Movement Primitive Sequencing via Attribute Grammars

Rudolf Lioutikov[1] and Jan Peters[1,2]

***Abstract*— In this work we propose attribute grammars as a mechanism to sequence movement primitives. Attribute grammars extend probabilistic context-free grammars by introducing attributes and conditions to the grammar symbols and rules. We show how such grammars can be applied to solve complex tasks by sequencing simpler subtasks. Each subtask is represented as movement primitive and the main task is solved by a sequence of primitives. By defining a general set of attributes and a corresponding evaluation scheme we introduce a general framework to transform probabilistic context-free grammars for movement primitive sequencing tasks to attribute grammars. We apply an attribute grammar to solve the task of picking and placing a stone in a game of tic-tac-toe.**

I. Introduction

Movement primitives (MPs) are a well established concept in robotics. MPs are used to represent atomic, elementary movements and are, therefore, appropriate for tasks consisting of a single stroke-based or rhythmic movement [1]. They have been used in a large variety of applications, e.g., table tennis [2], pancake flipping [3] and hockey [1]. However, for more complex tasks a single MP is often not sufficient. Such tasks require sequences of MPs for feasible solutions.

Considering a set or library of MPs, such sequences can be generated in a variety of ways, including Hidden Markov Models [4], Mixture Models [5] and other hierarchical approaches [6]. These approaches can be regarded as mechanisms that produce sequences of MPs. This perspective reveals a common, important downside: understanding these mechanisms requires a significant amount of expert knowledge. However, a declared goal of robotics is the deployment of robots into scenarios where direct or indirect interactions with non-expert users are required. Therefore, more intuitive sequencing mechanisms for non-experts are necessary.

This work proposes the use of formal grammars for the sequencing of MPs. In particular, we suggest attribute grammars with a general evaluation scheme, presented in this paper, for sequencing tasks. Formal grammars represent a formal description of symbols and rules, representing the structure of a corresponding language. They have been intensively studied in both natural language processing and compiler construction but have also been applied in a variety of fields, e.g., molecular biology [7], bioinformatics [8], computer vision [9] and robotics [10], [11]. Probabilistic context-free grammars (PCFGs) allow the implicit embedding of hierarchies within the rules of the grammar associating every produced sequence with at least one corresponding parse tree. A parse tree represents the derivation of the produced sequence in an intuitive way. Figure 1 shows a user playing a match of tic-tac-toe against a robot where the robots next move is represented by an easily comprehensible parse tree. The grammar and the parse tree are shown more detailed in Figure 3.

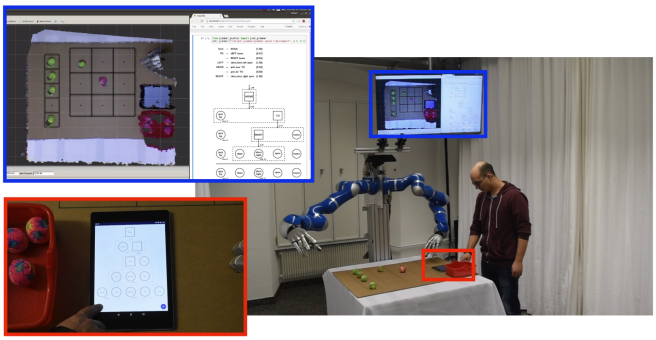

Fig. 1: Playing tic-tac-toe against the robot. The robot executes a sequence of primitives, picking up a stone and placing it on an empty field. The primitives correspond to terminals in an attribute grammar. The sequence of terminals was produced using the general evaluation scheme for movement primitive tasks presented in this paper.

Attribute grammars enhance the expressiveness of PCFGs even further. Attributes assigned to symbols contain values that evaluate during the production of the parse tree. Conditions defined on the attributes introduce control mechanisms into the production process that surpass common PCFGs. Furthermore, attribute grammars allow a straight forward incorporation of sensor input into the sequencing. For instance, the position of an object that is supposed to be picked up by the robot, could be encoded as an attribute. Depending on the attribute than the production of primitives that reach the object becomes more likely than production that would not be able to pick up the object.

Unfortunately these attributes enhance the complexity of the grammar and therefore make it less comprehensible for non experts. In this paper we present attributes and a corresponding evaluation scheme that generalizes across sequencing tasks, and needs only little to no adaptation for specific task, without the introduction of new attributes. While the introduced evaluation scheme appears complex at first, it is designed to stay mostly equal across different sequencing tasks. Therefore, the non-expert user does not have to be presented with the full scheme but merely with the attribute values specific to the task at hand.

A. Probabilistic Context-Free and Attribute Grammars

We first introduce the general concept of probabilistic context-free grammars and attribute grammars. A PCFG is a

[1]Intelligent Autonomous Systems, TU Darmstadt, 64289 Darmstadt, Germany, {lioutikov, peters}@ias.tu-darmstadt.de
[2]Max Planck Institute for Intelligent Systems, 72070 Tübingen, Germany

4-tuple $\mathcal{G} = \langle \mathcal{A}, \mathcal{V}, \mathcal{R}, \mathcal{S} \rangle$, consisting of a set of terminals $\mathcal{A} = \{a_1, a_2, a_3, \ldots, a_{|\mathcal{A}|}\}$, a set of nonterminals $\mathcal{V} = \{A_1, A_2, A_3, \ldots, A_{|\mathcal{V}|}\}$, a set of starting symbols $\mathcal{S} \subseteq \mathcal{V}$, and a set of production rules $\mathcal{R} = \{(A, \mathrm{R}_A, \boldsymbol{\rho}_A) | A \in \mathcal{V}\}$. The ordered set $\mathrm{R}_A \in ((\mathcal{A} \cup \mathcal{V})^+)^{|\mathrm{R}_A|}$ is referred to as the productions of the nonterminal $A \in \mathcal{V}$ and the elements $\rho \in \boldsymbol{\rho}_A$ of the multinomial $\boldsymbol{\rho}_A \in \Delta^{|\mathrm{R}_A|-1}$ are the probabilities or parameters of A. A common example grammar is the one describing the language $\mathsf{a}^n\mathsf{b}^n$

$$\begin{aligned} &\mathcal{G} = \langle \mathcal{A}, \mathcal{V}, \mathcal{R}, \mathcal{S} \rangle\,, \\ &\mathcal{A} = \{\mathsf{a}, \mathsf{b}\}\,, \quad \mathcal{V} = \{\mathsf{A}\}\,, \quad \mathcal{S} = \{\mathsf{A}\}\,, \\ &\mathcal{R} = \left\{ \left(\mathsf{A}, (\mathsf{ab}, \mathsf{aAb})\,, [0.7, 0.3]^T \right) \right\}. \end{aligned} \tag{1}$$

This grammar describes the language of all sequences that consist of any number of as followed by the same number of bs. A more common but less formal notation is

$$\begin{array}{lcl} \mathsf{START} & \rightarrow & \mathsf{A}\ (1.00) \\ \mathsf{A} & \rightarrow & \mathsf{ab}\ (0.70) \\ \mathsf{A} & \rightarrow & \mathsf{aAb}\ (0.30) \end{array} \tag{2}$$

which better illustrates, the function of a rule. The nonterminal on the left-hand side, A, can produce the sequences ab and aAb on the right-hand side with a probability of 0.7 and 0.3 respectively.

Attribute grammars are context-free grammars, where each terminal and nonterminal can be assigned one or more attributes. Usually attributes are distinguished into inherited and synthesized attributes. An inherited attribute belongs to a symbol on the right-hand side of a rule which obtains its value from attributes of the nonterminal of the left-hand side or other symbols on the right-hand side. A synthesized attribute is an attribute of the nonterminal on the left-hand side of a rule whose value is computed using attributes of the right-hand side symbols. The above example for instance can be transformed into an attribute grammar containing the attributes `depth` and `max_depth`.

$$\begin{array}{lcl} \mathsf{START} & \rightarrow & \mathsf{A}\ (1.00) \\ \mathsf{A} & \rightarrow & \mathsf{ab}\ (0.70) \\ & & \mathsf{A}.\mathtt{depth} = 1 \\ \mathsf{A}_1 & \rightarrow & \mathsf{aA}_2\mathsf{b}\ (0.30) \\ & & \mathsf{A}_1.\mathtt{depth} = \mathsf{A}_2.\mathtt{depth} + 1 \\ & & \mathsf{A}_2.\mathtt{max\_depth} = \mathsf{A}_1.\mathtt{max\_depth} - 1 \\ & & \mathtt{cond}\,(A_1.\mathtt{max\_depth} > 0) \end{array} \tag{3}$$

The indices of A_1 and A_2 simply distinguish between the same nonterminal within a single rule. The synthesized attribute `depth` evaluates to the number of recursion that occurred during the production of a sentence, while the inherited attribute `max_depth` defines how many recursion are at most supposed to occur. The latter is achieved by defining the condition that the second rule is only chosen if $\mathsf{A}_1.\mathtt{max\_depth} > 0$, resulting in sentences with at most `max_depth` number of as and bs. Such conditions extend the expressiveness of attribute grammars beyond the one of context-free grammars. For instance, the language $\mathsf{a}^n\ \mathsf{b}^n\ \mathsf{c}^n$ can not be represented by context-free grammars but easily by attribute grammars similar to the one above.

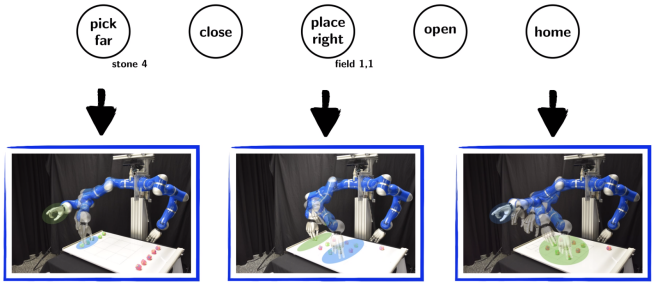

Fig. 2: A possible sequence of terminals produced by the attribute grammar and three of the corresponding primitives.

II. Attribute Grammars for Movement Primitive Sequencing

Defining such attributes and conditions requires detailed knowledge of the domain and the desired sentences the grammar is supposed to produce or parse, rendering the induction of an attribute grammar including the identification of the attributes very difficult if not infeasible. In our case we apply attribute grammars for the purpose of sequencing movement primitives. We define that each terminal in the grammar corresponds to a single movement primitive. A sequence of terminals therefore corresponds to a sequence of primitives as illustrated in Figure 2, where the terminal pick_far represents the execution of the corresponding primitive at this point in the sequence. The attribute stone 4 further indicates that the primitive would be conditioned on the position of the fourth stone.

Initially we induce a probabilistic context-free grammar from demonstrations[12]. Subsequently we extend this grammar with attributes and conditions that generalize to different movement primitive sequencing tasks and, therefore, need only little to no adaptation for specific tasks, with the exception of the initialization of the specific attribute values. We define the following attributes:

`transition`*:* This attribute defines where the current primitive ends and the next primitive is supposed to start

`endpoint`*:* The endpoint of a movement primitive.

`viapoints`*:* An ordered list of points that is supposed to be traversed by the sequence of primitives. The points are given in the state-space of the primitives. Once the first point is traversed by a primitive it is removed from the list and the next point is considered.

reachable*:* Given a primitive and a point in the primitive state-space, this condition evaluates if the point is reachable by the primitive.

producible*:* Given a nonterminal this condition evaluates to true if at least one of the corresponding right-hand sides is producible. A right-hand side is considered producible if all mandatory conditions evaluated to true, given the current set of attributes.

A. The tic-tac-toe pick-and-place task

We explain the functionality of the attributes in more detail using the example of the tic-tac-toe pick-and-place task. The

initial probabilistic context-free grammar, as seen in (4), was previously induced from observed demonstrations [12].

```
START  →  MOVE (1.00)
MOVE   →  pick_near TO (0.40) | pick_far TO (0.60)
TO     →  LEFT home (0.47) | RIGHT home (0.53)
LEFT   →  close place_left open (1.00)
RIGHT  →  close place_right open (1.00)
```
(4)

The production of the sequence always begins at the START nontermial. We assign two points, one for the position of a stone and one for the field the stone is supposed to be placed to the `viapoints` attribute. Furthermore, we set the `transition` attribute to the current position of the robot in the primitive state-space. We use the literals stone_pos, field_pos and cur_pos instead of the actual numerical values.

```
START.transition = cur_pos
START.viapoints = [stone_pos, field_pos]
eval: START
assert: producible(START)
assert: empty(START.viapoints)

START  →  MOVE (1.00)
MOVE.viapoints = START.viapoints
MOVE.transition = START.transition
eval: MOVE
assert: producible(MOVE)
START.viapoints = MOVE.viapoints
START.transition = MOVE.transition
```
(5)

The notation eval: indicates an evaluation of a nonterminal, e.g. eval: MOVE denotes, that at this point the MOVE nonterminal is evaluated. Moreover, we use the notation assert: to indicate that this condition must evaluate to true otherwise the entire right-hand side is removed from consideration as a possible production of the corresponding nonterminal given the current attribute set. If the START nonterminal is not producible, the task under the given attributes is not solvable. Furthermore, if the `viapoints`-list is not empty after evaluating START not all points were traversed and the task is also not considered solved.

The MOVE nonterminal contains multiple right-hand sides, each consisting of multiple symbols. The right-hand sides can be evaluated in parallel, i.e. the evaluation of each of the right-hand sides begins with the same set of attributes, independent of the changes that have occurred during the evaluation of the other right-hand side. In contrast, the symbols of a single right-hand side are evaluated sequentially, i.e, every symbol begins with the attributes set after the evaluation of the previous symbol. As mentioned before, terminals represent single movement primitives. It is important that a sequence of primitives does not contain any jumps in the state-space, since a real robot will not be able to make significant changes in its configuration instantaneously. Therefore, we ensure that every selected primitive starts where the previous primitive ended. In the proposed evaluation scheme for attribute grammars this is achieved by ensuring that the reachable condition holds for the primitive and the current `transition`-point. If the primitive can start from the `transition`-point, the `transition` attribute is set to the `endpoint` of the primitive afterwards. Furthermore, if there exists a point in the `viapoint`-list and the currently evaluated primitive can traverse that point it does so and extracts the point from the `viapoint`-list. Given that the possible adaptation of the primitive to the point could change the endpoint this has to happen before the `transition`-point is adapted.

```
MOVE  →  pick_near TO (0.40)
    assert: reachable(pick_near,MOVE.transition)
    if: reachable(pick_near,MOVE.viapoints[0])
        MOVE.viapoints = MOVE.viapoints[1 :]
    MOVE.transition = pick_near.endpoint
    TO.viapoints = MOVE.viapoints
    TO.transition = MOVE.transition
    eval: TO
    assert: producible(TO)
    MOVE.viapoints = TO.viapoints
    MOVE.transition = TO.transition
```
(6)

Only the evaluation for one of the two right-hand sides. The evaluation of the other right-hand side is defined analogously, but with the terminal pick_far instead of pick_near. The notation if: indicates that this condition is optional. The subsequent indented statements only occur if the condition evaluates to true. As mentioned before, the right hand sides are evaluated in parallel. Despite that both of the right hand sides next evaluate the TO nonterminal, the actual evaluations might differ due to two different sets of attribute values.

```
TO  →  LEFT home (0.47)
    LEFT.viapoints = TO.viapoints
    LEFT.transition = TO.transition
    eval: LEFT
    assert: producible(LEFT)
    TO.viapoints = LEFT.viapoints
    TO.transition = LEFT.transition
    assert: reachable(home,TO.transition)
    if: reachable(home,TO.viapoints[0])
        TO.viapoints = TO.viapoints[1 :]
    TO.transition = home.endpoint
```
(7)

Again two different possible right-hand sides are evaluated in parallel but only one is shown. The evaluation of the other right-hand side is defined equivalently, but with the nonterminal RIGHT instead of the LEFT. In contrast to the evaluation of MOVE the right-hand sides of TO require the evaluation of a nonterminal before the evaluation of a terminal.

B. A General Evaluation Scheme for Sequencing Tasks

A structure for both terminal and nonterminal evaluations is clearly evident. Every terminal a on the right-hand side of a rule with nonterminal A on the left-hand side is evaluated using the statements

```
assert: reachable(a,A.transition)
if: reachable(a,A.viapoints[0])
    A.viapoints = A.viapoints[1 :]
A.transition = a.endpoint
```
(8)

And every nonterminal B on the right-hand side of a rule with nonterminal A on the left-hand side is evaluated using

```
B.viapoints = A.viapoints
B.transition = A.transition
eval: B
assert: producible(B)
A.viapoints = B.viapoints
A.transition = B.transition
```
(9)

The presented evaluation schemes are very general and can be applied to any movement primitive sequencing task. Using not further specified via-points has the advantage that the evaluation does not restrict which primitive traverses which point. For instance, in the case of an obstacle it might be sufficient that the obstacle is passed at some point, but it does not necessarily matter which primitive avoids it. However, the unspecified list of via-points has a significant disadvantage. A primitive might require a certain via-point, for instance pick_near and pick_far have to know where the stone is positioned in order to pick it up successfully. Nothing in the current scheme associates via-points with a certain primitives. We solve this problem by introducing two additional attributes.

`keywords`*:* An unordered list of keywords. This attribute is assigned only to terminals before the evaluation and contains keywords identifying relevant points in the `targets` attribute.

`targets`*:* An unordered list of keywords, where each keyword is associated with an ordered list of points. The points are defined in the primitive state-space. A primitive containing a matching keyword in its `keywords` attribute extracts the first point in the corresponding list. The evaluation scheme for terminals is now defined as

```
assert: reachable(a,A.transition)
for: key in a.keywords
    assert: key in A.targets
    assert: reachable(a,A.targets[key][0])
    A.targets[key] = A.targets[key][1 :]
if: reachable(a,A.viapoints[0])
    A.viapoints = A.viapoints[1 :]
A.transition = a.endpoint
```
(10)

We introduce the for: notation to indicate an iteration and the in notation to indicate the existence of an element in a list. The evaluation scheme for nonterminals only changes such that the `targets` attribute is additionally passed down and received afterwards, analogously to the `viapoints` attribute.

C. Evaluating Parallel Attribute Sets

We already established, that the right-hand sides of a single nonterminal are evaluated in parallel. If more than one right-hand side evaluates without violating any asserts, there are consequentially multiple parallel sets of attributes returning from that nonterminal evaluation. Given that within one right-hand side the attributes are passed sequentially from symbol to symbol the question arises which of the multiple attribute sets should to be considered. A naive approach would be to select a random attribute sets. However, one attribute set might result in an unproducible right-hand side while another might not. We address this problem by storing every attribute set corresponding to a producible right-hand side in an ordered list. The order is defined randomly, while being weighted with the probabilities of the right-hand sides. Only the first set of attributes is considered, unless the set results in an assert violation, then the attribute set is discarded and the evaluation continues with the next set in the list. If no sets are left, the right-hand side is considered unproducible.

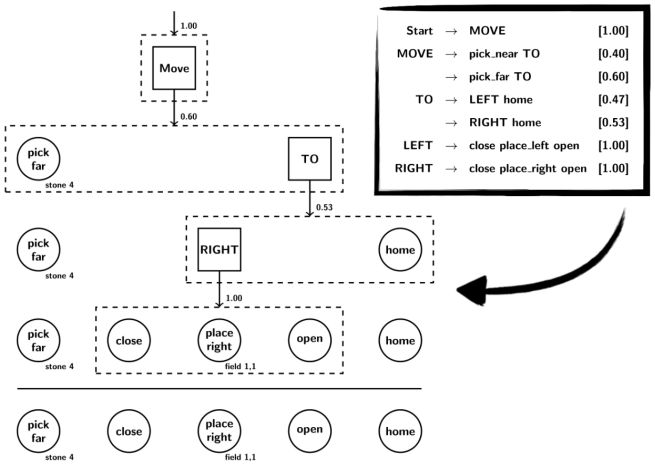

Fig. 3: A parse tree as produced by the attribute grammar for making a move in a game of tic-tac-toe. The grammar applied the general evaluation scheme for movement primitive sequencing as presented in this paper.

III. Experiments

We evaluated the described attribute grammar with the introduced evaluation schemes on a real robot platform using Probabilistic Movement Primitives (ProMPs)[1] as a movement primitive representation. We consider a point reachable if it is within $2\times$ the standard deviation of the ProMP distribution. However, other movement primitive representation as for instance Dynamic Movement Primitives (DMPs)[13] can also be used.

In the experiment a user was playing tic-tac-toe against the robot. The user was presented the grammar and a parse tree for every sequence produced by the grammar, including the information which stone would be picked and on which field the robot would place the stone. A corresponding parse tree and the producing grammar are shown in Figure 3. This allowed the user to not only foresee the robot behavior but also to understand why the robot chose this sequence of primitives, i.e., see which rules were applied at which time with which attributes. Given that the evaluation schemes are very general for both terminals and nonterminals it is not necessary to include the full schemes in the grammar or the parse trees presented to the user. In this experiment we restricted the presentation of the attributes solely to the `keywords` attributes and their value. The attribute value, e.g., the stone position was replaced with an index identifying the field the stone is on, since this might be more informative to a non-expert than the position in the corresponding primitive state-space.

Before every turn of the robot the positions of all stones were detected using a Kinect and a simple blob detection

algorithm. Afterwards one of the remaining robot stones was selected at random and a stochastic backwards induction algorithm determined the target field. The positions were assigned as $\texttt{target}[\text{stone}] = [\text{stone\_pos}]$ and $\texttt{target}[\text{field}] = [\text{field}_\text{p}\text{os}]$ respectively. Both the pick_near and the pick_far terminals were augmented with the $\texttt{keywords} = [\text{stone}]$ attribute. The place_left and place_right terminals were assigned the $\texttt{keywords} = [\text{field}]$ attribute. No further specifications or adjustments in addition to the general evaluation scheme had to be made.

IV. Conclusion

In this work, we introduced attribute grammars as a mechanism to sequence movement primitives. We defined attributes and conditions for sequencing tasks and introduced a general evaluation scheme. We enhanced an initially induced probabilistic context-free grammar for playing tic-tac-toe with the defined attributes and the evaluation scheme. The attribute grammar was used to produce sequences to pick up a stone and place it on an empty field using a real robot platform. In the future, we will extend the scheme to include the evaluation of concurrently executed primitives.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7-ICT-2013-10) under grant agreement 610878 (3rdHand) and from the European Union's Horizon 2020 research and innovation programme under grant agreement 640554 (SKILLS4ROBOTS).

References

[1] Paraschos, Daniel, Peters, and Neumann, "Using probabilistic movement primitives in robotics," *Autonomous Robots (AURO)*, accepted.

[2] Muelling, Kober, Kroemer, and Peters, "Learning to select and generalize striking movements in robot table tennis," *International Journal of Robotics Research (IJRR)*, pp. 263–279, 2013.

[3] Kormushev, Calinon, and Caldwell, "Robot motor skill coordination with em-based reinforcement learning," in *International Conference on Intelligent Robots and Systems, October 18-22, 2010, Taipei, Taiwan*. IEEE, 2010, pp. 3232–3237.

[4] Kulic, Ott, Lee, Ishikawa, and Nakamura, "Incremental learning of full body motion primitives and their sequencing through human motion observation," *I. J. Robotics Res.*, vol. 31, pp. 330–345, 2012.

[5] Lioutikov, Neumann, Maeda, and Peters, "Learning movement primitive libraries through probabilistic segmentation," *International Journal of Robotics Research (IJRR)*, 2017.

[6] Daniel, Neumann, Kroemer, and Peters, "Hierarchical relative entropy policy search," pp. 1–50, 2016.

[7] Chiang, Joshi, and Searls, "Grammatical representations of macromolecular structure," *Journal of Computational Biology*, vol. 13, pp. 1077–1100, 2006.

[8] Rivas and Eddy, "The language of RNA: a formal grammar that includes pseudoknots," *Bioinformatics*, vol. 16, pp. 334–340, 2000.

[9] Zhu, Mumford, *et al.*, "A stochastic grammar of images," *Foundations and Trends® in Computer Graphics and Vision*, vol. 2, pp. 259–362, 2007.

[10] Dantam and Stilman, "The motion grammar: Analysis of a linguistic method for robot control," *IEEE Trans. Robotics*, vol. 29, pp. 704–718, 2013.

[11] Lee, Su, Kim, and Demiris, "A syntactic approach to robot imitation learning using probabilistic activity grammars," *Robotics and Autonomous Systems*, vol. 61, pp. 1323–1334, 2013.

[12] Lioutikov, Maeda, Veiga, Kersting, and Peters, "Inducing probabilistic context-free grammars for the sequencing of robot movement primitives," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2018.

[13] Ijspeert, Nakanishi, Hoffmann, Pastor, and Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural computation*, vol. 25, pp. 328–373, 2013.