Reusing Motor Commands to Learn Object Interaction

Fabien Benureau, Paul Fudal, Pierre-Yves Oudeyer

Reusing Motor Commands to Learn Object Interaction

Fabien Benureau[1,2], Paul Fudal[1], Pierre-Yves Oudeyer[1]
[1]Inria Flowers Team, [2]Bordeaux University, France

***Abstract*—We propose the *Reuse* algorithm, that exploit data produced during the exploration of an first environment to efficiently bootstrap the exploration of second, different but related environment. The effect of the Reuse algorithm is to produce a high diversity of effects early during exploration. The algorithm only constrains the environments to share the same motor space, and makes no assumptions about learning algorithms or sensory modalities. We illustrate our algorithm on a 6-joints robotic arm interacting with a virtual object, and show that our algorithm is robust to dissimilar environments, and significantly improves the early exploration of similar ones.**

I. MOTIVATION

We consider the problem of a robot exploring an unknown environment without a predefined goal to achieve. Exploring an environment for its own sake allows to build a broad knowledge base and a diverse skillset. Those can then be exploited later-on, when the need to act towards a specific goal arises. Often, in those situations where a practical, useful outcome is required, ressources and time constraints preclude learning new skills, and necessitate to quickly find a solution by adapting an existing one to the problem.

This pattern of exploration for later exploitation is consistently present in human learning. A child playing with a toy will interact with it in different ways: pushing, grabbing, stacking, squeezing, shaking, chewing, or throwing it. While doing so, it acquires crucial motor skills and affordance knowledge about common objects. Similarly, school education will teach students on an array of topics. Some of this knowledge acquired will only be useful in a few situations during life, but those situations might not offer the possibility to acquire the skill outright given the time and effort required to do so. As such, the relevant skills need to be assimilated ahead of time, before being able to know which one will be useful. To replicate such learning patterns in robots, we need to create exploration algorithms, and be able to exploit the result of those explorations.

Exploration algorithms face a specific set of challenges in robotics, where sensorimotor spaces typically exhibit recurring characteristics: they are highly-dimensional, redundant, non-linear, and the noise is not homogeneously distributed, making exhaustive approaches impossible, and naive ones – such as random sampling – inefficient. Redundancy and high noise make many actions yield poor or duplicate data from a learning standpoint. And, as any robotic interaction takes significant time, the number of actions a robot can execute in practice is severely limited. As such, selecting actions that generate good learning data is crucial for robots.

We propose the *Reuse* algorithm that exploit the exploration of a previous environment to explore a new, different environment more efficiently. Here, environments are one-step, episodic sensorimotor black-boxes, with predefined motor and sensory channels. The exploration algorithm can interact with the environment by submitting motor commands, and receive a sensory feedback, after which the environment is reset. The environment encompass anything beyond the interface available to the exploration algorithm: any pre- or post-processing of the motor and sensory signals, the body of the robot, and its surroundings. In this article, we will specifically consider environments involving object interaction.

To introduce the idea behind the Reuse algorithm, let's consider a baby playing with a toy rattle. After sufficient interaction, the baby will have learned that the rattle produces salient sounds when shaken or thrown. If the baby is then introduced to a bouncy ball for the first time, the baby may want to investigate if this new object is working the same way as a rattle, and shake and throw the ball. Shaking will not produce salient results, but throwing the ball will. By retrying interactions that produced salient effects on the rattle, the baby quickly produced salient effects on the ball.

We exploit this idea on a robot that creates a model of a first environment by exploring it. The robot is then introduced to a different environment, and we investigate the benefits yielded by reexecuting actions in the second environment that produced salient observations in the first, compared to restarting the exploration from scratch in the second environment. This strategy bets that the sets of actions that produces salient effects are at least partially overlapping across environments. It does not assume, however, that the environments have the same sensory modality – in our previous example, the rattle produced auditory stimulation, while the bouncy ball produced visual ones. Moreover, our approach does not make assumptions about the learning algorithm employed in one environment or the other. We show that the Reuse algorithm improves the learning performance of similar environments while being robust to dissimilar ones.

II. PROBLEM

In this section, we formally define the problem.
Henceforth, a *task* is defined as a set (M, S, f, n).

- M is the motor space, and it represents a parameterization of the movements the robot can execute. It is a bounded hyperrectangle of $\mathbb{R}^{d_M}$, with d_M the dimension of the motor space. In this article, the motors of the robot are

operated using dynamic movement primitives [23], whose parameters are vectors of real values choosen in $\mathbb{R}^{d_M}$.
- S is the sensory space, of dimension d_S: an arbitrary, bounded, subset of $\mathbb{R}^{d_S}$, with d_S the dimension of the sensory space. *Effects* and *goals* (desired effects) are elements of S.
- f is a function from M into S, returning the sensory feedback of the environment to a given motor command.
- n is the maximum number of samples of f allowed. It defines the number of actions the robot is allowed to execute to construct an inverse model of the environment. This model is a function $g_t : S \mapsto \mathcal{P}(M)$, with $\mathcal{P}(M)$ the set of subsets of M. $\mathcal{P}(M)$ allows us to encode the redundancy of the environment into the model, should we wish to do so.

In the remainder of this article, $\mathbf{x}$ will be used for motor commands and $\mathbf{y}$ for effects. An observation is a pair $(\mathbf{x}, \mathbf{y})$ with $f(\mathbf{x}) = \mathbf{y}$.

The *exploration trajectory* ξ_A of a task $A = (M_A, S_A, f_A, n_A)$ is defined as the sequence of n observations $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{0 \leq i < n_A}$ with $f_A(\mathbf{x}_i) = \mathbf{y}_i$, acquired during exploration.

Given a finite set of test goals E_A taken in S_A, we can compute the average error of the model over E_A at time t as:

$$e_A(t) = \frac{\sum_{\mathbf{y}_i \in E_A} \|f_A(g_t(\mathbf{y}_i)) - \mathbf{y}_i\|}{|E_A|}$$

with $|E_A|$ the cardinal of E_A. E_A is not known of the learner.

In this paper, we are interested at the improvement in learning performance that can be achieved if the exploration trajectory of a *source task* $A = (M_A, S_A, f_A, n_A)$ is available during the learning of a *target task* $B = (M_B, S_B, f_B, n_B)$, with the only condition that $M_A = M_B$, compared to learning B without additional knowledge. We can straightforwardly relax the constraint to $M_A \cap M_B \neq \emptyset$, but we won't consider such cases in this article.

We define $e_B(t, A, \xi_A)$, the average error at time t, on a task B that has access to the exploration trajectory ξ_A of the source task A. The error $e_B(t, A, \xi_A)$ depends on B, and on the algorithm that will be used to exploit the information contained in the description of A and the exploration trajectory ξ_A.

III. RELATED WORK

The simplest exploration algorithm is random motor babbling where motor commands are randomly chosen in the motor space. Since we are evaluating the diversity of effects that can be produced by the robot, this approach is particularly inefficient in highly redundant motor spaces, where many different motor commands generate the same result. Goal-directed exploration techniques solve this problem by selecting random goals rather than motor commands: this ensures that the exploration will try to learn to accomplish different things rather than learning to accomplish the same thing in different ways. When the dimension of the motor space is high but the dimension of the sensory space is low, this can increase learning performance by orders of magnitude, as [8] and [7] demonstrated in inverse kinematics tasks. We will employ a simple random goal-babbling algorithm to explore our environments.

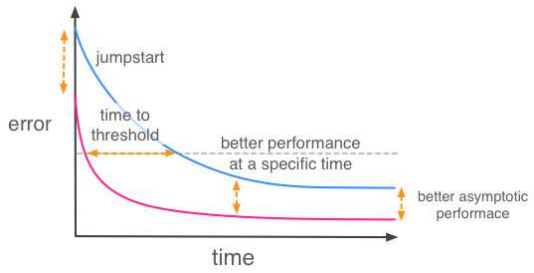

Fig. 1. Transfer can impact learning performance in different ways. The blue and pink curves are the average error in the forward or inverse model without and with transfer of learning data from a past task respectively. In the context of this paper, we are mainly interested in the error difference in the early stages of exploration.

Our approach brings together exploration algorithms and *transfer learning*. Transfer learning investigates the question of how to reuse what was learned in a situation in another [13], [15], [14]. A *source task* is learned, and knowledge extracted from this task is transferred to a *target task*, where it is leveraged to improve learning performance. Figure 1 introduces the different effect transfer typically can have on performance. Many different approaches have been proposed.

One can transfer the training data from the source task to the target task, eventually applying relevant transformations to it. In the context of reinforcement learning, model-based methods [12], [9] can transfer knowledge about the model of the environment to the target task. Starting point methods [18] set the final Q-table of a source task as the initial one of the target task, and usually provide a jumpstart for the performance. However, in either of those methods, when tasks don't match perfectly, an expert is needed to map the first task to the second task.

Another approach modifies the representation of the target task by leveraging source task knowledge, either reducing the dimensionality of the state or action space by discovering latent space parameterizations [16], or expanding it by adding new state variables in the target space [17]. If the environment is shared between tasks, the model of the world can be transferred to bias the learning of a control function, as in [19].

Our work shares resemblance to the imitation learning approach of [11], where source task policies are reused to guide exploration in the target task. In [11], transferred policies must, in the MDP formalism, share the same transition functions. Our algorithm does not have such a constraint, and the same actions will lead to different effects in each task.

Many of the transfer learning approaches would probably out-perform our approach in situations where they are applicable, as they make more assumptions about how the tasks are related, and exploit those assumptions. Our approach distinguishes itself by being applicable to autonomous settings,

and in particular applicable to situations where no assumptions can be made about the model, dynamics, or the existence of a reward signal for any of the environments.

Moreover, our approach is aimed at boostrapping the exploration of the target environment, rather than a specific learning goal.

This paper extends our previous work [10], were the evaluation was done on simple 2D simulations. Here we provide more complex environments using real robots and simulations, with motor primitives parameterized by dynamical motor primitives. Moreover, this work proposes a method of transfer simpler than [10].

IV. METHOD

Our method is organized around three algorithms. The first, EXPLORE(), describes the learning and exploration of the source task. The second, TRANSFER(), is applied at the end of the learning of the source task, and produces the data to be transferred to the target task. The third, REUSE(), controls how the transferred data impacts the exploration algorithm in the target task. The complete source code for these algorithms, as well as the one of the experimental setup is available to allow examination and ensure reproductibility[1].

A. Exploration and Learning for Source Tasks

Given a task (S_A, M_A, f_A, n_A), we train a predictor to compute a forward model of the environment, and use a constrained optimization routine on the predictor to compute the inverse model. For each interaction with the environment, the robot chooses the motor command to execute using a combination of motor and goal babbling.

*1) **Forward Model**:*

To approximate the function f from a set of observations, we employ Locally Weighted Linear Regression (LWLR) [1][2], a incremental machine learning algorithm. Since the aim here is to show a difference of learning performance between two exploration strategies, the performance baseline is of little concern, and there is no need to employ a sophisticated learning algorithm, the complexity of which might get in the way of understanding the effect studied here. The ability of the learning algorithm to extrapolate from existing observations is however crucial for an efficient exploration. LWLR met this criterion, while remaining simple and reasonably robust [3], which is why it was chosen here.

Given a set of observations $D = \{(\mathbf{x}_k, \mathbf{y}_k)\}$ where for each k, $f_A(\mathbf{x}_k) = \mathbf{y}_k$, and a query vector $\mathbf{x}_q$, for which we wish to predict the effect, we compute, for each point $\mathbf{x}_k$, the euclidean distance to $\mathbf{x}_q$ and derive a gaussian weight w_k:

$$w_k = e^{\frac{-\|\mathbf{x}_k - \mathbf{x}_q\|^2}{\sigma^2}}$$

We consider the matrices X with $X_{k,i} = (\mathbf{x}_k)_i$, Y with $Y_{k,i} = (\mathbf{y}_k)_i$, and $W = \text{diag}(w_0, w_1, ..., w_n)$, and compute:

$$\beta = (X^T W X)^+ (X^T W Y)$$

[1]To reviewers: it not yet available but it will be at the time of the publication

where $X^T W X$ is a positive definite symmetric matrix, and $(X^T W X)^+$ is its Moore-Penrose inverse [4].

Then:

$$\mathbf{y}_e = \beta^T \mathbf{x}_q$$

$\mathbf{y}_e$ is the LWLR estimate of $\mathbf{x}_q$, given the observed data D. We define the function PREDICTLWLR$(\mathbf{x}_q, D)$ that compute $\mathbf{y}_e$ for any $\mathbf{x}_q \in M_A$ given D.

In our implementation, σ, which control the locality of the regression, is dynamically computed. With d_{M_A} as the dimension of the motor space, we define a constant $N = 2d_{M_A} + 1$, and compute σ as the average distance of the N closest points of the query vector $\mathbf{x}_q$. All other points of D besides the N closest neighbors are given a weight of zero.

*2) **Inverse Model**:* Given a query point $\mathbf{y}_q \in S_A$, we want to compute an estimate of $\mathbf{x}_e \in M_A$ so that $\|f(\mathbf{x}_e) - \mathbf{y}_q\|$ is minimal.

Since M_A is a hyperrectangle of $\mathbb{R}^{d_{M_A}}$, we use L-BFGS-B [5][6], a quasi-Newton method for bound-constrained optimization, to minimize the error:

$$\mathbf{x}_e = \underset{\mathbf{x}}{\text{argmin}} \left(||\mathbf{y}_q - \text{PREDICTLWLR}(\mathbf{x}, D)||\right)$$

The optimization process is initialized with the motor command corresponding to the closest neighbor of $\mathbf{y}_q$ in the set of observations.

*3) **Source Task Exploration**:*

For each interaction with the environment, the exploration algorithm chooses a motor command to sample f_A. In robotics, a common situation is a motor space too large to be sampled exhaustively. In our experiments the number of allowed samples is small in comparison to what would be needed to exhaustively sample the motor space to a useful resolution. The works of [7] and [8] have shown that goal babbling is an effective method in these situations.

For each sampling of f_A, the exploration algorithm does either a random motor babbling action — picks a random point in the hyperrectangle M_A —, or does a goal babbling action, i.e. picks a point in the bounded sensory space S_A as a goal for the inverse model, and infers an motor command to execute that is likely to produce that goal.

In this paper, we are concerned with object interactions tasks. Such tasks distinguish themselves by having many motor commands producing no observable effect, because the robot did not manage to touch the object. To explore those spaces appropriately, the exploration algorithm is decomposed in two phases. The first phase is one of pure random motor babbling, and lasts an arbitrary number of K_{boot} samples. This allows the learner to gather several observations where the object was interacted with.

The second phase is dominated by goal babbling being chosen over motor babbling according to the probability p_{goal}. Since the goal babbling routine uses the current sensorimotor model f_A to optimise the infered motor commands to execute, it needs at least a few salient observations from the first phase to bootstrap correctly.

Algorithm 1: EXPLORE(A, K_{boot}, p_{goal}, p_{random})

Input:
- $A = (S_A, M_A, f_A, n_A)$, source task.
- K_{boot}, duration of pure motor bootstrapping.
- p_{goal}, ratio of goal babbling.
- p_{random}, ratio of random goal babbling.

Result:
- $\xi_A = \{\mathbf{x}_i, \mathbf{y}_i\}_{0 \leq i \leq n_A} \in (S_A \times M_A)^{n_A}$, exploration trajectory.

$\xi_A \leftarrow []$
for t **from** 0 **to** n_A **do**
  if $t \leq K_{boot}$ **or** RANDOM() $\geq p_{goal}$ **then**
    $\mathbf{x}_t$ = MOTORBABBLING(A)
  else
    $\mathbf{x}_t$ = GOALBABBLING(A, ξ_A)
  $\mathbf{y}_t \leftarrow f_A(\mathbf{x}_t)$ // execute the command
  add $(\mathbf{x}_t, \mathbf{y}_t)$ to ξ_A

MOTORBABBLING(A)
  choose $\mathbf{x}_t$ randomly in M_A
  return $\mathbf{x}_t$

GOALBABBLING(A, ξ_A)
  if RANDOM() $\leq p_{random}$ **then**
    choose a goal $\mathbf{g}_t$ randomly in S_A
  else
    choose a cell randomly in S_A among those that contain an already observed effect.
    choose a goal $\mathbf{g}_t$ randomly in the cell
  $\mathbf{x}_t = \underset{\mathbf{x}}{\text{argmin}} \, (||\mathbf{g}_t - \text{PREDICTLWLR}(\mathbf{x}_t, \xi_A)||)$
  return $\mathbf{x}_t$

Two strategies of goal babbling are employed. The first one (used with probability p_{random}) is random, with the next goal being uniformly drawn from S_A. The second one considers a uniform grid over S_A of a prespecified resolution along each axis. Then, it randomly chooses a grid cell among those that contains at least one effect, and creates a random goal within the cell. The first strategy tends to set goals that may be far from any observation, while the second will favor goals that are less ambitious and that generate motor commands not far from one whose effect has been observed. This is expressed in the GOALBABBLING() function in the EXPLORE algorithm.

B. Exploration in Target Tasks

1) ***Processing the Trajectory****:*

For each interaction in the second task, the learning algorithm can request reusing a motor command from the first task rather than doing random motor babbling. Goal babbling behavior is unchanged. Our reuse algorithm defines which motor command is transferred when such a request is made.

The whole assumption behind reexecuting motor commands from a previous tasks is that if they generated a diverse set of effects in the past task, they might generate a variety of effects in the current task as well, hence bootstrapping the model with good observations. Of course, this assumption hinges on the fact that the two tasks are sufficiently similar.

In order to generate a sequence of motor commands that generated a diverse set of effects, we reuse the grid of the goal babbling algorithm, and assign each cell with a bin. In this bin, we put the motor commands whose effect belong to the corresponding region. When a motor command is requested, we choose a random, non-empty, bin and draw, without replacement, a random motor command from the bin. This procedure is codified in Algorithm TRANSFER.

This procedure has a low computational cost, and only transfer structured set of motor commands. No sensory data is shared across tasks, hence the target task never tries to use the forward or inverse model of the source task.

Given a trajectory exploration ξ, we divide the sensory space into regions. In this article, we create regions in S_A using a simple grid. For each region of S_A, we define the bin b_R as the set of motor commands corresponding the observed effects belonging to R.

Algorithm 2: TRANSFER(ξ_A)

Input: $\xi_A = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{0 \leq i \leq n_A}$, exploration trajectory.
Result: $\mathcal{B}$, a set of motor commands bins.

$\mathcal{B}$ = empty set
Divide S_A into a set of regions $\mathcal{R}$
for $R \in \mathcal{R}$ **do**
  for $(\mathbf{x}_i, \mathbf{y}_i) \in \xi$, *with* $y_i \in R$ **do**
    add $\mathbf{x}_i$ to b_R
  add b_R to $\mathcal{B}$

2) ***Exploration****:*

We modify Algorithm 1 to replace the call to MOTORBABBLING() by a probabilistic call to REUSEBABBLING() and MOTORBABBLING(), according to a probability p_{reuse}, producing the REUSE algorithm.

V. EXPERIMENTS

We considered a hardware and simulated experimental setup where a 6 DOFs robotic arm interacts with an object, a cube or a ball, and observe the displacement of the object at the end of the interaction. For the hardware setup, we adopt a hybrid approach where the movement of the robot is executed on the hardware, while the trajectory of the end-effector is captured by a camera and replayed in a simulated physic engine where the interaction with a virtual object takes place. A video of the setup is available[2].

Algorithm 3: REUSE(B, $\mathcal{B}$, K_{boot}, p_{goal}, p_{transfer}, p_{random})

Input:
- $B = (S_B, M_B, f_B, n_B)$, target task.
- $\mathcal{B}$, set of bins of motor commands.
- K_{boot}, duration of pure motor bootstrapping.
- p_{goal}, ratio of goal babbling.
- p_{random}, ratio of random goal babbling.
- p_{transfer}, ratio of transfer motor babbling.

Result:
- $\xi_B = \{\mathbf{x}_i, \mathbf{y}_i\}_{0 \leq i \leq n_B} \in (S_B \times M_B)^{n_B}$, exploration trajectory.

$\xi_B \leftarrow []$
for t **from** 0 **to** n_B **do**
  if RANDOM() $\geq p_{goal}$ **then**
    if RANDOM() $\leq p_{transfer}$ **then**
      $\mathbf{x}_t$ = REUSEBABBLING(B, $\mathcal{B}$)
    else
      $\mathbf{x}_t$ = MOTORBABBLING(B)
  else
    $\mathbf{x}_t$ = GOALBABBLING(B, ξ_B)
  $\mathbf{y}_t \leftarrow f_B(\mathbf{x}_t)$ // execute the command
  add $(\mathbf{x}_t, \mathbf{y}_t)$ to ξ_B

REUSEBABBLING(B, $\mathcal{B}$)
  if *at least one bin of* $\mathcal{B}$ *is not empty* **then**
    choose a non-empty bin b_R of $\mathcal{B}$ randomly.
    draw $\mathbf{x}_t$ from b_R without replacement
    return $\mathbf{x}_t$
  else
    return MOTORBABBLING(B)

A. Hardware Experiments

The robots are a serial chain of 6 servomotors, with Dynamixel RX-64 as the three proximal motors and three RX-28 for the distal ones. Those servomotors are capable of delivering respectively 64 and 28 kg/cm of stall torque, with an angular resolution of 0.29 degrees, measured with a potentiometer, which is particularly subject to wear and tear. During the experiments, the motors were operated with a control loop of 100Hz.

1) Dynamic Movement Primitives:

The movements on the stems are generated using dynamic movement primitives (DMP). DMPs are parameterized dynamical systems introduced by Auke Ijspeert et al. [22]. They are computed from set of non-linear equations, that provides guarantees of smoothness, convergence, and robustness to perturbation. We chose DMPs, and the specific parametrization we explain below, because it allowed to express many different arm trajectories with a compact description. We use the

[2]fabien.benureau.com

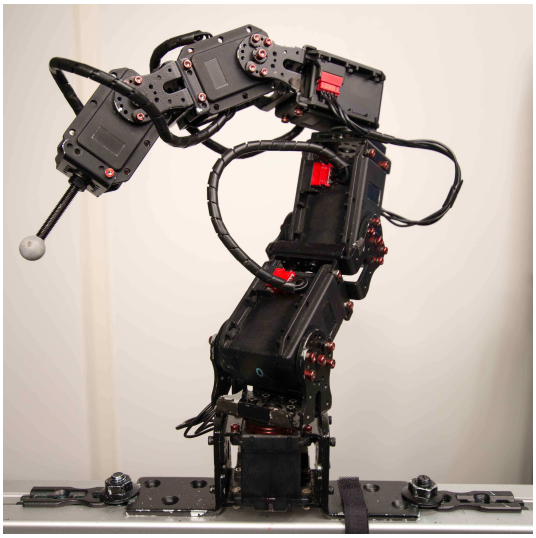

Fig. 2. Our robots sports 3 RX-64 servomotors from the base, then 3 RX-28 and a reflective marker at the tip.

implementation of Freek Stulp [25], based on [23] with the sigmoid variation of [24].

DMPs are based on damped spring dynamics, perturbed by a forcing term (equation 1), allow arbitrary smooth movements between start- and end-points. The forcing term is a sum of linear functions, each with a slope a_i offset b_i, and each weighted by a normalized gaussian activation function $\Phi_i(s_t)$ with center c_i and witdh σ_i (equation 3 and 4). s_t is the phase of the forcing term, described by an exponential decay term (equation 2. Those equations does not present the more complex case we used, where the sigmoid variation is included, see [24] for more details. In the following equations, τ is a temporal scaling factor, α and β are constant and g is the target state.

$$\tau \ddot{x}_t = \alpha(\beta(g - x_t) - \dot{x}_t) + f_t \quad (1)$$

$$\tau \dot{s}_t = -\alpha s_t \quad (2)$$

$$\Phi_i(s_t) = e^{-\frac{(s_t - c_i)^2}{2\sigma_i^2}} \quad (3)$$

$$f_t = \frac{\sum_{i=0}^{N} \Phi_i(s_t)(a_i s_t + b_i)}{\sum_{i=0}^{N} \Phi_i(s_t)} s_t \quad (4)$$

In our setup, the start- and end-points are the same ($g = x_0$), and correspond to the motor being in the zero position (depicted in figure 3). We used 2 basis functions per motor, with c_0 and c_1 fixed respectively at $1/3\tau$ and $2/3\tau$, with τ = 5s. σ_0 and σ_1 were shared by all motors. Each motor had independent a_0, a_1, b_0, b_1 parameters. With 6 motors, the motion trajectory of the robot can thus be described using a vector of dimension 26. After solving and integrating the dynamical system, we obtain each motor angular position in function of time. We set the ranges of the parameters so that 95% of trajectories would fall in between the angles the motor were able to produce, and clipped the rest to legal motor values.

Before executing the motion on the robot, we checked for self-collisions. If they are present, the trajectory is truncated to just before the collision to avoid damage.

2) **Hybrid approach:**

The robot has an reflective marker at the tip, which allows to capture its position at 120Hz during the movement using an OptiTrack Trio camera system, that has a sub-millimeter accurarcy. A virtual marker then replays the trajectory in a simulation where a virtual object has been put. The marker is the only object from the camera that is transported to the simulation, so it is the only part of the robotic arm that can collide with the object.

This hybrid approach between real hardware and simulated interaction yields many advantages: it is simpler, the robot never experience physical collisions, the precise measurement of the motion of the object does not require equipment. It also allow to create many different learning tasks, and allow the setup to be reproduced more easily. Yet, it also has problems. The interaction with the object is a poor match for what would happen in reality, and there is no kinesthetic feedback from the interaction with the object. Those problems are exacerbated with movements that push the object towards the ground.

3) **Virtual environment:**

For the interaction simulation, we used the robot simulator V-REP (Virtual Robot Experiment Platform). A simulation scene contains a virtual marker able to replay the captured trajectory. The marker only collides with the object, and does not interact with the ground. The object is constrained in its movements by the ground, the ceiling and four walls, forming a cube of 300 mm width. At the end of the simulation, a sensory feedback is computed and processed by sensory primitives.

4) **Sensory Primitive:**

We consider a simple sensory primitive that returns the displacement of the object projected on the ground at the end of the simulation. The displacement is returned as a vector of length 3: the displacement in x, in y, and a discreet dimension of saliency, which has value 0 if no collision happened, and 1 otherwise.

B. Software Experiments

We replicated the complete setup in simulation (figure 3). In simulation, the capture of the marker is ommited, but the marker is still the only part of the robot that can interact with the object.

At the end of the simulation, a sensory feedback is computed and processed by sensory primitives.

C. Learning Tasks

For all learning tasks, we allowed 1000 interactions, and fixed K_{boot} to 300, p_{goal} to 90%, p_{random} to 80% and p_{transfer} to 50%.

We consider 2 differents tasks on the hardware setup, a *cube* task and a *ball* task, with a width and diameter of 45mm respectively. In simulation, we consider the same tasks, but reduce the width and diameter to 25mm for added difficulty,

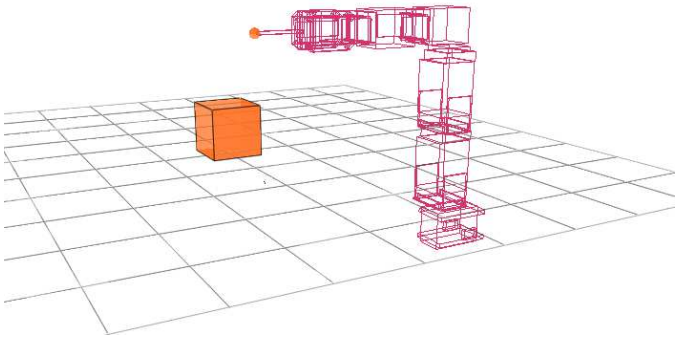

Fig. 3. The simulation environment, using ODE as the physic engine. All motors of the stems are in position zero, which correspond to the start and end position for each movement.

and another task *cube 2*, were the cube is moved to the right side by four time its width.

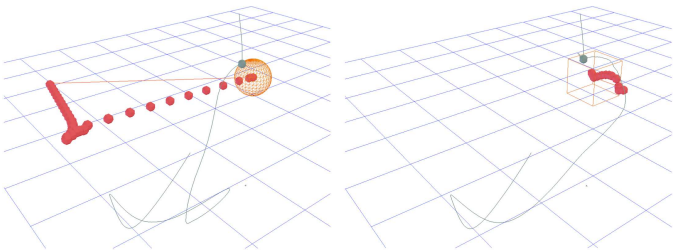

Fig. 4. A reused motor commands for the source ball task generates a different effect in the cube task. The motion of the marker (grey curve, with a dot for the position of the first collision) is affected by the interaction with the object. Red dots represent the position of the center of the object at fixed intervals. The walls are not represented, but their effects is seen on the motion of the ball.

For the tests, we consider an inverse model than compute the nearest neighbor for each test goal, rather than using LWLR (LWLR was used during the exploration phase). We generate 150 points uniformly distributed in S_B with the saliency dimension set to 1. Given a source task A and target task B (henceforth, we use the notation $A \rightarrow B$), compare the error $e_B(t)$ and $e_B(t, A, \xi_A)$ for regular values of t (we sampled at high frequency at the beginning, and then sampled every 25 interactions).

VI. RESULTS

A. Hardware Experiments

1) Effectiveness of reuse: Every combination of source/target task for the ball and cube was run four times on the setup, and additionally verified in simulation (10 repetitions). Below are the error graphs for one run in hardware (figure 5). All results were similar.

B. Simulation Experiments

1) Robustness of Reuse: To test the robustness of the reuse against dissimilar tasks, we considered the *cube* $\rightarrow$ *cube 2*

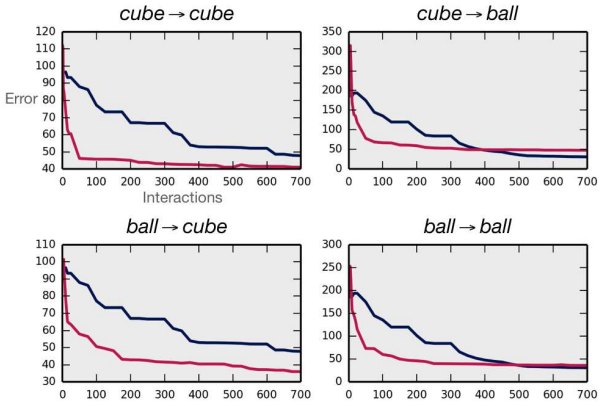

Fig. 5. Reusing motor command yield dramatic benefit for early learning. For each graph, the title give the source and target position, and the blue and pink curves are represent the error of the target task without and with reuse respectively. We can observe that the advantage of reuse in the cube target task is maintained at the end of the 700 interactions, whereas in the ball target task this advantage disapears after 400 iterations, occasioning even a negative performance in the case cube $\rightarrow$ ball at time $t = 700$

scenario, were the cube is moved significantly to the right. Most of the motor commands that were colliding with the cube the *cube* task would not collide anymore in the *cube2* task. With the reuse, the performance is not significanlty impacted.

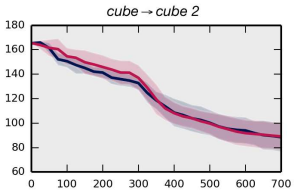

Fig. 6. Even in situations where is it not useful, the reuse is not detrimental to the learning performance. Averaged error over 10 repetitions.

It is important to note that, as reuse use iteraction to reexecute commands, the impact of using reuse for truly dissimilar tasks is necessarily negative. Its impact however, is limited since no bad data is introduced into learning. The negative impact of reuse could also be mitigated by recognizing that reuse does not create interesting observation quickly, and, when provided, using environmental cue to determine when two environments are similar.

2) Influence of Reuse: If we look at the distribution of effects here represented as the end position of the objects on the floor, we can observe several interesting things. Figure 7 shows the distribution for the *cube* task, the *ball* task and the *cube→ball* task. For the two latter task, effects are broken down by the type of exploration that generated them.

Looking at the ball task, we observe that random motor babbling is expensive in interactions while producing very few salient observations, here with a collision rate of about 3% over 350+ interactions. Goal babbling has a collision rate of more than 85%, over roughly twice as many interactions.

If the *cube* task is used as a source for the *ball* task, we observe that the reuse is highly effective, but that it generate effects skewed on one side, the same one that the effects of *cube* are skewed on. Because the motor babbling generated only 6 salient effects, the goal babbling exploration is heavily influenced by the effects produced through reuse, and effects produced at the end of learning for the *cube* $\rightarrow$ *ball* task are still heavily skewed, a pattern that did not develop to such a degree in the *ball* source task. This is an example of development being guided by previous knowledge.

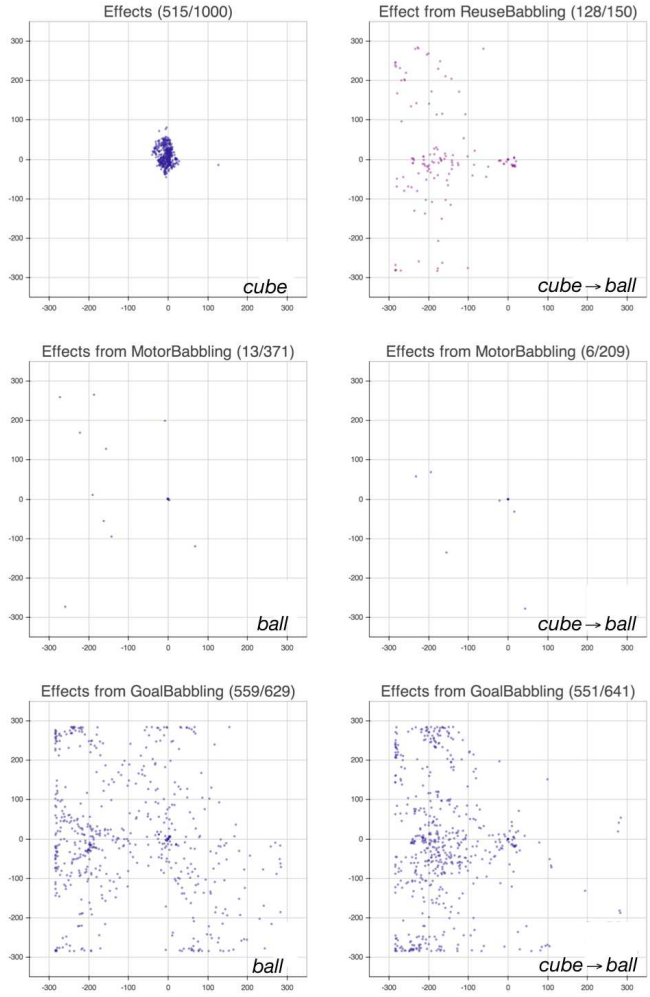

Fig. 7. The reuse of exploration trajectories can shape development. Here, the ball is learned to be moved predominantly on the left in *cube→ball*, compared to the unperturbed task *ball*. The *cube* source task displays a bias towards left effects, and produces skewed reuse observations in the *cube→ball* task, that have a lasting impact on the exploration. The number of salient effects versus the total number of interactions is indicated for each graph.

VII. DISCUSSION

As it currently stands, our approach has several limitations:

- The motor commands are grouped and discriminated in the sensory space. Yet, commands who generate effects in the same sensory region might have very different learning benefits. Our method does not distinguish between those.

- The setup of this article produced a constant effect in the majority of the motor space, and lots of variety in a small region of it. This inherently favors the approach we chose. We are currently experimenting with more complex situations to better evaluate the robustness of our algorithm.
- We only considered one learning algorithm in this article (optimization of a LWLR predictor using L-BFGS-B). We are currently running experiments with different learning algorithms to access robustness.
- In the experiments we arbitrarily fixed the parameters K_{boot}, p_{goal}, p_{random}, and p_{transfer} (before the results were produced). An empirical analysis of the influence of those parameters would be needed to better understand the dynamic of the transfer mechanism. Given that an interaction of the robot takes significant time, we were limited in the experiments will could conduct. We were able to deploy our simulation on a cluster, and plan to do such an analysis in the future.

ACKNOWLEDGEMENTS

This work was partially financed by the ANR MACSi and the ERC Starting Grant EXPLORERS 240 007. Computing hours for running simulations were graciously provided by the MCIA Avakas cluster.

REFERENCES

[1] W.S. Cleveland, S.J. Devlin, "Locally-Weighted Regression: An Approach to Regression Analysis by Local Fitting". *Journal of the American Statistical Association* 83(403):596610, 1988.

[2] C. G. Atkeson, A. W. Moore, S. Schaal, "Locally Weighted Learning", *Artificial Intelligence Review*, 11(1):11-73, 1997, doi:10.1023/A:1006559212014

[3] T. Munzer, F. Stulp, O. Sigaud, "Non-linear regression algorithms for motor skill acquisition: a comparison", *JFPDA14*, 2014.

[4] R. Penrose. "A generalized inverse for matrices", *Proceeding of Cambridge Philosophical Society* 51:406-413, 1955.

[5] R. H. Byrd, P. Lu and J. Nocedal, "A Limited Memory Algorithm for Bound Constrained Optimization", *SIAM Journal on Scien. and Stat. Computing* 16(5):1190-1208, 1995

[6] C. Zhu, R. H. Byrd and J. Nocedal, "L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization", *ACM Transactions on Mathematical Software*, 23(4):550-560, 1997

[7] A. Baranes, P-Y. Oudeyer, "Active Learning of Inverse Models with Intrinsically Motivated Goal Exploration in Robots", *Robotics and Autonomous Systems*, 2012

[8] M. Rolf, "Goal Babbling for an Efficient Bootstrapping of Inverse Models in High Dimensions", PhD Thesis *Bielefeld University*, 2012

[9] M. Lopes, T. Lang, M. Toussaint, P-Y. Oudeyer, "Exploration in model-based reinforcement learning by empirically estimating learning progress.", *Neural Information Processing System (NIPS)*, 2012

[10] F. Benureau, P-Y. Oudeyer, "Autonomous Reuse of Motor Exploration Trajectories", In Proc. *ICDL 2013*, Osaka, Japan.

[11] F. Fernández, M. Veloso. "Probabilistic policy reuse in a reinforcement learning agent." In *Proceeding of the fifth conference on Autonomous Agents and Multiagent Systems*, ACM, 720-727, 2006

[12] M. E. Taylor, N. K. Jong, P. Stone, "Transferring instances for model-based reinforcement learning." *Machine Learning and Knowledge Discovery in Databases.* Springer Berlin Heidelberg, 488-505, 2008

[13] M. E. Taylor, P. Stone, "Transfer learning for reinforcement learning domains: A survey." *The Journal of Machine Learning Research*, 10, (2009) 1633-1685

[14] S. J. Pan, Q. Yang, "A Survey on Transfer Learning", *IEEE Transactions on Knowledge and Data Engineering*, 10(1345-1359), 2010

[15] L. Torrey, J. Shavlik, "Transfer Learning", *Handbook of Research on Machine Learning Applications*, 2009

[16] F. Doshi-Velez, G. D. Konidaris, "Transfer Learning by Discovering Latent Task Parametrizations." In the *NIPS 2012 Workshop on Bayesian Nonparametric Models for Reliable Planning And Decision-Making Under Uncertainty*, 2012.

[17] M. G. Madden, T. Howley, "Transfer of experience between reinforcement learning environments with progressive difficulty." *Artificial Intelligence Review* 21.3-4 (2004) 375-398.

[18] S. Barrett, M. Taylor, P. Stone. "Transfer learning for reinforcement learning on a physical robot." In *The Ninth International Conference on Autonomous Agents and Multiagent Systems - Adaptive Learning Agents Workshop*, 2010.

[19] S. Thrun and T. Mitchell. "Lifelong Robot Learning". *Robotics and autonomous systems*, 1995.

[20] D. L. Silver, Y. Qiang, L. Lianghao, "Lifelong Machine Learning Systems: Beyond Learning Algorithms." *2013 AAAI Spring Symposium Series*, 2013.

[21] J. Konczak, "On the notion of motor primitives in humans and robots", *Lund University Cognitive Studies*, 2005

[22] A. J. Ijspeert, J. Nakanishi, S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots" *Proceedings. ICRA02. IEEE International Conference on Robotics and Automation*, 2:13981403, 2002.

[23] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors." *Neural Computation*, 25(2):328373, 2012.

[24] T. Kulvicius, K. Ning, M. Tamosiunaite, F. Worgotter, "Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting". *IEEE Transactions on Robotics*, 28(1):145-157, Feb 2012

[25] F. Stulp, `DmpBbo` – A C++ library for black-box optimization of dynamical movement primitives, https://github.com/stulp/dmpbbo.git, 2014