Semi-supervised Bootstrapping of a Movement Primitive Library from Complex Trajectories*

Andre Lemme1, René Felix Reinhart1 and Jochen Jakob Steil1

Abstract—Recent approaches have been advocated to learn a movement primitive library from demonstrations by using predefined motion features to identify and extract new movement primitives. In this paper, a new bootstrapping cycle is proposed, to build a suitable movement primitive library by (i) perceiving known primitives in complex trajectories, (ii) learn either new primitives or refines old ones, and (iii) consolidate the library by deleting unused primitives. The main contribution is that the movement primitives are in the center of the bootstrapping cycle and are learned semi-supervised on bases of the notion of co-articulation. We evaluate the learning behavior of this bootstrapping cycle in a toy example and with complex handwriting trajectories. Finally, we demonstrate the bootstrapping cycle of movement primitives in a full skill learning example, where a human tutor teaches the humanoid robot iCub how to perform "fishing" motions with a fishing rod.

I. INTRODUCTION

In robotic motion generation, imitation learning or programming by demonstration are general concepts to implement desired and task relevant movement primitives (MPs) [1], [2], [3]. However, single MPs are typically small entities and richness of motion results from the combination of several such building blocks.

Recent research has therefore advanced to represent complex trajectories with a library of movement primitives. A prerequisite to organize and adapt a movement primitive library (MPL) is the integration of decomposition and composition of MPs to model complex motions [1]. Given a library of movement primitives, the decomposition should identify a possible combination of these MPs to reproduce a task. However, the main question in such a learning architecture is: How can a library of MPs be autonomously bootstrapped?

The issue of identifying and extracting MPs autonomously from complex trajectories using a MPL has been addressed in [4], [5], [6]. The decomposition is accomplished by maximizing the likelihood that the demonstrated trajectory can be approximated by a sequence of MPs from the library. If no primitive exceeds a given threshold for the likelihood, a new MP is learned from the demonstration. In [7], an incremental learning scheme is proposed to sequentially learn a MPL. The library is represented by hidden Markov models (HMMs), which allow recognition of the MPs from an online stream of control commands. Another stochastic learning approach used in an incremental learning scenario is introduced in [8], [9], [10]. HMMs are used to represent MPs and also to obtain stochastic segmentations of complex motions into known primitives. In all these mentioned approaches segments of the demonstration which are not known in the MPL are used to learn new MPs as proposed in the programming by demonstration paradigm. In other approaches, segmentation points are solely defined by geometrical or kinematic features, e.g. critical points [11], zero velocity [12], spatio-temporal features [13], or general motion laws believed to underly human motion generation, e.g. two-thirds power law [14], minimum jerk [15]. These methods can be characterized as approaches where predesigned motion features directly determine the structure of the MPL.

We propose an autonomous learning architecture, where a MPL is bootstrapped by using the motion capabilities currently available, starting from only one straight MP. The MPL is used to compose and decompose complex trajectories by sequencing primitives from the library. The main contribution is to bootstrap new MPs by concatenating frequently used MPs to more complex ones. This is done by using self-generated training data, based on the notion of co-articulation [16] found in the human motion training process. That is, demonstrations do not serve as immediate training targets. Instead, reproduced trajectories generated with the current MPL are used for semi-supervised learning of new MPs. The MPL is further refined during multiple repetitions of a task without the prior selection of specific motion features. The concept of semi-supervised learning or
self-learning is mainly known in the classification and object detection domain [17], [18], where e.g. a classifier is trained at first with a small training data set. Then this classifier labels new data samples, which are used again to train the same classifier.

The paper is structured as follows: The bootstrapping cycle approach is introduced in Sec. II and a proof of concept example using artificial data is presented in Sec. III-A. In Sec. III-B, human handwriting motions are modeled with our approach. In Sec. IV, a robotic experiment using a full scale learning architecture on the humanoid robot iCub is discussed (see Fig. 1), before we conclude our work in Sec. V.

II. BOOTSTRAPPING OF A MOVEMENT PRIMITIVE LIBRARY

The core idea of the proposed approach is to combine three major steps. The first step is the processing, where we use the existing MPL to decompose a complex trajectory and generate new training data for the MP learning. The second step comprises learning, where the new training data is organized in training data sets for creating new or refine existing MPs. The third step consolidates the MPL by deciding which MP needs to be deleted.

Fig. 2 illustrates the steps of one such bootstrapping cycle obtained from decomposition. The sequence of MPs (see Fig. 2(1)) is used to self-generate articulated trajectory chunks (see Fig. 2(2)) as is described in more detail in Sec. II-C. These chunks are used either to create new MPs (Fig. 2(3)) or to refine existing MPs (Fig. 2(4)) as described in detail in Sec. II-C. The MPL holds additional information for each MP, e.g. how frequently a MP was used for decomposition and for how long it has been part of the library. After the learning phase, unused primitives are deleted (Fig. 2(5)). Due to the autonomous decomposition, it is not necessary to involve a human designer or teacher, who organizes training data before learning.

A. Movement primitive representation

In principle any MP representation can be used in this approach to provide the basic building blocks. We choose a standard dynamic movement primitive (DMP) approach [19]. DMPs comprise a spring-damper system, which is perturbed by a non-linear function $f$. This allows to represent arbitrarily shaped and smooth reaching motions as dynamical system:

$$\frac{1}{\tau} \ddot{x}_t = K(x_T - x_t) - D \dot{x}_t - K s(x_T - x_0) + K f(s),$$

which is coupled with a canonical system $\dot{s}_t = -\frac{1}{\tau}$, where $K, D$ are stiffness and damping constants with $D = 2\sqrt{\tau}$ to generate a critical damped system. The motion duration is determined by $\tau$. Eq. (1) is invariant to the relative position of the start and goal position. Stability of this dynamical system is ensured by the perturbation $f$ decreasing to zero at the end of the motion. Then, motion generation by Eq. (1) results in a linear convergence to the goal state $x_T$. For supervised learning of $f$, target outputs are computed according to

$$f_{\text{target}}(s) = - (x_T - x_0) + D K \dot{x}_t + s(x_T - x_0) + \frac{1}{K} \ddot{x}_t. \quad (2)$$

Then the regression problem is given by $L = \sum (f_{\text{target}}(s) - \hat{f}(s))^2$. Typically, the function $\hat{f}$ is learned as a superposition of Gaussian basis functions. However, in the following an Extreme Learning Machine (ELM, [20]) is used to represent $\hat{f}$. To exploit smoothing effects of the distributed hidden representation for learning new primitives by co-articulating successive primitives. ELMs are feed-forward neural networks that combine a non-linear and high-dimensional random projection of inputs into a hidden layer $h \in \mathbb{R}^M$ with efficient linear regression learning of a perceptron-like read-out layer $W^{\text{out}}$. The outputs $y$ are calculated according to

$$\hat{f}(s) = y = W^{\text{out}} h(s) = W^{\text{out}} \sigma(w^{mp}s),$$

where $\sigma(a_i) = (1 + \exp(-a_i - b_i))^{-1}$ are sigmoid activation functions applied to the neural activities $a_i = w_i^{mp}s$ for
all hidden neurons $i = 1, \ldots, M$. The input weight vector $\mathbf{w}_i^M \in \mathbb{R}^M$ and biases $b_i$ are randomly initialized according to a uniform distribution in $[-1, 1]$ and remain fixed.

### B. Processing of complex trajectories

The processing step tackles two tasks. First is the decomposition of the complex trajectory w.r.t. the known MPL (see Fig. 2(1)). Second is the preparation of training data for the learning step (see Fig. 2(2)).

1) Decomposition with a movement primitive library:

We use the decomposition algorithm introduced in [21] in order to decompose arbitrarily complex 2D trajectories w.r.t. a given MPL. The algorithm considers a MPL where MPs are represented by a collection of discrete shapes. This allows to use any MP representation.

The iterative process of the decomposition starts with a rough approximation of the demonstrated trajectory by a single MP and approximates step-by-step the geometric features of the trajectory. Segmentation points are discovered simply by a heuristic of maximum error by comparing the demonstration with the current best approximation. For more details of the decomposition algorithm, we refer the reader to [21].

2) Self-generating training data:

The bootstrapping cycle starts with only one single MP in the MPL, which is a straight line generated by the spring-damper system Eq. (1) with constant $f = 0$. The self-generating of new training data is based on the concept of co-articulation illustrated in Fig. 3:

- a) The process starts with an initial decomposition of a given demonstration (see Fig. 3(a)).
- b) We take each used primitive (see Fig. 3(b) straight lines) and two consecutive primitives from the reproduced trajectory (see Fig. 3(b)(1,2)).
- c) and add these extracted trajectories (compare Fig. 3(c)) to the training data (see Fig. 2(2)).

If $J$ segments are used for the decomposition, then $N = 2J - 1$ new trajectory chunks $\Omega_{\text{chunk}} = \{\hat{x}_{\text{chunk}}^{1}, \ldots, \hat{x}_{\text{chunk}}^{N}\}$ are sent to the learning step as illustrated in Fig. 2 and described in the following.

![Image](a) Initial decomp. (b) Co-articulation (c) New training data

Fig. 3. Toy example which illustrates the self-generation of new training data. The initial decomposition is shown in Fig. 3(a), where the segmentation points are indicated in magenta points and the reproduced trajectory produced by the primitives in green straight lines (i.e. consider that the library consists only of one straight line MP initially). This sequence of segments are then used to create new MP candidates by articulating two consecutive segments as shown in Fig. 3(b). These trajectory chunks are normalized and used as training data for learning. Fig. 3(c) illustrates the normalized training data.

### C. Semi-supervised learning

The processing step described in the previous section delivers chunks of trajectories $\Omega_{\text{chunk}}$ as shown in Fig. 3 which are used as training data. Each MP in the MPL is represented by a normalized trajectory $\Omega_{\text{MP}} = \{x_{\text{MP}}^{1}, \ldots, x_{\text{MP}}^{M}\}$, where $M$ is the number of the currently available MPs. To determine if a MP is already in the MPL, each trajectory in $\Omega_{\text{chunk}}$ is compared to each MP in the $\Omega_{\text{MP}}$. We focus in the comparison on the geometrical shape called “path” of the trajectory. Therefore, each trajectory $x_{\text{MP}} \in \Omega_{\text{MP}}, \Omega_{\text{chunk}}$ is resampled with $T_{\text{norm}}$ equidistant points representing only the path $\bar{x}$ of the trajectory $x$. The resampling is done by cubic spline interpolation. Two paths $\hat{x}_{\text{chunk}}$ and $\hat{x}_{\text{MP}}$ are compared by the $R^2$ metric:

$$R^2 = 1 - \frac{\sum (\hat{x}_{\text{MP}}(i) - \hat{x}_{\text{chunk}}(i))^2}{\sum (\hat{x}_{\text{MP}}(i) - \bar{x})^2}.$$  

where $\hat{x}_{\text{MP}}$ is the path represented by a MP stored in the MPL and $\hat{x}_{\text{chunk}}$ is the new path from the training data. A perfect match is indicated by $R^2 = 1$.

We introduce a threshold $\Theta$, which we choose between $0 < \Theta < 1$. With this similarity measure we can organize the training data and assign each trajectory $\hat{x}_{\text{chunk}} \in \Omega_{\text{chunk}}$ to the training data set corresponding to the MP represented by $x_{\text{MP}}$. Now two different cases can be identified: (A) no $\hat{x}_{\text{MP}}$ is similar to the observed $\hat{x}_{\text{chunk}}$ (i.e. $R^2 < \Theta$). Then, $\hat{x}_{\text{chunk}}$ becomes a $\hat{x}_{\text{MP}}$ (i.e. a representative trajectory in the MPL for a DMP which does not exist yet) and $\hat{x}_{\text{chunk}}$ is added to the corresponding training data set as the first trajectory. Note that the number of MP in $\Omega_{\text{MP}}$ is growing in this case, although no DMP representation is learned for this specific MP. However, by using this schema all follow up trajectories in $\Omega_{\text{chunk}}$ are compared also to this representative of the new MP. (B) a DMP representation already exists with a similar path $\hat{x}_{\text{MP}}$. Then the trajectory $\hat{x}_{\text{chunk}}$ is added to the training data set of the corresponding MP.

After the organization of the training data into training data sets the learning is triggered. To cope with the two possible cases (A) and (B), we use the on-line variant of the ELM denoted by on-line sequential ELM (OS-ELM [22]). Learning is organized in an initial learning phase and a sequential learning phase as explained in the following.

(A) the training data set can contain one or more trajectories and is used in the initial learning phase of a new DMP representation with $k = 0$:

$$P_k = (H_k^T H_k + \lambda I)^{-1},$$  

$$W_k^{\text{init}} = P_k H_k^T Y_k,$$

where $I$ is the identity and $\lambda > 0$ a regularization parameter. The matrix $H_k = (h(s(1)), \ldots, h(s(N_t)))$ harvests the hidden states for each input $s(l): l = 1 \ldots N_t$ in the training data set.

(B) A DMP representation already exists and needs to be...
TABLE I
IMPORTANT PARAMETERS WHICH NEED TO BE SET BY THE USER.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Experiment reference</th>
<th>Sec. III-A,B</th>
<th>Sec. IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decomposition</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>Error margin</td>
<td>0.025</td>
<td>0.075</td>
</tr>
<tr>
<td>$\delta$</td>
<td>Challenge seg. points</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>$\lambda$</td>
<td>Extreme learning machine</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$w^{inp}_{yp}$</td>
<td>Regression parameter</td>
<td>$1e^{-10}$</td>
<td></td>
</tr>
<tr>
<td>$K$</td>
<td>Number of hidden neurons</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>$\tau$</td>
<td>Dynamic Movement Primitive</td>
<td>uniform in $[-1,1]$</td>
<td></td>
</tr>
<tr>
<td>$\theta$</td>
<td>Clustering</td>
<td>0.98</td>
<td>1</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Threshold to add new MPs</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Threshold to delete old MPs</td>
<td>0.98</td>
<td>1</td>
</tr>
</tbody>
</table>

Fig. 4. A toy example to illustrate the learning process. The decomposition after 20 iterations of the learning cycle is shown in Fig. 4(a). The learned library is shown in Fig. 4(b). The number of used primitives in the decomposition during the learning process and the number of primitives in the MPL is shown in Fig. 4(c) together with the approximation error (RMSE), over 10 trials and 20 iterations.

As a first example we use a trajectory in form of three concatenated parabola shapes with different scaling and orientation as illustrated in Fig. 4.

We iterate the learning cycle for $G = 20$ iterations. The result is shown in Fig. 4. The approximation of the complex trajectory is shown in Fig. 4(a) and the learned library is shown in Fig. 4(b). It contains seven different shapes, beginning with the straight MP in the upper left corner. Note that besides the straight line and the MPs used in Fig. 4(a), three more primitives are in the MPL. Those additional primitives can be categorized into two different groups. First, there are old primitives which have not yet been deleted. Second, there are new MPs which have been added in the last iteration of the bootstrapping cycle, e.g. the bottom MP shaped like the letter ‘m’. We evaluated the number of primitives during 10 trials of the bootstrapping cycle. In Fig. 4(c), the used number of primitives over time is illustrated. Note that in the beginning of the learning process the total number of MPs in the library is growing quickly. After five iterations, however, the number of MPs starts decreasing due to the deletion of unused primitives. The number of primitives, which have $1 \ll \gamma$ and are available for the decomposition is also shown in Fig. 4(c). After the fifth iteration, the number of used primitives is almost constant and varies only marginally, which means that the bootstrapping is finished.

### B. Learning from handwriting demonstrations

After the proof of concept, a more complex trajectories from human handwriting are considered. A human subject demonstrated four handwriting motions of the word “Amarsi” on a tablet (see Fig. 6(a)). The same parameters from Sec. III-A are used (see Tab. I). In Fig. 5, the repro-
A recorded handwritten demonstration is used for bootstrapping a MPL. The decomposition after 100 iterations of the bootstrapping cycle and 10 trails is shown in Fig. 5(a). In Fig. 5(b), all MPs in the MPL after 100 iterations are shown. Note that the bootstrapping found complex primitives representing complex shapes like an ‘s’, ‘n’ or the first part of an ‘a’. The number of used primitives in the decomposition during the learning process and the number of primitives in the library are shown in Fig. 5(c) together with the approximation error (RMSE).

In this complex scenario, the qualitative results from the proof of concept are reproduced. In Fig. 5(c), the characteristic increase of the number of MPs in the library persists longer than in the example shown in Fig. 4. In all trials, the number of primitives is drastically reduced before the 20th iteration. This behavior can be modified by increasing the parameter $\gamma$, which will result in a slower decay of the number of MPs. In comparison to the example from Sec. III-A, the complexity of the MPs in the library is increased. The bootstrapping cycle based on co-articulation results in complex MPs, which approximate distinct shapes e.g. ‘s’, ‘n’ or the first part of an ‘a’ (see Fig. 5(b)). We use the point wise root mean square error (RMSE) between the demonstration and the recomposed motion to evaluate the performance. Note that after the 20th iteration the RMSE and the number of used MPs in the reproduction is almost constant. However, the number of MPs in the library is still varying. This indicates that the bootstrapping cycle is still adding new primitives, but these are not used in the decomposition of the complex trajectory and are deleted again. The amount of variation, depends on the ability to (i) approximate the new shapes and (ii) been able to identify it again in the complex trajectory. If a perfect solution for (i) and (ii) is found, the bootstrapping cycle would end with one MP in the MPL which can approximate the full complex trajectory and the variation would be zero.

The bootstrapping cycle is demonstrated in a robotic scenario with the humanoid robot iCub. A human teacher demonstrates in a physical interaction with iCub (i.e. kinesthetic teach-in) how to catch a “fish” with a toy fishing rod (see Fig. 1). This section demonstrates the application of semi-supervised bootstrapping of the MPL in a larger motion skill architecture.

A. Representation of a motion skill

A motion skill is constituted by a combination of several, partial representations, which address particular features of the skill and have different degrees of invariance. In a lower representational level, the tool kinematics for handling the fishing rod is learned from a kinesthetic teach-in similar to the work in [12]. In a higher level, which is in the focus of this paper, the “fishing” motion is represented by a set of MPs. The MPL is bootstrapped from complete demonstrations of a skill in an unsupervised fashion without predesigned or manual segmentation. Both the tool kinematics as well as the MPL can be re-learned and refined over the learning process.

In Fig. 6(b), the generalization performance of the learned library to new demonstrations is illustrated. Learning is conducted only on one “Amarsi” demonstration, whereas three more demonstrations serve as a test set. The test demonstrations are decomposed with the learned MPL. Fig. 6(b) clearly shows that the generalization to new demonstrations is possible with similar reproduction performance as on the training demonstration.

IV. TEACH ICUB HOW TO FISH

Fig. 6. Demonstrated trajectories (left). Amarsi 1-4 starting from the top. Generalization error (RMSE) statistics for four "Amarsi" handwriting motions (right). Solid lines depict RMSEs averaged over 10 trials of the bootstrapping cycle. The shaded areas depict standard deviations.
course of repeated demonstrations, i.e. are subject to open-ended learning which is also reflected on an architectural level in terms of a changing set of MPs.

In Fig. 7, the experimental setup for learning the fishing skill is illustrated. To indirectly be able to track the fishing hook, the robot is equipped with a 3D marker tracker in order to perceive the position of the fishing hook slightly below the marker. We use iCub cameras located in the head and the Image Component Library (ICL) available from here [23] for 3D tracking of the marker. Note, however, that the robot is not able to control the tool in the beginning, i.e. to position the fishing hook at a target, nor does it have a representation of a complex motion which is necessary to hook up the fish successfully. Both issues of learning the tool kinematics and the formation of a compact MP representation of the tool-tip motion are addressed in the skill architecture.

The following representational levels are shown in Fig. 7:

- **Movement Primitive Sequence**: Motions of the fishing hook are encoded as a sequence of MPs, which are formed previously by the proposed bootstrapping cycle. The motions are defined in a two-dimensional space which is sufficient to explain the considered fishing motions. Elements of this space are denoted by \( u \) and describe the projected 3D position of the marker attached to the fishing hook. The projection is performed by a principal component analysis (PCA).

- **Task Expansion**: The task expansion maps the current target from the compact movement space \( u \) to an explicit task formulation, i.e. end effector position and orientation of the right arm. For this purpose, the 2D target \( u \) generated by the current MP is first projected back to a 3D position by the PCA. Then, this 3D position of the fishing hook is mapped to the end effector position and orientation of the robot arm by means of a trained feedforward neural network. The learning of task expansions has been addressed before in [12].

- **Inverse Kinematics**: We use the inverse kinematics controller available in the iCub software repository [24] to control the right arm of the iCub according to the targets provided by the task expansion.

In the following, the formation of the MPL from a demonstration of the fishing skill is presented.

### B. Learning how to fish

In the kinesthetic teaching phase, the human teacher can move the robot’s arm while it holds the fishing rod. The controller for the right arm is switched to joint impedance mode using the force control implementation from the iCub software repository [25], [26], such that the arm is compliant to the external forces applied by the teacher.

After teaching, the recorded trajectories from the marker and joint angles are used for learning the tool kinematics and PCA. Then, the MPL is bootstrapped. In Fig. 8, the original demonstration (trajectory of the marker projected onto a
plane) of the fishing skill is illustrated in blue. In Fig. 8(a) the initial decomposition (green) with the MPL containing only the straight line is shown. Segmentation points found by the decomposition algorithm are indicated by magenta circles. Note that almost the same parameters are used as in the previous experiments (compare Tab. I) only the error margin \( \varepsilon \) is slightly increased to cope with the noise in the data.

After 4 iteration of the bootstrapping cycle, the decomposition shown in Fig. 8(b) is obtained. Note that the jerky demonstration is represented by smooth and well-formed MPs, which shows the robustness of the learning process against noise in the demonstration. The corresponding MPL is depicted in Fig. 9. A compact encoding of the skill is against noise in the demonstration. The corresponding MPL is depicted in Fig. 9. A compact encoding of the skill is shown in Fig. 8(b) is obtained. Note that the jerky demonstration is represented by smooth and well-formed

V. CONCLUSIONS

In this paper, a learning architecture to bootstrap a movement primitive library is introduced. The bootstrapping cycle integrates decomposition and composition of movement primitives to model complex trajectories. Movement primitives are learned semi-supervised and evolve from a single straight line primitive through co-articulation and refinement. The result is a compact movement primitive library, which generalizes to novel complex trajectories. The bootstrapping of the movement primitive library addresses both segmentation and representation in a coherent framework solely based on the available movement primitives. Therefore, motion features for segmentation and learning have not to be predefined.

Future work will address detailed evaluation of the bootstrapped movement primitives with focus on the emerging kinematic properties. Also the impact of different movement primitive representations and different decomposition methods on the bootstrapping cycle are interesting research questions.

REFERENCES