An Iterative Algorithm for Forward-Parameterized Skill Discovery

Adrien Matricon*†‡ David Filliat*‡ Pierre-Yves Oudeyer†‡
* U2IS, ENSTA ParisTech, Université Paris Saclay, Palaiseau, France
† INRIA, Université de Bordeaux, Bordeaux, France
‡ Flowers Team, INRIA / ENSTA ParisTech, France
Email: {adrien.matricon, david.filliat}@ensta-paristech.fr, pierre-yves.oudeyer@inria.fr

***Abstract*—We introduce COCOTTE (COnstrained Complexity Optimization Through iTerative merging of Experts), an iterative algorithm for discovering discrete, meaningful parameterized skills and learning explicit models of them from a set of behaviour examples. We show that forward-parameterized skills can be seen as smooth components of a locally smooth function and, framing the problem as the constrained minimization of a complexity measure, we propose an iterative algorithm to discover them. This algorithm fits well in the developmental robotics framework, as it does not require any external definition of a parameterized task, but discovers skills parameterized by the action from data. An application of our method to a simulated setup featuring a robotic arm interacting with an object is shown.**

I. Introduction

Long term planning becomes a difficult problem in robotics when it is performed in the low-level motor control space. An efficient way to reduce its complexity is to introduce higher-level actions that abstract away the low-level difficulties and enable us to work in a smaller and discrete space. For example, in the context of reinforcement learning, [1] introduces the concept of *options* (also called *skills*), which are high-level actions leading to a known state. In this framework, primitive skills (*e.g.* joint torques and angles controllers) can be composed into higher-level ones (*e.g.* picking up an object, going to lunch, traveling to a distant city), forming a repertoire of skills that can be used for planning.

Skill knowledge is what allows an agent to select and chain discrete actions towards a goal, and is therefore crucial for many applications. While this knowledge can be carefully crafted by expert engineers, there has been an effort in recent years in developing methods for skill and parameterized-skill learning ([2], [3], [4], [5]).

A *skill* can be defined as the knowledge of both an *action* (usually a policy) and its *result* (reward, new state of the system, etc), given a specific setup or *context* (initial state, weights of elements, etc). As such, skills can be understood as instances of forward models. They can also be understood as instances of inverse models, as they contain both the knowledge of a *task* (*i.e.* context and desired result) and of an action that solves it.

In the literature, the concept of skills is often generalized to *parameterized skills* in an inverse sense ([2], [3], [4], [5]): each variation of a parameterized task is associated to an action that solves it (*e.g.* in the thought experiment presented in Figure 1, a task-parameterized skill could be *how to move the cube to a given position*, which is defined for a limited domain of $(x_{initial}, x_{edge}, \boldsymbol{x_{object}})$ and which associates to each reachable position x_{object} a value of $x_{actuator}$ that produces this result).

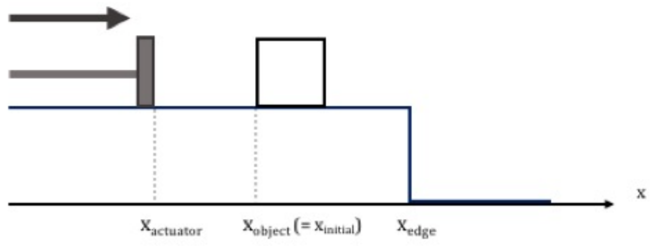

Fig. 1. Thought experiment: an actuator moves on a table along an horizontal axis x, its action parameterized by its final position $x_{actuator}$. It is moving towards an object at $x_{initial}$, in the direction of an edge located further at x_{edge}. $(x_{initial}, x_{edge})$ define the context of this setup, and its result is the final position of the object x_{object}.

In this paper, we introduce another way to generalize skills to parameterized skills, this time in a forward sense: the context and action are parameterized, and each of their variations is associated to the corresponding result (*e.g.* in the former example, a forward-parameterized skill could be *pushing the cube without it falling*, which is defined for a limited domain of $(x_{initial}, x_{edge}, \boldsymbol{x_{actuator}})$ and which associates to each value of $x_{actuator}$ the value of x_{object} resulting from this action).

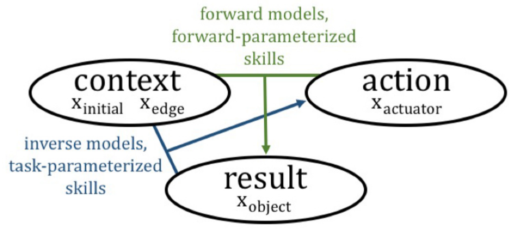

Fig. 2. Two types of parameterized skills

Figure 2 illustrates the difference between both parameterizations. Note that if a given result can be attained by various actions (like the final position x_{edge} which is attained by all values of $x_{actuator}$ superior to x_{edge} in the example), only one can be associated to a task instance by a task-parameterized skill, while a single forward-parameterized skill like *pushing the cube off the table* can cover all those cases, thus making this representation more versatile. Task-parameterized skills are very practical for industrial applications, with externally given tasks in controlled environments, but limit the adaptability in open-ended learning scenarios where having several ways to reach the same result may prove critical.

Forward-parameterized skills are particularly relevant to the field of developmental robotics where, without any external definition of parameterized tasks, a robotic agent is expected to discover sensorimotor contingencies, how to control its own body and how to interact with its environment [6]. Discovering forward-parameterized skills is a way to make representations of actions emerge that can be manipulated and chained at a symbolic level. Furthermore, similar forward models have already been shown to be very usable for planning [7].

In this paper, we introduce a formalism that reduces forward-parameterized skill learning to discovering and learning the smooth components of a locally smooth function. We propose an iterative algorithm to solve it by performing the constrained minimization of a complexity measure, and apply our method to a simulated setup featuring a robotic arm interacting with an object.

II. Related Work

A. *Task-parameterized skills*

While we focus on forward-parameterized skills, other researchers have investigated task-parameterized skills and how to build a function $\mathbf{a} = f(\boldsymbol{\tau})$, where $\boldsymbol{\tau}$ and $\mathbf{a}$ are vectors of parameters respectively representing a task and an action solving that task.

In [3], the usual Reinforcement Learning approach for a single task, *i.e.* to learn a policy of the form $\mathbf{u} = \pi_{\mathbf{a}}(\mathbf{s})$ (where $\mathbf{s}$ represents the state and $\mathbf{u}$ the control), is extended into a search for a general policy $\mathbf{u} = \pi_{\mathbf{a}}(\mathbf{s}, \boldsymbol{\tau})$ which solves all variations of the task. This approach enables an agent to learn a given task-parameterized skill, but does not address the problem of discovering and learning of several parameterized skills simultaneously.

In [5], parameterized skills are discovered through explicit learning of the function $\mathbf{a} = f(\boldsymbol{\tau})$. Some hypotheses enable the authors to ensure that each $\boldsymbol{\tau}$ corresponds to a unique $\mathbf{a}$, that f is locally smooth and that all values of $\mathbf{a}$ lie on disjoint manifolds in the action space. They then learn those manifolds, train an approximator on each one and map them to the task space. Discovering and learning the smooth components of f in such a way is expected to be of lower complexity and dimensionality than learning a global model covering the entire task space. While we share this idea of learning locally smooth low-complexity functions, applying the approach to forward models (discovering manifolds in the result space, learning approximators on each one and mapping them to the context-action space) cannot be done as there is not a one-to-one correspondence between possible manifolds in the result space and parameterized skills: a single result may be reachable by a variety of forward-parameterized skills, or even by an infinity of instances of a single parameterized skill.

B. *Locally smooth function approximation*

Our approach relies on approximating a locally smooth function and identifying its smooth components. From an algorithmic point of view, the idea of performing a regression to approximate a function that may not even be fully continuous has been explored in several learning algorithms.

In Locally Weighted Regression (LWR) [8], no explicit model is learned from the training data. When the value of the function at a given point $\mathbf{x}$ is needed, a local linear model is trained through linear weighted least square regression, with weights given by a kernel centered on $\mathbf{x}$ (generally a Gaussian kernel), giving more importance to data in the neighbourhood of $\mathbf{x}$ than to points far from it. That local model is then used to approximate the value of the function at $\mathbf{x}$. Locally Weighted Projection Regression (LWPR) [9] improves on LWR by performing a low-dimensional affine projection before fitting the linear model, allowing the regression to focus on the relevant dimensions. While both LWR and LWPR approximate functions that are not smooth and that are possibly only piecewise continuous, no explicit model is learned, making those methods impractical for our purpose. Furthermore, the locality of the regression also makes them data-inefficient.

In Kernel Ridge Regression (KRR) [10], also known as Kernel Regularized Least Squares, a global model is learned as a linear combination of basis functions, under the form:

$$f(\mathbf{x}) = \sum_{i=1}^{M} w_i \cdot k(\mathbf{x}, \mathbf{x}_i)$$

where the $(\mathbf{x}_i)_{i \leq M}$ are the input values of the datapoints and k is a kernel function (for example a Gaussian kernel). Gaussian Process Regression (GPR) is very similar to KRR with a Gaussian kernel, but slightly differs in the way the weights are computed during the regression [11]. Although KRR and GPR explicitly learn a model (unlike LWR and LWPR), it is a global model and not a collection of local models, which makes those methods impractical for discovering discrete components of a locally smooth function. Furthermore, the locality of the regression induced by the kernels makes those methods data-inefficient as well.

In Mixture of Experts (ME) [12], several approximators (generally neural networks) called *experts* are trained. The input space is split by a gating network that determines for each input the expert or blend of experts most likely to approximate the output. This approach seems to be what we need but, to the best of our knowledge, existing ME methods all fall short on one point: the number of experts and their complexity are not both automatically determined. Some adaptative approaches built on ME find ways to grow either the complexity or the number of experts after setting the other, like in [13] where there is only one expert at first and the number of experts increases as datapoints become more numerous, but to the best of our knowledge those approaches still require to set either the complexity or number of the experts before any learning occurs. Because of this, two or more experts might be needed to represent a single complex parameterized skill, while simple parameterized skills may be approximated by overly complex models. This makes the method data-inefficient and lowers its probability of generalizing well to new, unseen data.

C. *Constrained complexity minimization*

The idea of a constrained complexity minimization has been successfully applied to other problems in other fields of research. Rate-distortion theory [14] is a branch of information theory that deals with the question of representing a signal with as few bits by symbol as possible (complexity minimization) so that it can be reconstructed without exceeding a given distortion (constraint on the error). In motion planning, [15] approximate observed trajectories through a library of Dynamic Motion Primitives (DMP), using as few DMPs as possible under the constraint that the error must stay under a given threshold. The library itself is grown in such a way that it does not contain high-complexity DMPs if simpler ones have performed well on the past trajectories. While those approaches bear similarities to our own, they are applied to fundamentally different problems and are not usable for forward-parameterized skill learning.

III. FORWARD-PARAMETERIZED SKILLS

Let us use $\mathbf{c} \in \mathbb{R}^{N_c}$, $\mathbf{a} \in \mathbb{R}^{N_a}$ and $\mathbf{R} \in \mathbb{R}^{N_R}$ as notations for vectors of parameters respectively representing contexts, actions and results. We formally define a forward-parameterized skill as a deterministic function f, smooth over a subdomain of the context-action space, such that $\mathbf{R} = f(\mathbf{c}, \mathbf{a}) + \delta$ with δ a stochastic noise function. This assumes that f is unimodal and that stochasticity is entirely due to noise, which is a relatively common assumption. Similarly to [5], we expect parameterized skills to be of low complexity, compared to a global model covering the entire context-action space, and to each use only a subset of relevant dimensions from the context-action space.

Let us now consider the forward model of the robot's capabilities, describing all of its possible interactions with the world. Such a model would be of the form:

$$\mathbf{R} = F(\mathbf{c}, \mathbf{a}) + \Delta(\mathbf{c}, \mathbf{a})$$

where F is a deterministic function and Δ a stochastic noise function.

A parameterized skill can thus be seen as a restriction of F to one of the subdomains of the context-action space on which it is smooth. Learning parameterized skills is therefore equivalent to learning the smooth components of the locally smooth function F.

An intuition of this can be gained by looking at the simple thought experiment in Figure 1, where an actuator moves on a table along an horizontal axis towards an object. The result one would expect from such an experiment (represented in Figure 3) would show F smooth over three subdomains A, B and C, corresponding to three parameterized skills where the object is respectively left untouched, pushed forward on the table or pushed off the edge of the table. The frontiers between the subdomains reflect sudden changes in the context of the action (reaching the object or the edge of the table).

The goal of the algorithm we present in this work is to automatically discover forward-parameterized skills from the experimental data (*i.e.* to group together instances of the same parameterized skill while keeping distinct parameterized skills separate) and to learn an explicit model (*i.e.* an expert in the ME terminology) for each of them.

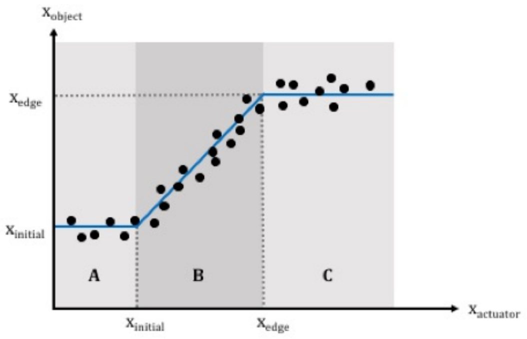

Fig. 3. Expected results of the thought experiment presented in Figure 1 $x_{object} = F(x_{actuator}) + \Delta(x_{actuator})$, with F (in blue) smooth on 3 subdomains A, B and C (see text for details)

In this work, we make the strong assumption (whose validity shall be discussed later on) that there is a known deterministic bound ε on Δ, such that for each dimension i and each value of $(\mathbf{c}, \mathbf{a})$, we have:

$$|\Delta^i(\mathbf{c}, \mathbf{a})| \leq \varepsilon^i(\mathbf{c}, \mathbf{a})$$

Each datapoint p in the training data is of the form $(\mathbf{R}_p, \mathbf{c}_p, \mathbf{a}_p, \boldsymbol{\varepsilon}_p)$, where $\boldsymbol{\varepsilon}_p = \varepsilon(\mathbf{c}_p, \mathbf{a}_p)$ can be interpreted as a precision on the value of $\mathbf{R}_p$.

IV. COCOTTE ALGORITHM

A. *Constrained complexity minimization*

At the heart of our approach is the idea of approximating F, which is a function that is only locally smooth. Functions that are not smooth can be approximated by smooth functions (most approximators are actually $\mathscr{C}^\infty$), but this generally comes at the cost of an increased complexity of the model (given a sensible measure of complexity).

The idea behind our algorithm is to explicitly minimize the overall complexity of a collection of models approximating F. As approximating this function with one model for each domain on which it is smooth should lead to a lower complexity than using a single model, we expect those domains to spontaneously emerge from the complexity minimization.

The quality of a model is also known to depend on how well that model's complexity reflects the complexity of the underlying reality it describes. Models that are too simple are generally unable to give good predictions, even on the training data (underfitting). Conversely, models that are too complex will have a low probability of generalizing well to new, unseen data [16] (overfitting). Given some constraints to prevent the models from being too simple, (*i.e.* to guarantee that good enough predictions can be given on the training data), we expect the collection of model resulting from a complexity minimization to have good generalization properties. We can also expect the algorithm to be data-efficient, as simple models need fewer datapoints to be trained.

B. Some definitions

We assume given (S_f, comp), where S_f is a set of families of approximator functions (*e.g.* the set containing the family of all first-order Fourier series, the family of all second-order Fourier series...), and $\mathrm{comp} : S_f \to \mathbb{N}$ is a complexity measure over this set (*e.g.* with the set from the previous example, the function associating each family of Fourier series to their order of approximation).

We define the *complexity of a function* f, as:

$$\mathrm{comp}(f) = \min_{\substack{s \in S_f \\ s \ni f}} \mathrm{comp}(s)$$

Further given a set S_d of M datapoints $(\mathbf{R}_i, \mathbf{c}_i, \mathbf{a}_i, \varepsilon_i)_{i \leq M}$, we define the *normalized error of a function* f *on* S_d the vector $\boldsymbol{E}(f) \in \mathbb{R}^{M \times N_R}$ such that:

$$\forall i \leq M,\ \forall j \leq N_R,\ \boldsymbol{E}(f)_{i,j} = \left| \frac{\mathbf{R}_i{}^j - f^j(\mathbf{c}_i, \mathbf{a}_i)}{\varepsilon_i{}^j} \right|$$

A function f is said to *explain* S_d if it verifies:

$$\forall i \leq M,\ \forall j \leq N_R,\ |\mathbf{R}_i{}^j - f^j(\mathbf{c}_i, \mathbf{a}_i)| \leq \varepsilon_i{}^j$$

that is to say if:

$$\|\boldsymbol{E}(f)\|_\infty \leq 1$$

Using this definition, we define the *complexity of a set of datapoints* as:

$$\begin{aligned}\mathrm{comp}(S_d) &= \min_{f \text{ explains } S_d} \mathrm{comp}(f) \\ &= \min_{\substack{s \in S_f \\ s.t.\ \exists f \in s \\ f \text{ explains } S_d}} \mathrm{comp}(s)\end{aligned}$$

C. An iterative algorithm for complexity minimization

We now introduce COCOTTE (COnstrained Complexity Optimization Through iTerative merging of Experts), an iterative algorithm to determine a collection of domains on which F is smooth and to learn explicit models approximating F on each of those domains. COCOTTE first assimilates those domains to sets of datapoints, proceeding in two steps:

1) It relies on *constrained complexity minimization* to determine, over all possible partitions of the datapoints, the one with the lowest sum of complexities and its associated models,
2) It then trains a classifier to determine the actual domains, *i.e.* to map every point of the context-action space to a set of datapoints and its associated model.

In step 1, trying all possible partitions of the datapoints and estimating their complexity has a high computational cost. Instead of exploring all those possibilities, we use a greedy iterative approach (see Algorithm 1), illustrated on Figure 4 on the case of a simple step function. This approach works in three phases:

1) The search is initiated in a state of high overall complexity, where each datapoint is in its own set and associated to a model that predicts it perfectly. This is the most granular partition of the datapoints.
2) Then sets of datapoints in this initial partition are iteratively merged into larger sets, decreasing the granularity and complexity of the partition. Candidate pairs of sets for merging are selected in a greedy manner, starting with the closest sets (the reason for this shall be discussed later on). If merging a pair of sets increases the overall complexity of the partition, they are left separate and the next candidate pair of sets is considered.
3) Finally, artifacts of the merging process are removed. They are sets of datapoints which appear early on in the merging phase at the frontiers between smooth components of F. We consider that a set whose points can all be distributed into other sets (also trying the closest ones first) without inducing a jump in complexity is an artifact, and that distributing its points should reduce the overall complexity of the partition.

As for step 2 of the algorithm, we use a standard Random Forest classifier. Though we did not explore the choice of classifier, other supervised classifiers should also work.

Algorithm 1 Step 1 of the COCOTTE algorithm

```
Input: $S_d$ set of datapoints
1) Initialization:
    $S_m \leftarrow \emptyset$                    ▷ partition of the datapoints
    for $p \in S_d$ do
        Add $\{p\}$ to $S_m$            ▷ Each point in its own set
    end for
2) Merging phase:
    repeat
        Select two sets $s_0, s_1$ in $S_m$
        if $\mathrm{comp}(s_0 \cup s_1) \leq \mathrm{comp}(s_0) + \mathrm{comp}(s_1)$ then
            Replace $s_0$ and $s_1$ by $s_0 \cup s_1$ in $S_m$
        end if
    until No two sets can be merged
3) Artifact elimination:
    repeat
        for $s \in S_m$ do
            Remove $s$ from $S_m$
            for $p \in s$ do
                Select a set $s' \in S_f$
                Add $p$ to $s'$
            end for
            if no overall complexity decrease then
                Roll back the changes, put $s$ back in $S_m$
            end if
        end for
    until No set can be distributed
```

D. Dealing with multidimensionality

When $\boldsymbol{R}$ is multidimensional or of unknown dimension, it may be more practical to use one-dimensional approximator functions over each dimension than to use multi-dimensional approximator functions over all dimensions at once.

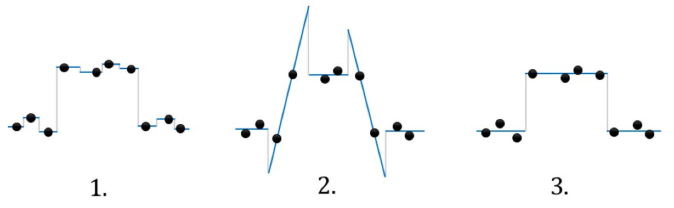

Fig. 4. Algorithm illustration on a simple step function: each point starts in its own set, then sets are iteratively merged together (starting with the closest ones), and finally artifacts at the frontiers between smooth components of F are removed and their point distributed into other models.

In this case, the complexity of $f = (f_0, ..., f_{N_R})$ is the multidimensional complexity $(\text{comp}(f_0), ..., \text{comp}(f_{N_R}))$ and the condition for merging two sets s_0 and s_1 is:

$$\forall j \leq N_R,\ \text{comp}(s_0 \cup s_1)^j \leq \text{comp}(s_0)^j + \text{comp}(s_1)^j$$

E. Implementation of COCOTTE

We implemented COCOTTE in C++ [1], using multivariate polynomial functions as one-dimensional approximators over each dimension. Given d available dimensions, the family of all polynomials of degree N using those dimensions is associated to the complexity $N \times d + 1$. As the complexity of a family of functions increases with d, the families using irrelevant dimensions are of higher complexity and discarded during the constrained complexity minimization, ensuring relevant variables selection.

Let s_0 and s_1 be two sets of datapoints that we are trying to merge, explained on a given dimension by polynomial functions of respective complexities c_0 and c_1. We know that if a merge is possible, on this dimension the complexity of the union of the sets falls between $\max(c_0, c_1)$ and $c_0 + c_1$. This allows us to use a binary search to search for function leading to the minimum complexity.

The distance between two sets of datapoints that we use to select pairs of candidates during the merging phase is the smallest Euclidian distance between a point of the first set and a point from the second set.

Determining if a given family of polynomial functions can explain the datapoints is done by searching for the function f within that family that minimizes the error $\|\boldsymbol{E}(f)\|_\infty$ and checking if this error is below 1. This search can be formulated as a linear program and performed by an adequate solver.

In order to greatly improve the speed of our implementation, we introduced a heuristic to reduce the number of families of polynomial functions that we have to consider: if we call V_0 and V_1 the sets of available dimensions in the families of functions used to respectively explain the formerly mentioned sets s_0 and s_1, we only consider families of polynomial functions using at most one dimension not in $V_0 \cup V_1$ to determine the complexity of $s_0 \cup s_1$. This heuristic ensures that the algorithmic complexity is linear in the number of irrelevant dimensions during each merge without harming the quality of the results.

[1]https://github.com/AdrienMatricon/cocotte

V. Experiments

In order to demonstrate our algorithm capabilities, we report first experiments where datapoints are generated from variations of two manually defined policies on a robotic arm, and show that our algorithm can discover meaningful skills from this data.

A. Simulation environment

We use the simulation environment from [17], which features a robotic arm fixed to a table on which lays a cubic object. The robotic arm configuration is completely determined by the angles of its 6 joints and the gripper, and is described by 7 real parameters between 0 and 1. In this environment, the arm always starts in the same configuration (see Figure 5), then performs a policy defined by a succession of waypoints.

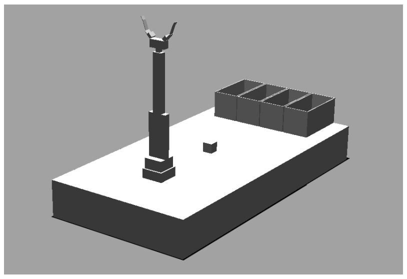

Fig. 5. Initial configuration of the simulation

We consider policies of the form $\pi_a = [q_0, q_1]$, equivalent to policies defined by q_1 from initial configurations given by q_0, so that in the previous notations $\mathbf{c} = q_0$ and $\mathbf{a} = q_1$.

B. Experimental setup

For visualization purposes, we choose to artificially make the underlying structure of the problem 2-dimensional. To that effect we manually determine two distinct behaviours A and B with policies $\pi_a^A = [q_0^A, q_1^A]$, where q_0^A and q_1^A are configurations respectively to the left and right of the cube so that the arm moves to the right towards the cube, and $\pi_a^B = [q_0^B, q_1^B]$, where q_0^B and q_1^B are this time respectively to the right and left of the cube so that the arm moves to the left towards the cube. We then introduce variations of π_a^A by making both the context and action vary:

$$\pi = [q_0^A + t_0 \times (q_1^A - q_0^A),\ q_0^A + t_1 \times (q_1^A - q_0^A)]$$

with $t_0 \in [-1, 0]$ and $t_1 \in [t_0, 1.5]$. We introduce similar variations of π_a^B and gather the data from all simulations, with $\mathbf{R}$ being the final position of the cube, to which we add a uniform noise of amplitude 1cm on each coordinate, and ε being the sum of the estimated amplitudes of the simulation noise and added noise. We randomly perform variations of π_a^A or π_a^B, and gather a training set of 300 datapoints as well as a test set of 10000 datapoints that we will use to visualize what has been learned.

We expect COCOTTE to discover the structure of the data by itself, and to identify discrete parameterized skills such as *pushing the cube to the left* and *pushing the cube to the right*. We expect it to group most variations of π_a^A together, most variations of π_a^B together, and to learn a general model for each parameterized skills. We also expect it to learn models that are at most 2-dimensional, as each variation of π_a^A or π_a^B is by construction entirely determined by (t_0, t_1), and to identify relevant dimensions.

C. Results

After the training, for each datapoint $(\mathbf{R}_p, \mathbf{c}_p, \mathbf{a}_p, \varepsilon_p)$ of the test set, we ask the algorithm to assign $(\mathbf{c}_p, \mathbf{a}_p)$ to a parameterized skill and to predict the final position of the cube $\mathbf{R}_p$. As can be seen on Figure V-C, which shows those predictions for the y-coordinates, four models have been learned. Those are meaningful and correspond to parameterized skills: *pushing the cube on the table to the left* (in green), *pushing the cube on the table to the right* (in orange), *pushing the cube off the table to the right* (in blue), and *not moving the cube* (in red).

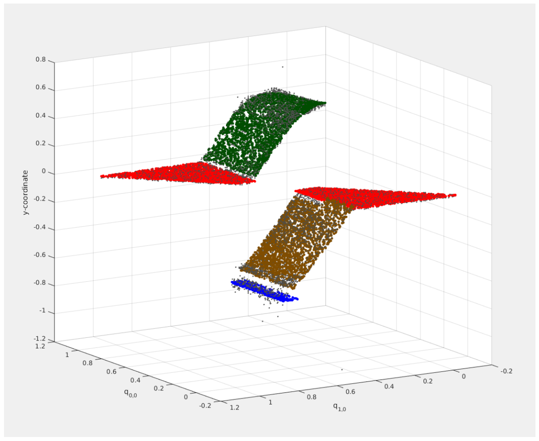

Fig. 6. Predictions of the y-coordinate of $\mathbf{R}$ on the test set. In grey: actual values from the simulation. In blue, green, red, orange: predictions from the 4 models. The green model is of complexity 4 (degree 3, 1 relevant dimension $q_{1,0}$), the orange model of complexity 3 (degree 2, 1 relevant dimension $q_{1,0}$), while the other two models are constant polynomials (complexity 0). See text for more information.

More quantitative results are displayed on Figure 7, which shows the evolution of the total complexity and error rate over the y-coordinate as COCOTTE is run on random subsets of the training set of increasing cardinality. At first, there are few constraints and the overall complexity of models is low. It then increases with the number of points, *i.e.* of constraints, before going down as the density of points becomes high enough for the local structure to emerge and for the constraints to avoid wrong merges (artifacts). It finally converge to the complexity of the model presented in Figure . As for the error-rate, it steadily goes down as more points are added, which seems to confirm that our approach is not prone to overfitting and leads to good generalization properties.

D. Discussion

As argued in the introduction, our approach does not suppose the external definition of parameterized tasks, but is able to discover parameterized skills from data, thus being applicable in a developmental scenario beyond the initial controlled experiments reported in this paper. From a developmental

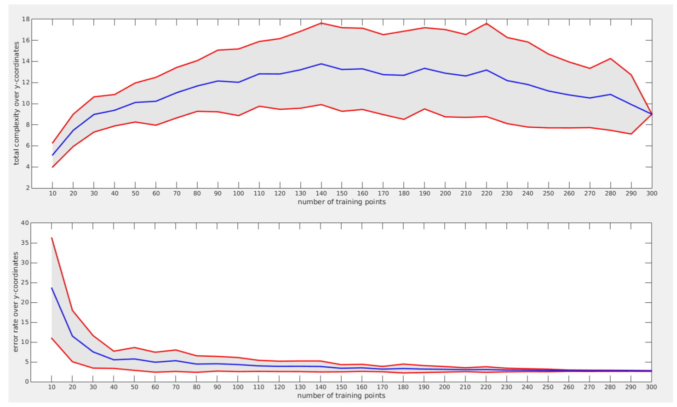

Fig. 7. Evolution of the total complexity and error-rate over the y-coordinates when running COCOTTE over random subsets of the training set of increasing cardinality (showing mean and standard deviation over 80 executions of the algorithm). Above: evolution of the total complexity over the y-coordinates (sum of the one-dimensional approximator complexities over all models). Below: evolution of the error-rate (percentage of points in the test set with prediction error above the precision treshold). See text for more information.

perspective, it would therefore be interesting to have the robot actively gather new datapoints by exploring the parameter space of previously discovered skills so as to check their limits of validity. It would also be interesting to associate this procedure with motor and goal babbling in order to explore the space of possible effects and discover new skills.

In the experiments we performed, our algorithm managed to discover parameterized skills and to learn an explicit model for each of them. Those parameterized skills seem meaningful and correspond to parameterized skills that humans could intuitively identify, like *pushing the cube off the table to the right*. The collection of models that has been learned constitutes a repertoire of discrete parameterized skills, which can be used for planning. Although the robotic agent was made to explore variations of two externally given policies π_a^A and π_a^B, one must note that the parameterized skills that were discovered do not directly reflect those policies: the parameterized skill *not moving the cube* generalizes instances of both π_a^A and π_a^B, while the other instances of π_a^A lead to the two distinct parameterized skills *pushing the cube on the table to the right* and *pushing the cube off the table to the right*.

The merging phase of our algorithm is performed in a greedy manner, selecting the pair of closest sets first as candidate for merging. The reason for this is to have local structures emerge first, especially in the areas dense with datapoints. Local structure manifests itself as a decrease in complexity: when merging the sets s_0 and s_1, local structure causes $\text{comp}(s_0 \cup s_1) < \text{comp}(s_0) + \text{comp}(s_1)$. This decrease in complexity makes it harder to merge $s_0 \cup s_1$ with any other set s_2, as it needs to verify:

$$\begin{aligned}\text{comp}(s_0 \cup s_1 \cup s_2) &\leq \text{comp}(s_0 \cup s_1) + \text{comp}(s_2)\\ &< \text{comp}(s_0) + \text{comp}(s_1) + \text{comp}(s_2)\end{aligned}$$

We therefore constrain the models we learn to comply with the local structures, which makes it much harder for wrong merges to happen. In practice, as soon as a certain density of datapoints is reached, the only wrong merges we see happening during experiments are those at the frontiers between two smooth components of F, where the merges do not reflect a local structure and do not reduce the complexity. The resulting sets are then removed during the artifact elimination phase of our algorithm, and their datapoints distributed among other sets.

The main assumption we make in this work is that of the existence of a known deterministic bound ε on the stochastic noise function Δ, an assumption that we use to define the ability of a function to explain a set of datapoints, which is at the core of COCOTTE. This assumption strongly contrasts with the usual hypothesis of a Gaussian noise, and may seem like a strong assumption because it does not allow for any outlier. Let us first note that if there are indeed no outliers, our assumption is actually much weaker than the hypothesis of a Gaussian noise, because it does not assume any distribution for Δ at all. This being said, the reason why we allow ourselves to make this assumption is that we have other ways we could use to deal with outliers. For example, adding outliers to a set of datapoints should cause a jump in complexity, because they are inconsistent with the inliers. If they are relatively low in number, they should stay within small sets during the merge phase, and interfere very little with the emergence of the good models. We should therefore be able to devise a filtering method to discard them after the merge phase, similarly to what we did for artifact elimination.

An easily overlooked aspect of our iterative algorithm is that it does not involve hand-tuning any parameter or hyperparameter. We do rely on estimates of ε in the data we use as input for our algorithm, which does behaves like a parameter if chosen by hand (too low and some sets will not be merged, too high and the models will underfit the data), but those estimates have a real-world meaning and are not meant to be hand-tuned. For example, if $\mathbf{R}_i$ is a value measured by a sensor, the corresponding ε_i should be the precision on that sensor given by the manufacturer.

VI. Conclusion

We have presented a method to discover and learn explicit models for parameterized skills. The key idea is that parameterized skills can be seen as smooth components of a locally smooth global model of the robotic agent's capabilities. Framing the problem as a constrained complexity minimization, we proposed an iterative algorithm to solve it and reported promising results, discovering meaningful parameterized skills within a robotic agent's experiences. In future work, we intend to apply our algorithm in real developmental robotics conditions, where the robotic agent gathers data by itself through motor and goal babbling.

Acknowledgements

This work is supported by the DREAM project[2] through the European Unions Horizon 2020 research and innovation program under grant agreement No 640891.

[2]http://www.robotsthatdream.eu

References

[1] R. S. Sutton, D. Precup, and S. P. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, no. 1-2, pp. 181–211, 1999. [Online]. Available: http://dx.doi.org/10.1016/S0004-3702(99)00052-1

[2] J. Kober, A. Wilhelm, E. Oztop, and J. Peters, "Reinforcement learning to adjust parametrized motor primitives to new situations," *Auton. Robots*, vol. 33, no. 4, pp. 361–379, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10514-012-9290-3

[3] M. P. Deisenroth, P. Englert, J. Peters, and D. Fox, "Multi-task policy search," *CoRR*, vol. abs/1307.0813, 2013. [Online]. Available: http://arxiv.org/abs/1307.0813

[4] F. Stulp, G. Raiola, A. Hoarau, S. Ivaldi, and O. Sigaud, "Learning compact parameterized skills with a single regression," in *IEEE-RAS International Conference on Humanoid Robots*, 2013.

[5] B. C. da Silva, G. Konidaris, and A. G. Barto, "Learning parameterized skills," in *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012. [Online]. Available: http://icml.cc/discuss/2012/826.html

[6] M. Asada, K. Hosoda, Y. Kuniyoshi, H. Ishiguro, T. Inui, Y. Yoshikawa, M. Ogino, and C. Yoshida, "Cognitive developmental robotics: A survey," *IEEE Trans. Autonomous Mental Development*, vol. 1, no. 1, pp. 12–34, 2009. [Online]. Available: http://dx.doi.org/10.1109/TAMD.2009.2021702

[7] A. Ghadirzadeh, J. Bütepage, D. Kragic, and M. Björkman, "Self-learning and adaptation in a sensorimotor framework," in *2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016*, D. Kragic, A. Bicchi, and A. D. Luca, Eds. IEEE, 2016, pp. 551–558. [Online]. Available: http://dx.doi.org/10.1109/ICRA.2016.7487178

[8] C. G. Atkeson and S. Schaal, "Memory-based neural networks for robot learning," *Neurocomputing*, vol. 9, no. 3, pp. 243–269, 1995. [Online]. Available: http://dx.doi.org/10.1016/0925-2312(95)00033-6

[9] S. Vijayakumar and S. Schaal, "Locally weighted projection regression: Incremental real time learning in high dimensional space," in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, P. Langley, Ed. Morgan Kaufmann, 2000, pp. 1079–1086.

[10] C. Saunders, A. Gammerman, and V. Vovk, "Ridge regression learning algorithm in dual variables," in *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconsin, USA, July 24-27, 1998*, J. W. Shavlik, Ed. Morgan Kaufmann, 1998, pp. 515–521.

[11] F. Stulp and O. Sigaud, "Many regression algorithms, one unified model: A review," *Neural Networks*, vol. 69, pp. 60–79, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2015.05.005

[12] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Computation*, vol. 3, no. 1, pp. 79–87, 1991. [Online]. Available: http://dx.doi.org/10.1162/neco.1991.3.1.79

[13] P. Oudeyer, F. Kaplan, and V. V. Hafner, "Intrinsic motivation systems for autonomous mental development," *IEEE Trans. Evolutionary Computation*, vol. 11, no. 2, pp. 265–286, 2007. [Online]. Available: http://dx.doi.org/10.1109/TEVC.2006.890271

[14] C. E. Shannon, "Coding theorems for a discrete source with a fidelity criterion," *IRE Nat. Conv. Rec*, vol. 4, no. 142-163, p. 1, 1959.

[15] A. Lemme, R. F. Reinhart, and J. J. Steil, "Self-supervised bootstrapping of a movement primitive library from complex trajectories," in *14th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2014, Madrid, Spain, November 18-20, 2014*. IEEE, 2014, pp. 726–732. [Online]. Available: http://dx.doi.org/10.1109/HUMANOIDS.2014.7041443

[16] V. N. Vapnik, "The nature of statistical learning theory," 1999.

[17] P. Ecarlat, A. Cully, C. Maestre, and S. Doncieux, "Learning a high diversity of object manipulations though an evolutionary-based babbling," in *Proceedings of the workshop Learning Object Affordances, IROS 2015*, Hambourg, 2015, pp. 1–2.