Modular Active Curiosity-Driven Discovery of Tool Use*

Sébastien Forestier[1] and Pierre-Yves Oudeyer[2]

Abstract— **This article studies algorithms used by a learner to explore high-dimensional structured sensorimotor spaces, such as in tool use discovery. In particular, we consider goal babbling architectures that were designed to explore and learn solutions to fields of sensorimotor problems, i.e. to acquire inverse models mapping a space of parameterized sensorimotor problems/effects to a corresponding space of parameterized motor primitives. However, so far these architectures have not been used in high-dimensional spaces of effects. Here, we show the limits of existing goal babbling architectures for efficient exploration in such spaces, and introduce a novel exploration architecture called Model Babbling (MB). MB exploits efficiently a modular representation of the space of parameterized problems/effects. We also study an active version of Model Babbling (the MACOB architecture). These architectures are compared in a simulated experimental setup with an arm that can discover and learn how to move objects using two tools with different properties, embedding structured high-dimensional continuous motor and sensory spaces.**

I. INTRODUCTION

A major challenge in robotics is to learn sensorimotor models in high-dimensional continuous motor and perceptual spaces. Of particular interest is the acquisition of inverse models which map a space of sensorimotor problems to a space of motor programs that solve them. For example, this could be a robot learning which movements of the arm and hand can push or throw an object in each of several target locations, or which arm movements allow to produce which displacements of several objects potentially interacting with each other (e.g. in the case of tool use). Specifically, acquiring such repertoires of skills through incremental exploration of the environment has been argued to be a key target for life-long developmental learning [1], [2], [3].

To approach this challenge, various works have considered the parameterization of these motor and problem spaces. For example, motor programs can be encoded through Dynamical Movement Primitives parameterized by a vector of real numbers [4]. Similarly, it is possible to embed targeted sensorimotor problems (also called space of effects or task space) in a dual parameterized space such as the coordinates of the target object location [5], [6], [7], potentially combined with parameters characterizing the position of obstacles [4].

This dual parameterization is useful for several reasons. First, given a database of experiences associating parameters of motor programs to a set of sensorimotor problems they solve (e.g. the effects they produce), it is possible to use optimization and regression techniques to infer the parameters of motor programs that solve new sensorimotor problems which parameters were not encountered during training [4], [5], [6], [8], [7]. Second, it allows efficient data collection leveraging the interactions among sensorimotor problems as achieved in goal babbling exploration [7], [9], [10] and other related approaches [8]: when the learner is searching for parameters optimizing one sensorimotor problem (typically using policy search or related stochastic optimization methods [11]), it will often discover parameters that are improving other sensorimotor problems - and update their current best solutions accordingly [7].

Next to approaches that have considered finite sets of parameterized problems [4], [12], other approaches [7], [9], [13], [8], [10] have considered the challenge of autonomous exploration and learning of continuous fields of parameterized problems (e.g. discovering and learning all the feasible displacements of objects and their motor solutions). Among this latter approaches, the technique of goal babbling [7], [9], [10] (which can be made active [7], [10]) was shown to be highly efficient for complex tasks such as learning to throw an object in all direction with a flexible fishing rod [14], learning omnidirectional legged locomotion on slipping surfaces [7] or learning to control a robotic pneumatic elephant trunk [9].

However, to our knowledge, results of goal babbling approaches (as well as results of other approaches to learning inverse models) were so far achieved in relatively low-dimensional spaces of parameterized problems. Furthermore, they were also experimented in sensorimotor spaces with little structure, and in particular have not yet been applied to sensorimotor problems involving tool use.

In this article, the primary question we address is: Can goal babbling approaches efficiently drive exploration in high-dimensional structured sensorimotor spaces, such as in tool use discovery? As we will show, applying them as they exist does not allow an efficient exploration of the sensorimotor space. Rather, we will present a novel algorithmic architecture for exploration, called **Model Babbling**, that drives sensorimotor data collection by considering a modular representation of the sensorimotor space: instead of considering a flat architecture mapping a motor space to a single high-dimensional space of effects, it considers a set of submodels mapping the motor space to various subspaces of the space of effects. When selected, each of these submodels is explored using the goal babbling approach, and the architecture leverages the fact that exploring one submodel produces data that can improve other submodels.

*This work was not supported by any organization

[1]Sébastien Forestier is with Inria Bordeaux Sud-Ouest and Université Bordeaux, 200 Avenue de la Vieille Tour, 33405 Talence, France sebastien.forestier@inria.fr

[2]Pierre-Yves Oudeyer is with Inria Bordeaux Sud-Ouest and Ensta ParisTech, 200 Avenue de la Vieille Tour, 33405 Talence, France pierre-yves.oudeyer@inria.fr

A secondary issue we study is whether active learning methods can improve the efficiency of this Model Babbling approach. In particular, we present the **M**odular **A**ctive **C**uriosity-driven m**O**del **B**abbling (**MACOB**) architecture, where a measure of empirical learning progress is used by a multi-armed bandit algorithm to select which model to explore [7], [14], [2], [10].

The study we present is instantiated in a simulated experimental setup with an arm that can discover and learn how to move objects using two tools with different properties. Compared to other work that have studied autonomous tool use learning [15], [16], [17], [18], this study is original in that it combines 1) considering the problem of how to design efficient exploration algorithms (rather than how to design efficient exploitation algorithms that can build compact models from the data collected through exploration); 2) considering a problem of tool use discovery where tools are objects with initially no special status with respect to other objects (i.e. the robot does not know they are "tools").

Together with this paper we provide open-source Python code[1] with Jupyter notebooks that show how to reproduce the experiments and analysis.

II. Methods

A. Environment

We designed a robotic setup where a 2D simulated arm can grasp two sticks that can be used to move out-of-reach objects, and can't control other objects neither with the hand nor the sticks. The different items of the environment and their interactions are precisely described in the next sections. See Fig.1 for a possible state of the environment.

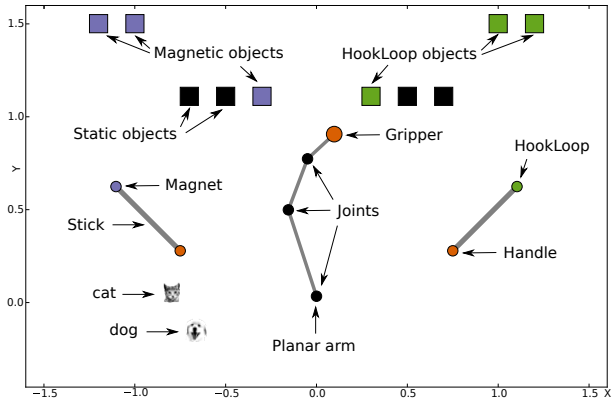


Fig. 1. A possible state of the environment.

1) Robotic Arm: The 2D robotic arm has 3 joints plus a gripper located at the end of the arm. Each joint can rotate from $-\pi\ rad$ to $\pi\ rad$ around its resting position, mapped to a standard interval of $[-1, 1]$. The length of the 3 segments of the arm are 0.5, 0.3 and 0.2 so the length of the arm is 1 unit. The resting position of the arm is vertical with joints at $0\ rad$ and its base is fixed at position $(0, 0)$. The gripper g has 2 possible positions: *open* ($g \geq 0$) and *closed* ($g < 0$) and its resting position is *open* (with $g = 0$). The robotic arm has 4 degrees of freedom represented by a vector in $[-1, 1]^4$. A trajectory of the arm will be represented as a sequence of such vectors.

2) Motor Control: We use Dynamical Movement Primitives [19] to control the arm's movement as this framework allows the production of a diversity of arm's trajectories with only few parameters. Each of the 4 arm's degrees-of-freedom (DOF) is controlled by a DMP starting at the rest position of the joint. Each DMP is parameterized by one weight on each of 2 basis functions and one weight specifying the end position of the movement. The weights are bounded in the interval $[-1, 1]$ and allow each joint to fairly cover the interval $[-1, 1]$ during the movement. Each DMP outputs a series of 50 positions that represents a sampling of the trajectory of one joint during the movement. The arm's movement is thus parameterized with 12 weights, represented by the motor space $M = [-1, 1]^{12}$.

3) Objects and Tools: Two sticks can be grasped by the handle side (orange side) in order to catch an out-of-reach object. The sticks have length 0.5 and are located at positions $(-0.75, 0.25)$ and $(0.75, 0.25)$ as in Fig. 1. One stick has a magnet on the end and can catch magnetic objects (represented in blue), and the other stick has a hook-and-loop tape to catch another type of objects (objects represented in green). If the gripper is closed near the handle of one stick (closer than 0.25), this stick is considered grasped and follows the gripper's position and the orientation of the arm's last segment until the gripper opens. In one condition, we will use environmental noise as a gaussian noise of standard deviation 0.1 added to the (normally equal to 0) angle between the stick and the arm's last segment, different at each of the 50 movement's steps. If the other side of one stick reaches (within 0.25) a matching object (magnetic or hook-and-loop), the object will then follow the end of the stick. Three magnetic objects are located at positions $(-0.3, 1.1)$, $(-1.2, 1.5)$ and $(-1., 1.5)$, so that only one is reachable with the corresponding stick. Three hook-and-loop objects are located at positions $(0.3, 1.1)$, $(1., 1.5)$ and $(1.2, 1.5)$, so that only one is reachable with the corresponding stick. Also, two animals (one cat and one dog) have initial position $(-0.1, 1.1)$ and $(0.1, 1.1)$. They walk randomly following a gaussian noise of standard deviation 0.01 on X and Y dimensions updated at each of the 50 steps of a trial. Finally, four static objects (black squares) have also no interaction with other objects.

4) Sensory Feedback: At the end of the movement, the robot gets sensory feedback representing the trajectory of the different items of the environment during the arm's movement. This feedback is composed by the position of each item at 3 time points: at steps 17, 33, and 50 during the movement of 50 steps. First, the trajectory of the gripper is represented as a sequence of X and Y positions and the aperture (1 or -1) of the gripper (S_{Hand}, 9D). Similarly,

[1]Source code and notebooks are available as a Github repository at `https://github.com/sebastien-forestier/IROS2016`

the trajectories of the end points of the sticks are sequences of X and Y positions (S_{Stick_1} and S_{Stick_2}, 6D each). Also, the trajectory of each object is a sequence of X and Y positions: S_{Object} with $Object \in \{ Magnetic_1, Magnetic_2, Magnetic_3, HookLoop_1, HookLoop_2, HookLoop_3, Cat, Dog, Static_1, Static_2, Static_3, Static_4\}$. Those spaces are all in 6 dimensions, in the interval $[-1.5, 1.5]^6$. The total sensory space S has 93 dimensions.

B. Training procedure

The training procedure is composed of 100000 iterations, which is large enough so that with different architectures, reachable and learnable objects' sensory spaces are significantly explored, but not too large so that in all conditions exploration still finds significantly new effects. At each iteration, the agent executes a motor trajectory and gets the associated sensory feedback. The arm, tools and other objects are resetted to their initial state at the end of each iteration. The problem settings for the agent is to explore its sensorimotor space and collect data so as to generate a diversity of effects and that this collected database of learning exemplars can be used to build inverse models to be able to reproduce those effects. The total exploration of different sub-spaces, and competence to reach given goals in these spaces will be assessed.

An agent is described as two independent components: an exploration algorithm and an exploitation algorithm (see Fig. 2). The exploration algorithm decides at each iteration which motor command to explore, and gathers sensory feedback to build a database of sensorimotor experiences. As detailed below, the exploration algorithm can make use of the current database of sensorimotor experiences to define a coarse but fast surrogate inverse model to orient the exploration process. On the other hand, the exploitation algorithm uses the database built during exploration to generate a potentially more precise inverse model, i.e. to find motor commands to reach sensory goals given by the experimenter based on the explored data. The inverse model of the exploitation algorithm can be built during exploration as an incremental and asynchronous process, or built at the end of exploration as a batch process. Exploration and exploitation architectures are described in the next sections.

C. Exploration Architectures

1) Random Motor Babbling: We first define a control architecture where the agent always chooses random motor commands to try in the environment (RmB, see Algo. 1).

Algorithm 1 Random Motor Babbling (RmB)

Require: Motor space M, Sensory space S

```
database ← VoidDatabase(dim(M), dim(S))
loop
    m ← RandomMotor(M)
    s ← Environment(m)
    Add(database, (m, s))
return database
```

2) Random Goal Babbling: In the other exploration architectures the agent performs Goal Babbling. With this method, it self-generates goals in the sensory space and tries to reach them but adding some exploration noise to its motor commands to discover new effects. To generate those goals, different strategies have been studied [7]. It was shown that estimating the learning progress in different regions of the sensory space and generating the goals where the progress is high leads to a fast learning. However, this cannot be applied in a 93D sensory space as a learning progress signal cannot be efficiently estimated in this volume.

Consequently, we use random goal babbling: goals are randomly generated in the sensory space. This method was nevertheless proven to be highly efficient in complex sensorimotor spaces [9]. To perform goal babbling, the agent uses a sensorimotor model that learns a mapping between M and S and provide the inverse inference of a probable motor command m to reach a given a sensory goal s_g (see Algo. 2 and 3). The sensorimotor model stores sensorimotor information of the form $(m + \epsilon, s)$ with $m + \epsilon$ being the inferred motor parameters to reach the sensory goal, plus gaussian exploration noise (of standard deviation $\epsilon = 0.01$), and $s \in S$ the associated sensory feedback in a sensorimotor database. Section II-C.5 explains in more details two different algorithms that will be used to implement inverse models. We use the Explauto autonomous exploration library [20] to implement the sensorimotor models and goal babbling. In our implementation, the agent first begins by exploring random motor commands to bootstrap the sensorimotor model until at least 2 distinct sensory points have been reached, and then it starts goal babbling.

Algorithm 2 Random Goal Babbling Step

Require: Sensorimotor model `sm_model`

```
s_g ← RandomGoal(S)
m ← Inverse(sm_model, s_g)
ε ← Gaussian(μ = 0, σ = 0.01)
s ← Environment(m + ε)
Update(sm_model, (m + ε, s))
return (m + ε, s)
```

Algorithm 3 Random Goal Babbling Experiment

Require: Motor space M, Sensory space S

```
database ← VoidDatabase(dim(M), dim(S))
sm_model ← InitializeSensorimotorModel(M, S)
loop
    (m, s) ← RandomGoalBabblingStep(sm_model)
    Add(database, (m, s))
return database
```

3) Model Babbling: We call *flat* exploration architecture the random goal babbling strategy applied to explore directly a mapping between the motor space M (12D) and the sensory space S (93D). However, the 93D sensory space can be separated into several sub-spaces to reflect the different

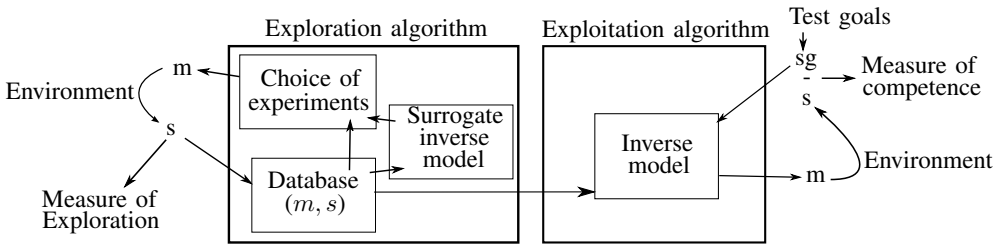


Fig. 2. Agent's two components: the exploration and exploitation algorithms.

items of the environment. We define a factored version of the sensorimotor space as 15 sub-spaces reflecting the structure of the environment: S_{Hand} is the sensory space representing the trajectory of the gripper, S_{Stick_1} and S_{Stick_2} the trajectory of the end points of the sticks, and one sensory space for each other object. We thus define a modular architecture to explore 15 sensorimotor models at the same time (one for each sensory sub-space). Each of those modules functions in the same way as a random goal babbling flat architecture, with M as motor space but a specific sensory space (e.g. module 2 has S_{Stick_1} as sensory space). Nevertheless, at each iteration the modular architecture first has to choose the model in which to pick a random goal. We call this procedure Model Babbling. Once a model is chosen, it generates a random goal in its sensory space, infer motor parameters to reach that goal, and adds the same exploration noise as in the flat architectures. Finally, when motor parameters m have been tested in the environment and the 93D feedback s received, the sensorimotor mappings of all modules are updated with their respective dimensions of s. For instance, module 1 is updated with m and the 9 dimensions of s from 0 to 8, and module 2 is updated with m and the 6 dimensions of s from 9 to 15.

In a first condition, we introduce a random choice of babbling module, which we call Random Model Babbling (See Algo. 4, RMB).

Algorithm 4 Modular - Random Model Babbling

Require: Motor space M
Require: Sensory spaces S_i for $i \in \{1..15\}$
`database` $\leftarrow$ VoidDatabase($dim(M), dim(S)$)
for $i \in \{1..15\}$ **do**
  `sm_model_i` $\leftarrow$ SMModel(M, S_i)
loop
  `mod_i` $\leftarrow$ RandomModule($\{1..15\}$)
  $(m, s) \leftarrow$ RandomGoalBabblingStep(`sm_model_i`)
  for $j \in \{1..15\}$ **do**
    Update(`sm_model_j`, (m, Projection(s, S_j)))
  Add(`database`, (m, s))
return `database`

In strategic learning, different parameterized problems and strategies to solve them are available and the agent learns which strategies are useful for which problems. In [21], the authors show that an active choice of the outcomes and strategies based on the learning progress on each of them increases learning efficiency compared to a random choice. Section II-C.4 gives more details on how the measure of learning progress on each module is computed and used to actively choose the babbling module. In [7], the authors develop the SAGG-RIAC architecture of algorithms where the sensory space is automatically splitted into regions where the learning progress is monitored, and goals are generated in regions where the progress is high. Here, instead of differentiating the learning progress in different regions of a single space, we differentiate it in different sensory spaces.

Algorithm 5 Modular - Active Model Babbling

Require: Motor space M
Require: Sensory spaces S_i for $i \in \{1..15\}$
`database` $\leftarrow$ VoidDatabase($dim(M), dim(S)$)
for $i \in \{1..15\}$ **do**
  `sm_model_i` $\leftarrow$ SMModel(M, S_i)
  `i_model_i` $\leftarrow$ IModel(S_i)
  $I_{mod_i} \leftarrow 0$
loop
  $i \leftarrow$ ChooseModule(I_{mod_i} for $i \in \{1..15\}$)
  $(m, s) \leftarrow$ RandomGoalBabblingStep(`sm_model_i`)
  $I_{mod_i} \leftarrow$ UpdateInterestModel(`i_model_i`, s_g, Projection(s, S_i))
  for $j \in \{1..15\}$ **do**
    Update(`sm_model_j`, (m, Projection(s, S_j)))
  Add(`database`, (m, s))
return `database`

4) Active Model Babbling: To implement an active choice of model to explore (Active Model Babbling), we first define a measure of interest based on the learning progress on each of the 15 models. When a module has been chosen to babble, it draws a random goal s_g, and finds motor parameters m to reach this goal. The actual reached outcome s in the sensory sub-space of the model, associated to m might be very different from s_g. We define a distance D_{S_i} between two points s and s' in a sensory sub-space S_i as the L_2 distance divided by the maximal distance in this sensory space, in order to scale this measure across the different spaces:

$$D_{S_i}(s, s') = \frac{||s - s'||}{max_{s_1,s_2}||s_1 - s_2||} \quad (1)$$

We define the interest $I(s_g)$ associated to the goal $s_g \in S_i$:

$$I(s_g) = D_{S_i}(s'_g, s') - D_{S_i}(s_g, s) \quad (2)$$

where s_g and s are the current goal and reached sensory points, and s'_g and s' are the previous goal of that module that is the closest to s_g, and its associated reached sensory point. The interest of a module is initialized at 0 and updated to follow the progress of its goals (with rate $n = 1000$):

$$I_{mod}(t) = \frac{n-1}{n}\ I_{mod}(t-1) + \frac{1}{n}\ I(s_g) \quad (3)$$

where t is the current iteration: $t \in [1..100000]$.

Algorithm 6 Update Interest Model

Require: Interest model `i_model`
Require: Sensory goal s_g, outcome s
$(s'_g, s') \leftarrow$ NearestNeighbor(`goal_database`, s_g)
$I(s_g) = |D_{S_i}(s'_g, s') - D_{S_i}(s_g, s)|$
$I_{mod_i} \leftarrow \frac{n-1}{n} I_{mod_i} + \frac{1}{n}\ I(s_g)$
Add(`goal_database`, (s_g, s))
return I_{mod_i}

Finally, we implement a multi-armed bandit algorithm to choose the babbling module at each iteration [7], [10]. The choice of module is probabilistic and proportional to their interest, with $\epsilon = 10\%$ of random choice to allow an exploration/exploitation tradeoff. We call MACOB those modular active exploration architectures (Modular Active Curiosity-driven mOdel Babbling).

5) Sensorimotor models: Here we describe two algorithms to provide fast, incremental and online forward and inverse model based on a sensorimotor database of motor commands and associated sensory feedback. The first algorithm is the Nearest Neighbor (NN) algorithm, which finds the nearest neighbor of a given point in a database based on a kd-tree search. The forward model is implemented by the following: given a motor command m, the NN algorithm finds the nearest motor command m' in the motor part of the database, and return the sensory point associated to m'. Also, the inverse of a sensory position s is computed as the motor part m' of the nearest neighbor s' of s in the sensory part of the sensorimotor database (see Algo. 7).

Algorithm 7 NN Sensorimotor Model

function INITIALIZE(M, S)
  `sm_database` $\leftarrow$ VoidDatabase($dim(M), dim(S)$)
function UPDATE((m, s))
  Add(`sm_database`, (m, s))
function FORWARD(m)
  $(m', s') \leftarrow$ NearestNeighbor(`sm_database`, m)
  return s'
function INVERSE(s_g)
  $(m', s') \leftarrow$ NearestNeighbor(`sm_database`, s_g)
  return m'

The second algorithm allows to interpolate and extrapolate the forward model around explored points with the Locally Weighted Linear Regression (LWLR, [22]). Given a motor command m, LWLR computes a linear regression of the forward model based on the $k = 10$ nearest neighbors of m in the motor part of the database, weighted locally. The weights of the k nearest neighbors of m depends on the distance to m with a gaussian decreasing function of standard deviation $\sigma = 0.1$, and LWLR then computes the prediction s_p of m with this local regression (see Algo. 8). On the other hand, the inverse m^* of a sensory goal s_g is found by the minimization of the predicted distance between the reached and goal sensory points as the error function $e(m) = ||Forward(m) - s_g||^2$ with an optimization algorithm (we use the L-BFGS-B algorithm [23]). We limit the number of forward model evaluations (which uses LWLR) to 200.

6) Summary of Exploration Architectures:

- RmB: Random motor Babbling control (Algo. 1),
- F-NN-RGB: Flat, Nearest Neighbor forward and inverse models, Random Goal Babbling (Algo. 2, 3, 7),
- F-LWLR-RGB: Flat, Locally Weighted Linear Regression forward model and optimization-based inverse model, Random Goal Babbling (Algo. 2, 3, 8),
- M-NN-RMB: Modular, Nearest Neighbor forward and inverse models, Random Model Babbling (Algo. 2, 3, 4, 7),
- M-NN-AMB: Modular, Nearest Neighbor forward and inverse models, Learning Progress based Active Model Babbling (Algo. 2, 3, 5, 7),
- M-LWLR-RMB: Modular, Locally Weighted Linear Regression forward model and optimization-based inverse model, Random Model Babbling (Algo. 2, 3, 4, 8),
- M-LWLR-AMB: Modular, Locally Weighted Linear Regression forward model and optimization-based inverse model, Active Model Babbling (Algo. 2, 3, 5, 8).

The database of associations (m, s) can be used by any exploitation architecture to build an inverse model of the environment. The next section describes the exploitation architectures that are compared in the results.

Algorithm 8 LWLR-BFGS Sensorimotor Model

function INITIALIZE(M, S)
  `sm_database` $\leftarrow$ VoidDatabase($dim(M), dim(S)$)
function UPDATE((m, s))
  Add(`sm_database`, (m, s))
function FORWARD(m)
  `knns` $\leftarrow$ KNearestNeighbors(`sm_database`, m)
  `weights` $\leftarrow$ GaussianWeights(Distance(`knns`, m))
  `R` $\leftarrow$ ComputeLWLRegression(`knns`, `weights`)
  $s_p \leftarrow$ `R`(m)
  return s_p
function INVERSE(s_g)
  `error`$(m) = ||$FORWARD$(m) - s_g||^2$
  $m^* \leftarrow$ L-BFGS-B-Minimize(`error`)
  return m^*

D. Exploitation Architectures

In this paper, exploitation architectures are used to evaluate the quality of the exploration database generated by the different exploration architectures. An exploitation architecture generates an inverse model of the environment based on a database of previously explored motor commands and their associated sensory feedback, to sensory goals given by the experimenter. We are also interested in comparing the inverse models built by different combinations of exploration database and exploitation architectures. We evaluate the accuracy of resulting inverse model to reach points in two spaces of interest, $S_{Magnetic_1}$ and $S_{HookLoop_1}$. Indeed, those spaces represent the only objects that can be moved by one of the sticks as they are not static and not out-of-reach. One set of goals is randomly drawn in the 2D subspace corresponding to the final position of each of the two interesting objects (1000 goals in each).

We define two exploitation architectures generating inverse models: one based on the Nearest Neighbor algorithm (NN, Algo. 7), and one based on the Locally Weighted Linear Regression forward model and an optimization-based inverse model (LWLR, Algo. 8).

Given a goal s_g (e.g. $s_g = (0.5, 0.5)$, as the final position of the reachable magnetic object), the NN algorithm looks into the explored database, finds the nearest sensory reached point s along the dimensions of the target effect, and returns its associated motor command m. On the other hand, the LWLR algorithm builds a forward model based on a locally weighted linear regression, and an optimization algorithm (L-BFGS-B) finds the motor command m that minimizes the predicted distance between the 2D reached and goal sensory points.

III. RESULTS

We run 100 trials of 100000 iterations with environmental noise and 100 trials without noise, for each of the 7 exploration architectures. We measure the total exploration of 6D spaces of interest $S_{Magnetic_1}$ and $S_{HookLoop_1}$ after 100000 iterations, and provide results depending on the exploration architecture and environmental noise on the orientations of the sticks. Then, for each of the 1400 explored databases, we test the inverse models generated by the two exploitation architectures in the 2D sub-spaces of the final position of the two objects of interests, with the same 1000 random goals for each space. We chose those 2D spaces to allow the visualization of a projection of the learnt skills (as in Fig. 3), but the actual skills are high-dimensional (9D for the hand, 6D for each tool and object). We provide a measure of competence of each combination of exploration and exploitation architectures as the median of the distance between the goals and reached sensory points, when environmental noise was present and when the environment was deterministic.

A. Exploration

1) Examples of object exploration: Figure 3 shows qualitatively the exploration of the two reachable and movable objects (corresponding to sensory spaces $S_{Magnetic_1}$ and $S_{HookLoop_1}$) for one trial of some exploration architectures, without environmental noise. The blue points are the 2D end positions of the reachable magnetic object, and green points are the end positions of the reachable hook-and-loop object, for the 100000 iterations of an exploration trial. First, random motor babbling exploration architecture managed to grab the sticks to move one of the object only for a small proportion of the 100000 iterations. Also, only the modular exploration architectures could explore a large proportion of the 2D spaces.

2) Evolution of interests in active model babbling: Figure 4 shows one example of the evolution of the interest of some of the 15 modules of exploration architecture M-NN-AMB. The first module to make progress is the module learning to move the hand, and its exploration finds the magnetic stick and thus allows the module corresponding to this stick to make more progress (at around iteration 10000), which exploration finally allows the discovery that this stick can be used to move one of the magnetic objects and make progress on that task (at around iteration 20000). Notably, modules corresponding to unreachable or static objects have an interest strictly equal to 0.

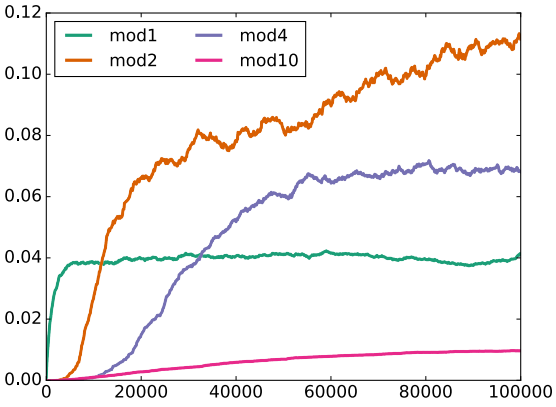
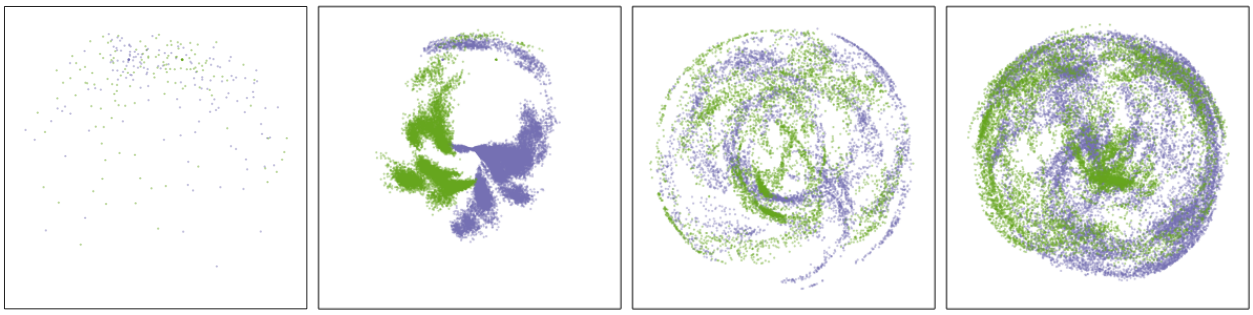


Fig. 4. Interest of some modules along the 100000 iterations, with exploration architecture M-NN-AMB. Module 1 corresponds to the sensory space of the hand. Module 2: magnetic stick. Module 4: reachable magnetic object. Module 10: cat.

3) Exploration measure: The total exploration is measured in $S_{Magnetic_1}$ and $S_{HookLoop_1}$ as the number of cells reached in a discretized grid of 10^6 cells (10 cells on each of the 6 dimensions). For each exploration architecture, we provide in Table I the median, extrema and quartiles of the number of reached cells (median on 2 spaces times 100 trials). In the following, we give results of non-parametric statistical Mann-Whitney U tests for pairs of conditions.

First of all, the comparison of any of the flat exploration architectures (using NN or LWLR, with or without environmental noise) with any of the modular exploration architectures shows that flat architectures have explored less than modular architectures ($p < 0.05$). The effect is small for example if we compare condition F-LWLR-RGB with environmental noise (median 387 reached cells) with con-



(a) RmB (b) F-NN-RGB (c) M-NN-RMB (d) M-NN-AMB

Fig. 3. Position of the two reachable and movable objects at the end of each of the 100000 iterations, for one trial of each exploration architecture. Blue points: position of reachable magnetic object. Red points: reachable hook-and-loop object.

TABLE I
EXPLORATION OF SPACES OF INTEREST

Exploration architectures	Env. Noise	Min	Q1	Median	Q3	Max
RmB	No	57	67	73	78	93
	Yes	62	75	80	85	100
F-NN-RGB	No	1	1	14	89	380
	Yes	1	1	16	116	746
F-LWLR-RGB	No	98	203	245	294	442
	Yes	182	319	387	486	818
M-NN-RMB	No	285	374	415	456	682
	Yes	356	455	508	563	763
M-NN-AMB	No	88	452	536	668	1380
	Yes	156	431	517	721	1453
M-LWLR-RMB	No	368	512	555	607	801
	Yes	449	574	623	691	906
M-LWLR-AMB	No	456	743	870	1046	1440
	Yes	522	811	987	1153	1752

dition M-NN-RMB without noise (median 415). However, the difference is large between this flat architecture and the best exploring modular architecture, M-LWLR-AMB with environmental noise (median 987).

Secondly, the comparison of the conditions where only the model babbling choice differs shows that without environmental noise, active model babbling increases exploration with respect to random model babbling. Indeed, architecture M-NN-RMB has explored less (median 415) than architecture M-NN-AMB (median 536, $p < 10^{-23}$), and architecture M-LWLR-RMB also has explored less (median 555) than architecture M-LWLR-AMB (median 870, $p < 10^{-55}$). If we consider environmental noise, random model babbling architecture using LWLR (median 623) explored less than the active one (median 987, $p < 10^{-39}$).

B. Exploitation

The quality of the different inverse models is assessed at the end of the 100000 exploration iterations, by giving random goals in $S_{Magnetic_1}$ and $S_{HookLoop_1}$, executing the motor commands given by the inverse model (without environmental noise), and measuring the distance between goals and actually reached sensory points. We draw 1000 random sensory goals in each of two spaces of interest, $S_{Magnetic_1}$ and $S_{HookLoop_1}$, and use those same goals for the evaluation of each combination of exploration and exploitation architectures. Table II provides the median L_2 distance between the goals and the reached sensory points for each condition (median on 2000 points times 100 trials). In the following, we give results of non-parametric statistical Mann-Whitney U tests for pairs of conditions.

Firstly, both if we consider conditions with environmental noise or not, all databases generated by flat exploration architectures and tested by any of the two exploitation architectures show a larger competence error than any of the databases explored with modular architectures and tested with both exploitation architecture ($p < 10^{-100}$). For instance, without environmental noise, the best performing flat condition is F-LWLR-RGB exploited with the NN algorithm, with a median competence error of 0.123, whereas the worst performing modular condition is M-NN-RMB, exploited with the LWLR algorithm, with a median competence error of 0.050, more than two times better.

Secondly, considering only exploration conditions without environmental noise, all databases generated with random model babbling architectures and tested by any of the two exploitation architectures show a larger competence error than any of the databases generated with active model babbling and tested with both exploitation architectures ($p < 0.05$). For instance, the median competence error using random model babbling and the NN algorithm both in exploration and exploitation is 0.046 whereas with active goal babbling it is 0.035. Using LWLR, those errors are 0.039 (RMB) and 0.026 (AMB).

IV. DISCUSSION

We have introduced two new algorithmic architectures for incremental exploration of sensorimotor spaces, exploiting a modular representation of these spaces. Random Model Babbling selects randomly which model to explore (which is itself explored through goal babbling) and Active Model Babbling (MACOB) uses a multi-armed bandit algorithm to

TABLE II

COMPETENCE ERROR IN SPACES OF INTEREST

Exploration architecture	Env. Noise	NN	LWLR
RmB	No	0.185	0.711
	Yes	0.307	0.871
F-NN-RGB	No	0.745	1.018
	Yes	1.174	1.253
F-LWLR-RGB	No	0.123	0.171
	Yes	0.376	0.422
M-NN-RMB	No	0.046	0.050
	Yes	0.248	0.261
M-NN-AMB	No	0.035	0.037
	Yes	0.285	0.300
M-LWLR-RMB	No	0.038	0.039
	Yes	0.216	0.227
M-LWLR-AMB	No	0.026	0.026
	Yes	0.215	0.226

maximize empirical learning progress. In a simulation involving complex high-dimensional motor (12D) and sensory (93D) spaces, we showed that these modular architectures were vastly more efficient than goal babbling methods used with flat representations, for all combinations of inverse models in the exploration and exploitation architectures. In particular, by focusing exploration on relevant parts of the space, they allowed the learner to discover efficiently how to move various objects using various tools, while flat architectures were not able to discover large parts of the space of effects. We also showed that active model babbling was significantly more efficient than random model babbling (yet the difference is smaller than between modular and flat architectures). To extend the scope of these results, further work is needed to study whether these results hold in experimental setups involving real physical robots, tools and objects.

Given the potential importance of such modular representations to address the challenge of incremental learning of skills in high-dimensional spaces, and within a life-long developmental perspective, this work also points to the need for algorithmic mechanisms that can generate automatically such representations.

Finally, from the perspective of modeling the discovery of tool use, an interesting feature of the experimental setup was that "tool" objects did not have a special status: they were represented in the same way as any other object. Yet, the active model babbling architecture converged to explore preferentially these objects and discovered their actual use as tools. Further algorithms could be developed to transform the modular architecture in a hierarchical architecture where causal dependencies between objects could be represented and leveraged, with the discovery of explicit object categories such as "tools". A possible approach could be to differentiate "tools" from other objects using a relative measure of learning progress, following the approach presented in [24] (section VIII. B. 2) to differentiate the self/body, physical objects and "others".

REFERENCES

[1] A. Cangelosi, M. Schlesinger, and L. B. Smith, *Developmental robotics: From babies to robots*. MIT Press, 2015.

[2] G. Baldassarre and M. Mirolli, *Intrinsically Motivated Learning in Natural and Artificial Systems*. Springer, 2013.

[3] E. Ugur, Y. Nagai, E. Sahin, and E. Oztop, "Staged development of robot skills: Behavior formation, affordance learning and imitation with motionese," *IEEE Transactions on Autonomous Mental Development*, vol. 7, no. 2, 2015.

[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. Da Silva, G. Konidaris, and A. Barto, "Learning parameterized skills," in *ICML*, 2012, pp. 1679–1686.

[6] A. Ude, A. Gams, T. Asfour, and J. Morimoto, "Task-specific generalization of discrete and periodic dynamic movement primitives," *Robotics, IEEE Transactions on*, vol. 26, no. 5, pp. 800–815, 2010.

[7] A. Baranes and P.-Y. Oudeyer, "Active learning of inverse models with intrinsically motivated goal exploration in robots," *Robotics and Autonomous Systems*, vol. 61, no. 1, 2013.

[8] A. G. Kupcsik, M. P. Deisenroth, J. Peters, and G. Neumann, "Data-efficient generalization of robot skills with contextual policy search." in *AAAI*, 2013.

[9] M. Rolf, J. Steil, and M. Gienger, "Goal babbling permits direct learning of inverse kinematics," *Autonomous Mental Development, IEEE Transactions on*, vol. 2, no. 3, 2010.

[10] A. Fabisch and J. H. Metzen, "Active contextual policy search," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3371–3399, 2014.

[11] F. Stulp and O. Sigaud, "Robot skill learning: From reinforcement learning to evolution strategies," *Paladyn. Journal of Behavioral Robotics*, vol. 4, no. 1, pp. 49–61, September 2013.

[12] F. Stulp, L. Herlant, A. Hoarau, and G. Raiola, "Simultaneous on-line discovery and improvement of robotic skill options," in *International Conference on Intelligent Robots and Systems (IROS)*, 2014.

[13] C. Moulin-Frier, S. M. Nguyen, and P.-Y. Oudeyer, "Self-organization of early vocal development in infants and machines: the role of intrinsic motivation," *Frontiers in Psychology*, vol. 4, 2014.

[14] S. M. Nguyen and P.-Y. Oudeyer, "Socially guided intrinsic motivation for robot learning of motor skills," *Autonomous Robots*, vol. 36, no. 3, pp. 273–294, 2014.

[15] F. Guerin, N. Kruger, and D. Kraft, "A survey of the ontogeny of tool use: from sensorimotor experience to planning," *Autonomous Mental Development, IEEE Transactions on*, vol. 5, no. 1, 2013.

[16] A. Stoytchev, "Behavior-grounded representation of tool affordances," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE, 2005.

[17] A. Antunes, G. Saponaro, A. Dehban, L. Jamone, R. Ventura, A. Bernardino, and J. Santos-Victor, "Robotic tool use and problem solving based on probabilistic planning and learned affordances."

[18] V. Tikhanoff, U. Pattacini, L. Natale, and G. Metta, "Exploring affordances and tool use on the icub," in *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on*. IEEE, 2013, pp. 130–137.

[19] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural computation*, vol. 25, no. 2, 2013.

[20] C. Moulin-Frier, P. Rouanet, P.-Y. Oudeyer, and others, "Explauto: an open-source Python library to study autonomous exploration in developmental robotics," in *ICDL-Epirob-International Conference on Development and Learning, Epirob*, 2014.

[21] S. Nguyen and P.-Y. Oudeyer, "Active choice of teachers, learning strategies and goals for a socially guided intrinsic motivation learner," *Paladyn*, vol. 3, no. 3, 2012.

[22] W. S. Cleveland and S. J. Devlin, "Locally weighted regression: an approach to regression analysis by local fitting," *Journal of the American statistical association*, vol. 83, no. 403, pp. 596–610, 1988.

[23] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on Scientific Computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[24] P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner, "Intrinsic Motivation Systems for Autonomous Mental Development," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 2, Apr. 2007.