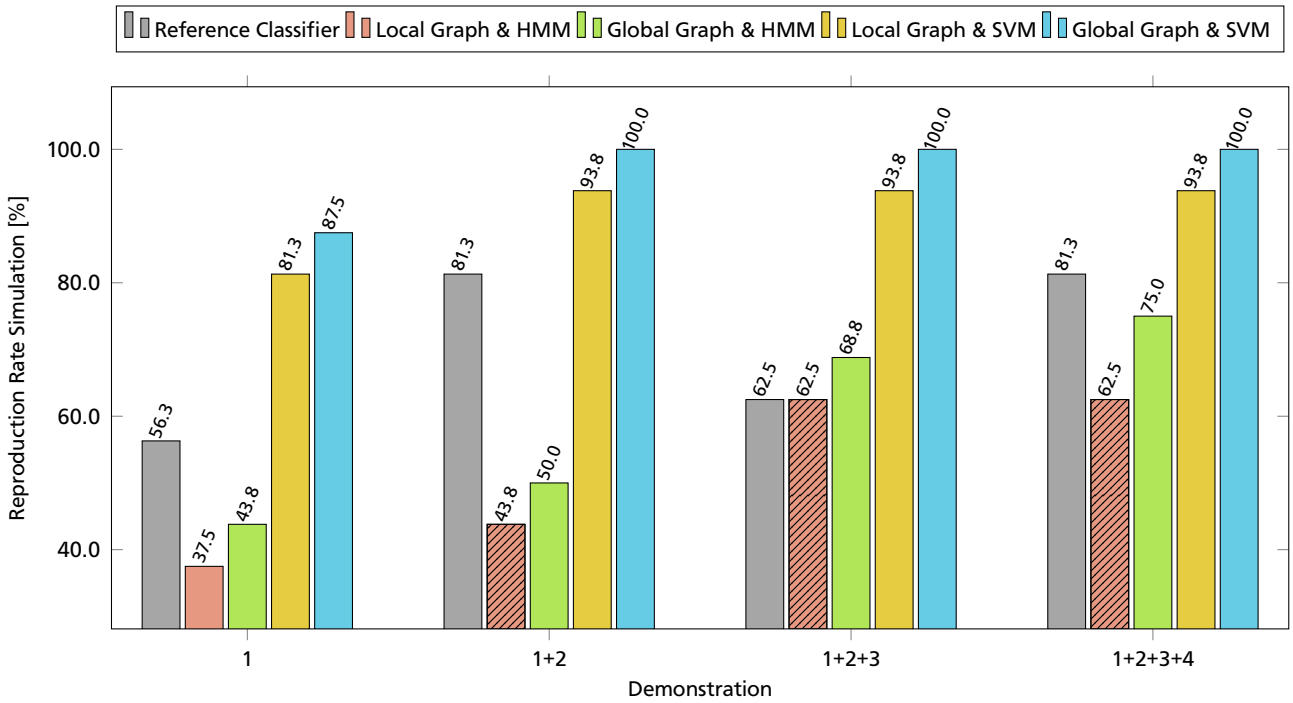


Figure 3.7.: Reproduction results in percent of different demonstrations for SVMs and HMMs both using the global and local sequence graph. Shown are the successfully performed primitive switches compared to the overall switches. Dashed entries indicate dangerous behaviors. The reference classifier is a SVM which was created without using a sequence graph.

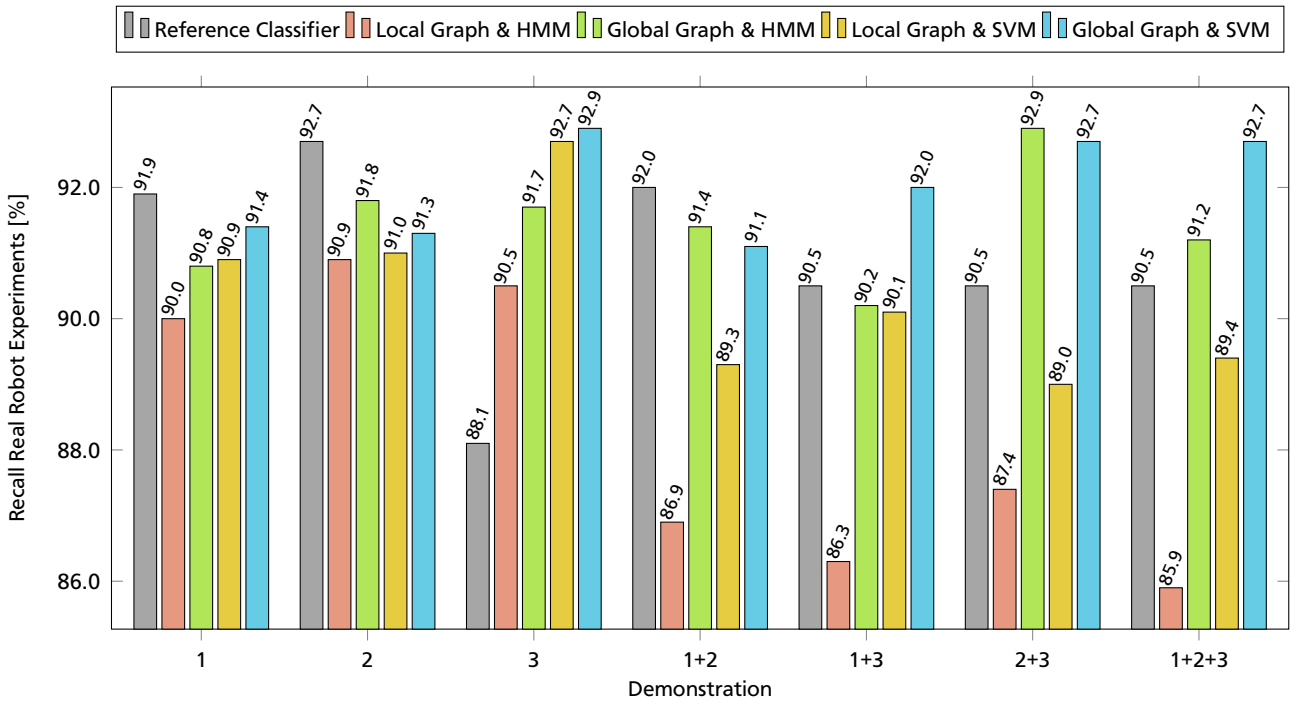


Figure 3.8.: Classification recall in percent based on three kinesthetic demonstrations for SVMs and HMMs both using the global and local sequence graph.

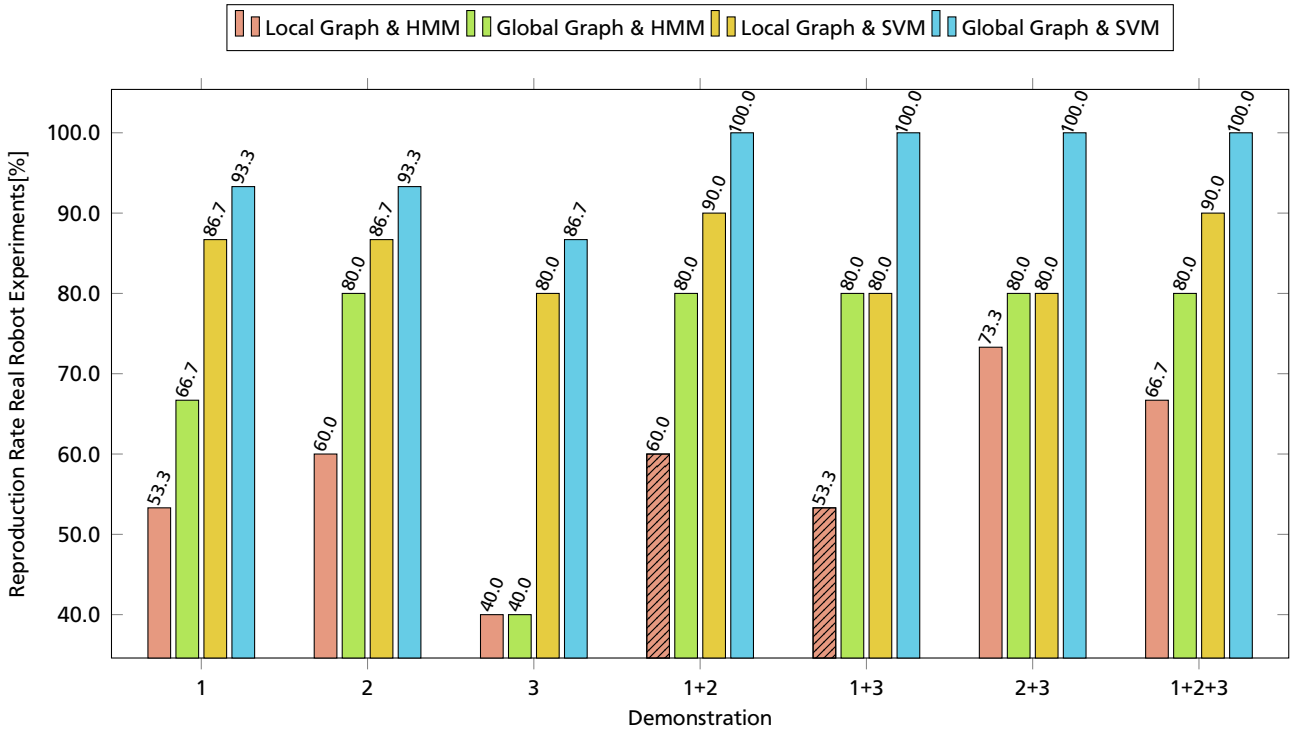


Figure 3.9.: Reproduction results of different demonstrations for SVMs and HMMs both using the global and local sequence graph. Shown are the successfully performed primitive switches compared to the overall switches. For dashed entries the reproduction failed.

4 Discussion

In this chapter we will discuss the results of the simulation and the experiments with the Barrett WAM. We will also propose some possible enhancements of our approach. The enhancements might lead to even better reproduction results or make the approach applicable in more scenarios. They were not implemented because they are beyond the scope of this thesis.

4.1 Experiments

The presented results in Figures 3.6-3.9 show that our system is able to reproduce the demonstrated skill properly if the appropriate combination of sequence graph and classifier is chosen. The results emphasize the advantages of the global over the local sequence graph as well as of the SVMs over the HMMs. In the following, we will discuss the results in detail.

4.1.1 Sequence Graphs

Both created sequence graphs for the light bulb task are shown in Figure 3.5. The local graph differs in two important aspects from the global graph. First, all nodes representing the same primitive are merged into a single node. Second, this merging introduces paths in the graph that have not been demonstrated, such as the one marked as red in the figure.

Executing the movement shown by the red path leads to the following sequence: The robot unscrews the bulb, starts returning to its initial position and subsequently goes back to the position above the light bulb holder instead of going to the bin. Normally the movement of going down to the bulb is executed at the beginning of the task. At this point in the sequence the bulb is in the holder. Therefore the robot additionally opens its hand when going down to the holder, so that the bulb can be grasped as soon as the target position is reached. When executing the movements shown by the red path, the bulb is already in the robot's hand, but it is still trying to go down to the holder and additionally opens its hand. As a result, the bulb slips out of the fingers and falls on the ground.

Note that both transitions of the path are correct in a sense that they present valid transitions. A wrong behavior only occurs if both transitions are chosen consecutively. When going to the initial position, the classifier uses the goal distance of this primitive and additionally the two goals distances of the possible successors in the graph as features. These features are not meaningful enough to allow a correct classification and thus sometimes an incorrect transition was taken during reproduction. An additional feature such as the state of the hand (closed or opened) would enable a correct classification. This confirms our assumption, that a more enhanced feature

set is necessary when using the local sequence graph. The assumption is based on the more compact description of the skill, which leads to more classes being involved in the classification problem.

The problematic primitive is split up into two nodes in the global sequence graph. Both nodes only have one possible successor, which simplifies the classification problem. In both cases, the classifier only has to decide when to switch between the primitives instead of choosing between multiple alternatives. As this is possible with the feature set of our setup, the problem is not occurring here.

The global representation in general performs only better if the skill is modeled properly as in our case. The algorithm creating the graph inherently makes assumptions about the skill, such as preferring longer cycles over shorter ones. If such an assumption is not matching the requirements of the skill, false transitions can be introduced by the algorithm. The system then can be biased towards wrong decisions, resulting in a worse performance than by using the more general local representation.

4.1.2 Classification

It is apparent from the presented results, that HMMs perform worse than SVMs. However, the system still manages 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 Figure 2.8). Due to the overfitting HMMs 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 HMM training and recommend methods which are trying to find large margins (e.g., Sha and Saul [2007]).

SVMs based on the global sequence graph perform best. 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. Only when using a HMM together with the local sequence the robot sometimes had to be shut off because the loose bulb was not detected by the system. As a force is applied in upward direction during the unscrewing, the bulb is lifted up as soon as it gets loose. If a loose bulb was not detected by the system, the applied force led to an uncontrollable lifting of the arm and the execution of the task had to be stopped in order to avoid damaging the robot.

Using the reference classifier without dimensionality reduction sometimes yields better recall results than using a graph based classifier (see Figure 3.6 and 3.8). Still, the reproduction shows that a real benefit is achieved with a sequence graph. It is notable here, that our approach does

not require each feature to be assigned to a primitive. It is possible to use the whole feature set. In that case, using a sequence graph still would be beneficial due to the disallowing of unseen transitions. Assigning features to primitives can also reduce the accuracy of a system. We discussed a related problem in the previous Section 4.1.1. Here, the information whether the fingers of the robot are open or closed is needed for the classification when using the local sequence graph. The information is globally available but not used by the classifier as it is not assigned to one of the involved primitives. The main conclusion that can be drawn here is that assigning features to primitives can lead to an intuitive dimensionality reduction, but attention has to be paid on the choice of the features as well as on the assignment.

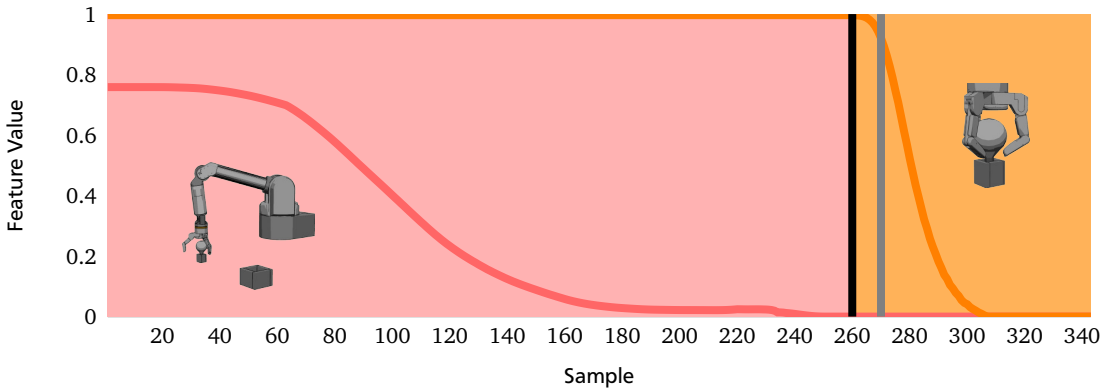
Although the reference classifier provided acceptable results in simulation, the reproduction of the movement on the Barrett WAM failed completely. When starting the movement, the classification outcome was always the same. As a result, the robot only opened its hand and did not execute any other primitive. The fact that the robot was executing this primitive at a state in the sequence and in a position of the robot where it has not been demonstrated shows the drawbacks of a single overall classifier. Despite good classification results, the classifier was not able to generalize from the demonstrations in a way that a reproduction was possible. The reason is that the found hyperplanes separate the training data in wrong dimensions. For example, the hyperplane of the primitive responsible for opening the hand is extended to an area in which it should not be activated. The problem here is, that the feature space grows rapidly with an increasing number of features and hence less training data is available compared to the volume of the feature space. Finding the borders between the classes therefore gets harder, as it is not clear how the empty space should be classified.

The poor reproduction result of the reference classifier shows the benefit of our approach. The reduction of the classes as well as the reduction of the dimensionality simplifies the classification problem and makes the overall problem of learning a sequential skill manageable.

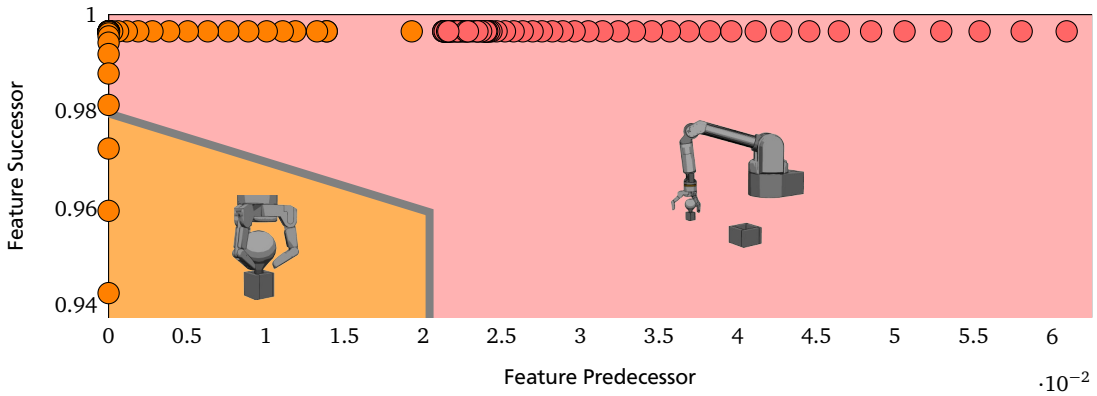
4.1.3 Role of the Teacher

A successful reproduction of a skill also depends on the trainer. Although the robot can be trained intuitively, attention has to be paid to some characteristics of kinesthetic teaching.

Figure 4.1a shows data of a bad demonstration. The snippet shows the samples at the beginning of the task, when the robot's end effector has to go down to the light bulb. Starting from around data point 180 until the switch between the primitives (black vertical line), the trainer tries to adjust the robot's hand precisely above the bulb before initiating the primitive which closes the fingers. The features are not changing significantly in this range and as more data points with similar values exist for the preceding primitive, the state of being above the bulb is assigned to this primitive as depicted in Figure 4.1b. As shown there, the activation of the succeeding primitive gets triggered if its feature value drops below a certain value. The problem is, that the



(a) Sampled data. The black line indicates the transition point between the preceding rose primitive and its orange successor. The gray line shows the classification border found by the classifier.



(b) Classification result for the data in feature space. The data points are represented by the dots and the classification results are indicated by the different background color. The activation of the succeeding orange primitive is triggered if its feature value drops below a certain threshold. As the feature value first starts to change after the triggering, the robot gets stuck because of this chicken-and-egg problem.

Figure 4.1.: A snippet of sampled data from a bad demonstration where the trainer needs a lot of time for adjusting the hand precisely above the light bulb. This leads to the features at the actual transition point being labeled as belonging to the wrong primitive and requires either a manual relabeling or a new demonstration for a successful reproduction.

decrease of the feature value is a consequence of the activation of the primitive. Because of this chicken-and-egg problem, the robot then gets stuck during the reproduction of the movement.

It is notable here that this is not a classification problem. The classifier separates the data almost as good as possible as shown by the small distance between the black and gray line in Figure 4.1a. Instead, it is a teaching problem that needs either a manual relabeling or a new demonstration where the transitions are triggered faster. Another possibility is to use more demonstrations for the training, as the influence of outliers is reduced with an increasing number of successful demonstrations. The results of Section 3.2 also show that it is possible to learn the skill more precisely when using more demonstrations.

Another problem occurs if a transition is triggered at a state of the robot which can not be reached during reproduction. As the goals of the primitives are not learned from the demonstrations but are predefined, the robot is expected to be in a certain state for each switch between primitives. For example, when activating the initial primitive where the end effector has to go down to the light bulb, the robot always reaches its target position above the bulb. When being in the target position, the primitive's goal distance feature is in a certain range, e.g., $g \in [10^{-2}, 10^{-1}]$. The value 0 will not be reached due to inaccurate sensors or friction. If the transition was triggered when the feature value was in the range $0 < g < 10^{-2}$, the classifier is expecting this value for a switch during reproduction. As these values are never reached, the robot can get stuck.

Note that this exemplary transition is only needed one time for the whole skill. To allow for a generalization more than one demonstration can be used. Using multiple demonstrations usually reduces the influence of a bad transition and increases the probability of finding a classification border in the desired range.

4.2 Possible Enhancements

In this section some possible enhancements of the proposed approach are presented, that may increase the accuracy of the system or its generalization capabilities. The presented enhancements are only suggestions and it is unsure if a real improvement of the system could be achieved.

4.2.1 Error Detection

Our approach is not able to detect any errors in the movement. The most likely primitive is chosen at every time step, regardless of how large the actual likeliness is. An error detection requires a comparison between the expected and measured feature values. A potential error occurs if the mismatch between the values is too large. Here, several difficulties arise which will be discussed in the following. A mismatch between expected and measured feature values can have different underlying reasons. Among others these are:

1. An incorrect transition in the sequence graph was taken due to a misclassification.
2. The skill is not modeled properly. The robot is considered to be in a different phase of the sequence than it actually is (e.g., because a transition is in the graph which should not be there).
3. A part of the robot produced a malfunction, e.g., the robot hit an obstacle.
4. The current sensor values differ from the actual values, e.g., due to a measuring error.

The error rate of the two points depends on the feature set, accuracy of the classifiers, and proper modeling of the sequence graph. While these points can be influenced by our system, the other points are hardware errors. For the detection of an error, the classifiers have to provide a confidence in their decision in addition to the classification result itself. The system can then be trained to consider low confidence values as potential errors.

HMMs are probabilistic models and therefore directly supply a confidence of their decision in addition to the classification itself. The standard formulation of SVMs instead is not providing a confidence value. However, it is possible to get probability estimates from a SVM, e.g., by using the output for the training of the parameters of a sigmoid function [Platt, 1999]. The confidence score for a SVM can be seen as distance measurement between the current data point and the border of the separating hyperplane. It is also possible to get probability estimates when using the LIBSVM library, whereby the authors use the method presented in [fan Wu et al., 2003].

Despite having a confidence score for the classification, an error detection can still be a hard problem. The reason is that it is often not clear whether the confidence score is low because of a wrong model (or one of the other presented points) or because the current state of the robot differs too much from the demonstrations. When using simple thresholds for the confidence score, a low threshold might lead to errors being ignored (false negatives). As opposed to this, a large threshold might lead to the detection of too many errors (false positives). As variations are considered as errors in such cases, this would limit the generalization capabilities of the system. Due to the problem of finding a good threshold, a more sophisticated model for detecting errors is necessary in most cases. Instead of interpreting a low confidence score, another possibility is to simply ask the teacher for help. Incorporating this additional knowledge into the decision is called “active learning”.

4.2.2 Learning from Failures

The presented system is not able to learn over time when reproducing the movement. The classifiers are trained with the demonstration data and the parameters are not changed after that. Also, the sequence graph has a fixed structure which is created solely from the demonstrations.

A possible enhancement of our approach therefore could be to use reinforcement learning (RL), which allows for a self-improvement based on a reward function. Reinforcement learning has been widely used in robotics (e.g., Daniel et al. [2013], Kober et al. [2010], Morimoto and Doya [1998], Pastor et al. [2011]). In our case, RL could be used to adapt the graph representations, for example by removing false transitions or adding additional ones. RL could also be used to improve the underlying primitives, e.g., by adapting the goals. However, applying RL affords much effort and goes out of scope of this work.

Learning from failures also requires an adaption of the classifier parameters. SVMs are batch mode classifiers that usually require the complete data set for a retraining. In consequence

of that either the demonstration data has to be available and a complete training has to be performed after each faulty reproduction or a modified training algorithm has to be used. For an online learning algorithm for SVMs, the interested reader is referred to the literature (e.g., Diehl and Cauwenberghs [2003], Kivinen et al. [2004]). HMMs are usually trained using an expectation-maximization algorithm (EM). Here, also online learning algorithms exist, such as [Mongillo and Deneve, 2008]. In our case, the transition matrix needs to be adapted and the GMMs have to be trained again. We also use an EM algorithm for the training of the GMMs. Here, an incremental learning algorithm is proposed by the authors of [Zhang et al., 2010], for example.

4.2.3 Representing Multiple Skills

For this work, only the skill of unscrewing a light bulb was taught. By representing more than one skill with a single sequence graph, it would be possible to teach a robot different skills. The robot then could choose which skill it performs automatically depending on the environment. Another possibility would be to chose the skill manually, e.g., by choosing a path in the graph or by determining the final node in the representation.

In the following, we briefly discuss the suitability of the two presented sequence graphs for modeling multiple skills. When using the local sequence graph and a fixed set of primitives, the number of nodes remains the same with an increasing number of skills. However, the number of transitions grows and the graph becomes more and more fully connected. To ensure taking the right transitions when reproducing a movement, it is very likely that more features are necessary and hence the feature dimension would grow rapidly. We therefore suggest not using the local sequence graph for the representation of multiple skills.

Although it is not implemented, the global approach is inherently able to model multiple skills. The merging algorithm presented in Section 2.3.2 can incorporate two different sequence graphs into one representation. The algorithm merges equal sequences into the same nodes and introduces branches in the graph as soon as two nodes differ. Introducing branches leads to a higher number of nodes compared to the local graph. The resulting graph is flatter and a node has in average less outgoing transitions. Due to this, the classifiers have to choose between less alternatives, which makes the classification easier. In our opinion, the more sophisticated representation of the global sequence graph therefore better fits the requirements.

Note that the proposed approach uses a fixed feature set which all skills would have to use. A dynamic feature set depending on the skill would require further adaptions of the approach as well as of the software framework.

5 Conclusion and Future Work

In this thesis, 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 HMMs) 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.

In future work some simplifications made in this thesis will be relaxed. We aim at learning more complex 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. This synchronization requires that the causal and temporal characteristics of a skill are considered in the representation. For the learning, we still plan to use a predefined set of MPs. However, the set may contain irrelevant and redundant primitives. Additionally, we plan to use a segmentation algorithm instead of labeling the data manually. In Section 4.2 we presented other possible enhancements that could improve the accuracy of the system. These enhancements will be considered when extending the presented approach to a more complex sequential skill learning system.

A Appendix

A.1 Dynamic Movement Primitives

A Dynamic Movement Primitive (DMP) consists of a set of ordinary differential equations:

$$\tau \dot{v} = K(g - x) - Dv + f \tag{A.1}$$

$$\tau \dot{x} = v \tag{A.2}$$

$$\tau \dot{s} = -\alpha s \tag{A.3}$$

The first two equations build the transformation system. It can be seen as a basic point attractor system in which a damped spring is attached to a goal position g. The system is perturbed by a non-linear acceleration f using the equations

$$f(s) = \frac{\sum_i \psi_i(s) w_i}{\sum_i \psi_i(s)} s(g - x_0) \tag{A.4}$$

$$\psi_i(s) = e^{-h_i(s-c_i)^2}. \tag{A.5}$$

The perturbation forces the system to follow the desired trajectory. Eq. (A.3) is known as the canonical system. It is initially set to 1 and converges to 0, thus monitoring the task progress. It can be easily seen, that $\lim_{s\to 0} f(s) = 0$. Thus f vanishes and the combined system converges to the unique attractor point g. The parameters of f can be learned with state of the art regression methods.

The advantages of the DMP formulation are the compact description and the time-invariance due to the canonical system. The representation is very robust to perturbations and allows a good reproduction of the movement in noisy environments.

A.2 Hidden Markov Models

A Hidden Markov Model (HMM) is a simple dynamic Bayesian network. A model consists of a fixed amount of states whereby each state produces an observable output. The output is a probabilistic function which means each possible output can be produced by a state with a certain

probability. The states itself are hidden from the observer and can only be guessed by the output observations. Formally, a HMM is a 5-tuple

$$\lambda = (S; V; A; B; \pi) \tag{A.6}$$

$$S = \{s_1; ...; s_n\} V = \{v_1; ...; v_m\} \tag{A.7}$$

where S is the set of states, V is the alphabet of the possible observations, which are called emissions in the context of the HMM. $A \in \Re^{n \times n}$ is the transition matrix. The element a_{ij} of the ith row and jth column of the matrix represents the probability of switching to state j when being in state i. $B \in \Re^{n \times m}$ is the observation matrix, with the element $b_i(v_j)$ depicting the possibility of making the observation v_j in state s_i. $\pi \in \Re^n$ is a vector with

$$\pi_i = P(X_1 = s_i) \tag{A.8}$$

The presented equations describe a discrete output HMM. A continuous output can be used by replacing V and B with a Gaussian distribution. Each emission of a state is then represented by the mean and variance of the Gaussian. In general, every distribution function can be used like for example multiple Gaussians for each state.

The parameters of the HMM can be learned by using the Baum Welch algorithm, an implementation of the expectation-maximization algorithm. In our case, the transition probabilities can be estimated directly as the data is completely labeled. It is crucial to have a good guess for the initial parameters of the model. Otherwise the algorithm can give a sub-optimal result. To avoid overfitting and encode a sequence correctly, the number of states has to be chosen carefully. The Bayesian information criterion (BIC) can be used to get a good hint for the number of states.

A.3 Support Vector Machines

SVMs belong to the maximum margin classifiers and are trying to separate the feature space into hyperplanes. Every hyperplane represents one class and data points are assigned to classes depending on their position in the feature space. For a binary classification the goal is to minimize the equation

$$\min_{\boldsymbol{w}, b, \xi} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w} + C \sum_{n=1}^{N} \xi_n \tag{A.9}$$

under the constraints

$$y_i(\boldsymbol{w}^T\boldsymbol{\phi}(x_n)+b) \geq 1-\xi_n \tag{A.10}$$

$$\xi_n \geq 0, n=1,...,N. \tag{A.11}$$

Here, b is the bias, $\boldsymbol{w}$ is a weight vector, ξ_n are the slack variables and C is a regularization parameter which controls the penalty of the slack variables. The latter two parameter allow for misclassifications and make the algorithm more robust against outliers. For $\boldsymbol{\phi}$ any kernel function

$$K(x_n, x_m) = \boldsymbol{\phi}(x_n)^T\boldsymbol{\phi}(x_m) \tag{A.12}$$

can be used. It maps the features into a higher dimensional space in which the margin between the classes is maximized based on the previous equations. The assignment of a feature vector x to one of the two classes is done by evaluating

$$\text{sgn}(\boldsymbol{w}^T\boldsymbol{\phi}(x)+b). \tag{A.13}$$

Equations (A.9)–(A.11) form a quadratic programming problem that can be solved by introducing Lagrange multipliers. The approach can be extended to multiclass classification by using 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 often. We decided to use the freely available LIBSVM library [Chang and Lin, 2011] as implementation for the SVM and we use radial basis functions as kernels:

$$K(x_n, x_m) = \exp(-\gamma\|x_n - x_m\|^2), \gamma > 0 \tag{A.14}$$

The regularization parameter C of (A.9) and the kernel parameter γ of (A.14) are parameters which have to be defined manually. As LIBSVM allows for a different regularization parameter for each class, $k+1$ parameters have to be set beforehand. If the dimensionality of the feature space is not too high they can be found by doing a grid search, hence simply systematically testing different values of the parameters.

List of Figures

List of Algorithms

Bibliography

B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

A. Billard, S. Calinon, and F. Guenter. Discriminative and Adaptive Imitation in Uni-Manual and Bi-Manual Tasks. *Robotics and Autonomous Systems*, 54(5), 2006.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., 2006. ISBN 0387310738.

S. Calinon and A. Billard. Active teaching in robot programming by demonstration. In *IEEE Int. Symp. on Robot and Human Interactive Communication*, 2007.

S. Calinon, F. Guenter, and A. Billard. On learning, representing, and generalizing a task in a humanoid robot. *IEEE Trans. on Systems, Man, and Cybernetics*, 37(2):286–298, 2007.

C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Trans. on Intelligent Systems and Technology*, 2011.

C. Daniel, G. Neumann, O. Kroemer, and J. Peters. Learning sequential motor tasks. In *IEEE Int. Conf. Robotics and Automation*, 2013.

S. Degallier and A. Ijspeert. Modeling discrete and rhythmic movements through motor primitives: a review. *Biological Cybernetics*, 103(4):319–338, 2010.

C. Diehl and G. Cauwenberghs. Svm incremental learning, adaptation and optimization. In *Proc. Int. Joint Conf. Neural Networks*, 2003.

S. Ekvall and D. Kragic. Learning task models from multiple human demonstrations. In *IEEE Int. Symp. on Robot and Human Interactive Communication*, 2006.

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.

T. fan Wu, C.-J. Lin, and R. C. Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, pages 975–1005, 2003.

J. R. Flanagan, M. C. Bowman, and R. Johansson. Control strategies in object manipulation tasks. *Current Opinion in Neurobiology*, 16(6):650–9, 2006.

T. Flash and B. Hochner. Motor primitives in vertebrates and invertebrates. *Current Opinion in Neurobiology*, 15(6):660 – 666, 2005.

D. Forte, A. Gams, J. Morimoto, and A. Ude. On-line motion synthesis and adaptation using a trajectory database. *Robotics and Autonomous Systems*, 60:1327 – 1339, 2012.

J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. *IEEE Trans. on Signal Processing*, 52(8):2165–2176, 2004.

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. doi: 10.1109/ROBOT.2010.5509672.

D. Kulic, W. Takano, and Y. Nakamura. Representability of human motions by factorial hidden markov models. In *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2007.

D. Kulic, W. Takano, and Y. Nakamura. Combining automated on-line segmentation and incremental clustering for whole body motions. In *Int. Conf. Robotics and Automation*, 2008.

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.*, 31(3): 330–345, 2012.

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.

F. Meier, E. Theodorou, F. Stulp, and S. Schaal. Movement segmentation using a primitive library. In *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2011.

F. Meier, E. Theodorou, and S. Schaal. Movement segmentation and recognition for imitation learning. *Journal of Machine Learning Research*, 22:761–769, 2012.

G. Mongillo and S. Deneve. Online learning with hidden markov models. *Neural Comput.*, 20(7): 1706–1716, 2008.

J. Morimoto and K. Doya. Reinforcement learning of dynamic motor sequence: Learning to stand up. In *IEEE Int. Conf. Intelligent Robots and Systems*, 1998.

K. Muelling, J. Kober, and J. Peters. Learning table tennis with a mixture of motor primitives. In *IEEE/RAS Int. Conf Humanoid Robots*, 2010.

C. Nehaniv and K. Dautenhahn. *Imitation in animals and artifacts*, chapter The Correspondence Problem, pages 42–61. MIT Press, 2002.

B. Nemec and A. Ude. Action sequencing using dynamic movement primitives. *Robotica*, 30: 837–846, 2012.

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.

S. Niekum, S. Chitta, A. Barto, B. Marthi, and S. Osentoski. Incremental semantically grounded learning from demonstration. 2013.

A. L. Pais, K. Umezawa, Y. Nakamura, and A. Billard. Learning robot skills through motion segmentation and constraints extraction. HRI Workshop on Collaborative Manipulation, 2013.

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

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. doi: 10.1109/ICRA.2011.5980200.

P. Pastor, M. Kalakrishnan, L. Righetti, L. Righetti, and S. Schaal. Towards Associative Skill Memories. In *IEEE/RAS Int. Conf. Humanoid Robots*, 2012.

V. Pavlovic, J. M. Rehg, and J. Maccormick. Learning switching linear models of human motion. In *Neural Information Processing Systems*, 2000.

J. Peters. Machine learning of motor skills for robotics. Technical Report 05-867, USC, 2005.

J. C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.

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

F. Sha and L. K. Saul. Large margin hidden markov models for automatic speech recognition. In *Advances in Neural Information Processing Systems*, 2007.

W. Takano and Y. Nakamura. Humanoid robot's autonomous acquisition of proto-symbols through motion segmentation. In *IEEE/RAS Int. Conf. Humanoid Robots*, 2006.

A. Ude, A. Gams, T. Asfour, and J. Morimoto. Task-specific generalization of discrete and periodic dynamic movement primitives. *IEEE Trans. Robotics*, 26(5):800–815, 2010.

Y. Zhang, L. Chen, and X. Ran. Online incremental em training of gmm and its application to speech processing applications. In *IEEE Int. Conf. Signal Processing*, 2010.