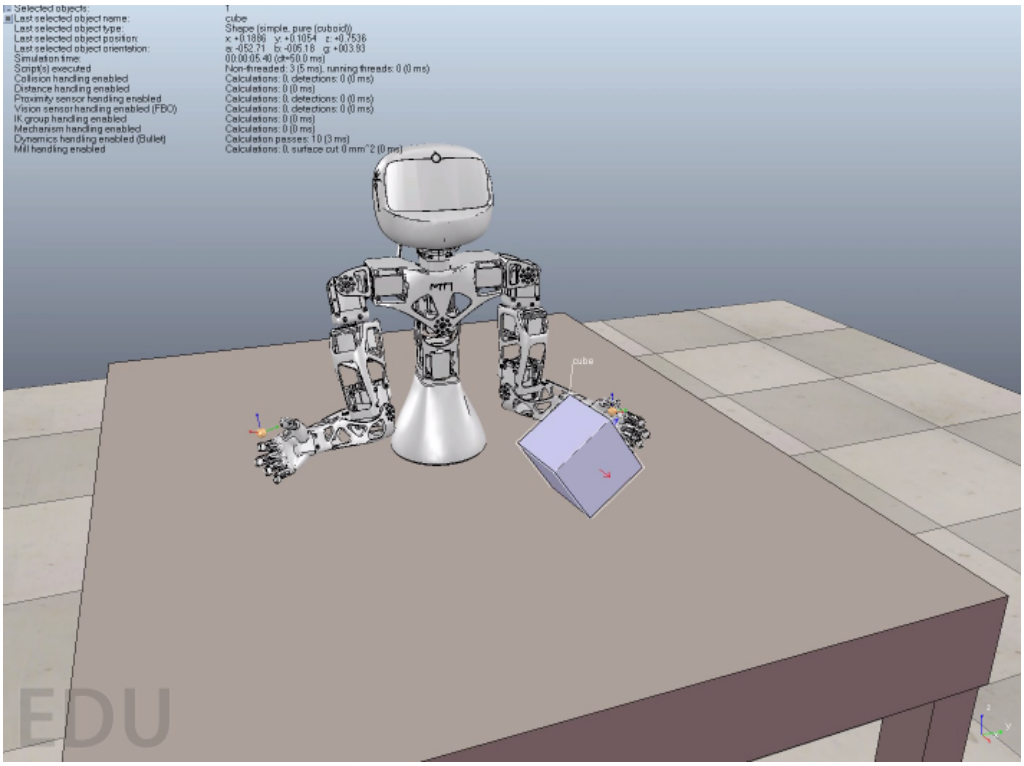
Robotic models of learning and development of speech and tool use.

Sébastien Forestier

October 15, 2015

Documentation and roadmap of my Master's internship and the beginning of my PhD thesis.



Contents

1 Introduction

During their first years of life, infants progressively learn to make complex arm gestures and finger movements to interact with the objects of the environment. At the same time, they also learn to produce physical effects on the world with another mean. They discover that some of their vocalizations can be used as tools to influence their social peers, as through the action of these social peers they are able to produce physical effects. To do so they learn how to use their vocal tract composed of many complex actuators from the larynx to the lips. An important scientific question is how these influences are discovered and by which mechanisms the corresponding abilities are developed.

Recent work in computational modeling has shown how mechanisms of active curiosity-driven learning, combined with imitation learning, could progressively self-organize developmental stages of increasing complexity in vocal skills. These developmental stages share many properties with the vocal development of infants in the first 6 months of life [Moulin-Frier et al., 2014a].

Like in other computational models of vocal development [Guenther et al., 2006, Howard and Messum, 2011, Warlaumont et al., 2013], the vocal agents in the model of Moulin-Frier et al. do not associate sounds to meaning, and do not link vocal production to other forms of action. In other models of language acquisition, one assumes that vocal production is mastered, and hand code the meta-knowledge that sound should be associated to referents or actions [Mangin and Oudeyer, 2012]. But understanding what kind of mechanisms can explain the smooth transition between the learning of vocal sound production and their use as tools to affect the world is still largely an open question.

In this work, we will extend models of curiosity-driven goal babbling and their interaction with social interaction presented in [Moulin-Frier et al., 2014a] so as to make steps towards addressing this question. In particular, we will integrate additional mechanisms to handle task and action spaces where the task spaces initially given to the robot can be encapsulated as primitive actions to affect newly learned task spaces.

We will first develop simple extensions of the SAGG-RIAC architecture with different sensorimotor models that can be learned in a hierarchical manner. A sensorimotor model is defined as first a task spaces (or sensori space) where tasks (or sensori states to be reached) can be set by a teacher, or self-generated by the learner and also a motor space to be used to solve those tasks. A sensorimotor model is learned well enough when the learner knows how to solve the tasks (or reach sensori goals) for which it has the means to. Then, in an action to solve a new task, a new model can reuse that previous model by giving it a sub-task that will be useful to solve the current one. We are here describing a hierarchy of models that the agent can learn and extend with new ones.

Given a simple hierarchy of 2 models, one reusing the other, a straightforward learning mechanism will be to explore the first model and after some time, to explore the second one. We will also study other mechanisms to handle this hierarchy of sensorimotor models that all need to be explored with an certain amount of trials that the agent could estimate.

In order to evaluate the hierarchical learning mechanisms, we will describe a first task for which a simple hierarchy of 2 sensorimotor models with the second reusing the first can be used to learn the task. A simulated robot arm will have to explore its motor space in order to understand how to push an object to different locations. The robot will first explore how to make movements with its hand

and then use this skill to explore the task of pushing an object. The simulated environment having a high variability due to collisions' unstability, that makes comparisons of architectures difficult. We will thus also implement a more simple environment composed of a simple geometric arm moving in 2 dimensions. Then, more complex hierarchies will be tested in an environment with the simple arm where a sequence of 2 movements has combinatorial consequences on the sensors.

Measuring exploration or learning is not generally equivalent. Exploration is a measure of how well the agent has found diverse outcomes in its task space. It makes no assumptions about the interest of exploring given regions of the task space from the viewpoint of the engineer, neither if we want the robot to learn specific tasks or if we know that certain tasks will be more useful later for the robot. However, in unstable parts of the sensorimotor environment, the exploration measured makes no sense for the robot as it might not have the competence to reach again that goal. For instance when a robot is trying to throw a ball to given position on a table, even if the robot tries the same motor command, it might never reach again a reached position due to the variability in the generated arm movement, and the collision between the arm and the ball. On the other hand, learning measures the competence of the robot on some tasks: how well the robot reaches a set of user-defined goals of the task space. The agent is evaluated on some tasks even if it did not know that it would be evaluated on those tasks. As the agent self-generates goals in the task space, it might not yet have explored that evaluated part of the environment, or in another run, have explored it very early, so it is hard to interpret this competence as a general progress. We will thus define an intermediate measure that will evaluate how well competent is the robot on the explored part of the task space.

Then, in order to assess a possible development of language seen as a tool to produce physical effects on an environment, we will develop a multi-modal environment composed of the DIVA vocal synthesizer [Guenther et al., 2006] and the previous 2D arm, in which combinations of sounds and arm movements make combinatorial changes in the environment.

In the firsts learning algorithms developed, the hierarchy of models to learn is given to the agent as if it already knew the combinations of actions to explore in order to learn the successive tasks. This is a strong hypothesis as infants have first to discover what types of actions to combine to get a specific outcome. For instance, the discovery that pushing on the knees and on the arms with certain coordination for a 6-month-old baby allow to move with an interesting high speed, but it does not need to be taught by parents. We will then develop an algorithm to autonomously complexify the sensorimotor contingencies learned by reusing the previously learned actions, with no predefined hierarchy of sensorimotor models. In this work we did not have time to assess those learning architectures but that will be done in future experiments.

Finally, we will derive a way to integrate those hierarchical learning architectures with social interactive learning from the SGIM-ACTS learning architecture [Nguyen and Oudeyer, 2012]. Different implementations will be developed, that could allow imitation of teachers and more generally social guidance in our models. The evaluations of those social learning algorithms remains to be done.

In the next section, related work will be presented on curiosity-driven learning, models of language acquisition, hierarchical representation of actions and social interaction. In Section 3, the methods of the experiments will be explained and we will present the details of the algorithms. In Section 4, the results of the conducted simulations will be presented, and a general discussion will be developed in Section 5.

2 Related work

2.1 Curiosity-driven exploration and developmental trajectories

Active curiosity-driven learning is a paradigm in which the robot or learning agent chooses itself what to explore or learn in its environment, based on some sort of internal model of its progress in the task of exploring or learning [Cangelosi et al., 2010]. With an intrinsic motivation to explore and learn in interesting situations, situations where the agent is progressing the most, the expected future progress is thus maximized. The agent does not necessarily know exactly what engineers are expecting from it, or on what task it should measure its own progress, but general principles permit to define what is an interesting situation. If the agent builds an internal model of its sensorimotor contingencies (e.g. What happen if I push that button ?), then it can monitor the accuracy of its sensorimotor model by experimenting with its environment, and compute the progress of its sensorimotor model, that will lead it to bias its experiments in those regions of the sensorimotor model that have shown a high prediction progress.

These ideas of curiosity-driven learning have a grounding in infants psychological experimentation [Forestier, 2014] that show that infants are seeking, by play, situations of intermediate novelty: situations neither too novel, neither fully understood. For instance, in order to understand how infants learn about the statistical properties of their sensory environment, Kidd and collaborators [Kidd et al., 2012] have tested how 8-month-old infants' attention to their environment is modulated by its statistical complexity. They presented 3 objects popping out of their respective boxes, in a sequence of varying complexity, as measured by its information content or surprise: the negative log probability of an object popping out of a given box. The measure of infants' attention is related to the time after which they look away from the object, given by eye-tracking tools. They show that infants have a Goldilocks preference: a preference for sequences neither too simple nor too complex. The explanation of this effect is that this preference gives infants a useful rate of sensory information and to avoid wasting cognitive resources on too complex stimuli. This is defined also in the "Flow" theory [Csikszentmihalyi, 1990] defining a state of pleasure obtained when engaged in activities challenging enough but not too much.

In order to model information seeking mechanisms and curiosity-driven learning suggested by behavioral experiments in humans and animals, Pierre-Yves Oudeyer uses an embedded setup with developmental robots. Those robots are learning in an open-ended manner: they are not defined to solve any task but rather are designed to be able to adapt to any unpredicted environment. The authors developed computational architectures to handle this curiosity driven learning: including the IAC [Oudeyer et al., 2007] and SAGG-RIAC [Baranes and Oudeyer, 2010] architectures. Here an intrinsic motivation to discover the sensorimotor environment is defined as a measure of prediction progress in the motor space (IAC) or competence-progress in the sensory space (SAGG-RIAC). The SAGG-RIAC architecture keeps a predictive model of the competence progress of the robot in different explored parts of the sensory space. To measure its own competence, the robot is self-generating goals to reach in the sensory space, and see how close to the goal it manages to go. The exploration is then biased towards parts of the space which give the highest competence progress, this mechanism ensuring a useful exploration in a space of high dimensionality where random exploration would not enable any learning. Implementations of these ideas include the Playground Experiment [Oudeyer et al., 2007] and self-organization of vocalizations [Moulin-Frier et al., 2014a].

In the Playground Experiment, a quadruped robot is placed in an infant play mat, with a responding robot peer next to it. The robot has to learn how to use its motor primitives to interact with its environment (IAC architecture). They observe the self-organization of developmental trajectories: in different runs with the same parameters, the robot explores objects and actions with different orders.

In the self-organization of vocalizations experiments, a simulated agent has to understand how to use a vocal synthesizer with the help of humans' phonetic items. The SAGG-RIAC architecture can generate phonetic goals to reach with the simulated vocal tract, in parts of the sensory space where competence progress is high, or try to imitate humans' phonetic items, choosing the strategy that again shows the best competence progress. They conclude that developmental trajectories of increasing complexity are emerging, with regularity and diversity. The diversity comes from different mechanisms: random generation in the algorithms, variability in the environment, and the multiples attractors of the learning dynamic system.

These developmental trajectories are similar to what can be observed during child development, which is an evidence toward a general purpose curiosity-driven system in infants' learning [Oudeyer and Smith, 2014]. A behavioral and cognitive epigenesis could develop and complexify behavioral and cognitive phenomena by the interaction with the environment. Hence communication skills could be the result of curiosity-driven exploration. Thus curiosity-driven learning, as an exaptation mechanism, could develop skills useless at the moment but which can turn to be recruited later for more complex unforeseen competences. This idea leads them to an evolutionary perspective saying that an intrinsic motivation for learning for itself, could have become at some evolutionary point a more efficient mechanism than one for developing specific genetically encoded skills to survive and reproduce [Oudeyer and Smith, 2014].

In our hierarchical implementations, even if the sensorimotor models will be part of a hierarchy of models, each of them will be explored using the SAGG-RIAC algorithm. Another computational layer will have to choose which model to explore based on their learning progress.

2.2 Models of language acquisition

In some models of language acquisition, the produced sounds are not related to actions the agent or the caregiver could make on the objects of the environment. The DIVA model (Directions Into Velocities of Articulators, [Guenther et al., 2006]) is a neural network that simulates the cortical interactions to produce syllables, that provides an account of different production phenomena, from speaking skill acquisition to coarticulations. The Elija model [Howard and Messum, 2011] also uses an articulatory synthesizer to pronounce sounds. It first gets a reward for exploration of its vocal outputs, and then is interacting with a modeled caregiver that imitates its sounds like a mother would do: either mimicking the infant's sounds or rather providing an intermediate sound between the infant's one and the adult one. The model learns the associations between its vocal commands and the response of the caregiver. The agent can thus learn the name of objects by trying to reproduce caregiver's utterances. In a self-organizing map neural network model of motor prespeech production [Warlaumont et al., 2013], experiments show that a reinforcement based on the similarity of the model's output sounds with a given set of vowels can bias the post-learning model's babbling sounds towards that reinforced set of vowels.

In other models of language acquisition, one assumes that vocal production is mastered, and hand code the meta-knowledge that sound should be associated to referents or actions. For instance, the

model from Mangin et al. [Mangin and Oudeyer, 2012] gets as input dance-like combinations of human movement primitives plus ambiguous labels associated to these movements. Using Non-negative Matrix Factorization [Paatero and Tapper, 1994], the model learns to recognize the movement primitives common to them, and when given new combinations, is able to produce a good set of labels to describe the combination.

We rather suppose that the agent has to autonomously learn that vocal sounds can be used, as tools, to affect the environment.

2.3 Hierarchical representation of actions

The study of the control of manipulation actions in humans has revealed a modular representation of actions either in the cerebral cortex and in the spinal cord with compositionality: an infinite number of movements can be expressed through combination of simple primitives, and generalization: certain neurons (higher in the hierarchy) can represent actions independently of the effectors used [Cangelosi et al., 2010].

The same idea holds for language expressiveness which is based on syntactic hierarchical combinations on a vocabulary, that open infinite semantic possibilities. Greenfield has also argued that this parallel between manipulation and language compositionality can be found in the human ontegenic development with combinatorial steps for manipulation and syntax acquired approximately at the same period and in the same order [Greenfield, 1991]. Also, the author explains that the development of the neural substrates for language and tool use could be an ontogenic homology as first of all the same neural computations for hierarchical combinations and their semantics should take place for both modalities, and furthermore experiments with Broca's and Wernicke's aphasics show that hierarchical organization for language and manipulation is linked. Broca's aphasics, who have less syntactic organization of speech were shown to also have problems of representation of the hierarchical organization of constructions with blocks, whereas Wernicke's aphasics, whose syntax is normal but speech semantics is impaired, succeed in representing such objects hierarchies.

Functional MRI experiments by Higuchi et al. have shown that the human's neural substrates for tool use and language is indeed shared in the dorsal BA44 Broca's area [Higuchi et al., 2009], which gives evidence for the similar neural computations used. They furthermore argue that these results supports the hypothesis that tool use have appeared first in primate evolution in F5 area, and then the language has developed in humans reusing part of tool use and manipulation neural substrates in human's Broca area, homolog of primate's F5.

Like a developing child, a developmental robot will have to incrementally explore skills that add up to the hierarchy of previously learned skills throughout its life, with a constraint being the cost and time of experimentation. We will seek to define curiosity-driven hierarchical learning architectures that could reuse the sensorimotor contingencies previously learned and to combine them to explore more efficiently new complex sensorimotor models.

Different computational models have the possibility to learn skill hierarchies. In finite environments represented by a factored Markov Decision Process [Vigorito and Barto, 2010], an intrinsic motivation towards actions maximizing Dynamic Bayesian Networks' structure has been shown to allow the learning of the environment's structure.

In continuous environments but with discrete actions, Metzen et al. [Metzen and Kirchner, 2013] use the framework of options [Sutton et al., 1999] to learn skill hierarchies. An intrinsic motivation rewards positively the novelty of the states encountered and negatively the prediction error of the learned skill model.

The model from Fabisch et al. [Fabisch and Metzen, 2014] learns in a setting with a discrete task space (called contexts). It uses an intrinsic motivation for learning progress, and a Multi-Armed Bandit algorithm (D-UCB) to choose on which context the agent should train for. The Upper Confidence Bound algorithm chooses between contexts given their estimated learning progress and the uncertainty of these estimations by picking the context with the maximum upper confidence bound. In other words, it maximizes the expected reward plus something related to the uncertainty associated with it, selecting either contexts with certain high rewards or ones with uncertain poor reward. This algorithm embeds directly a solution the exploration-exploitation trade-off problem as it represents the exploitation of knowledge by the expected progress and the exploration of other solutions by the uncertainty bonus. This algorithm supposes a stationary learning progress on each context so the authors use an adaptation (D-UCB, [Kocsis and Szepesvári, 2006]) to encompass non-stationary learning progress.

In a fully continuous setting, Mugan et al. [Mugan and Kuipers, 2009] have developed an algorithm that first learns a qualitative representation of environment states and actions in order to then learn the structure of Dynamic Bayesian Networks representing the temporal contingencies of those states and actions. In order to choose which action to practice, the authors use the IAC [Oudeyer et al., 2007] where the agent is intrinsically motivated to choose actions that are estimated to yield high prediction error progress.

Here, we will rely more specifically on the SAGG-RIAC architecture [Baranes and Oudeyer, 2013]. This architecture learns a single mapping between continuous motor and sensori (or task) spaces with a competence-based intrinsic motivation. In our hierarchy of sensorimotor models, each model will be explored using the SAGG-RIAC procedure, but it could be replaced by another one without changing the mechanisms to learn the hierarchy that will be assessed.

2.4 Learning from social interaction

During their first year of life, infants learn progressively how to use their vocal tract. At birth, they produce immature protophones like squeals, growls or quasi-vowels, and they are able to produce the vowels and speech-like syllables of their native language [Fenson et al., 1994] at one year old. Research on prelinguistic infants has been carried out about the way they learn to produce those specific sound, and has shown that they do not explore and learn only by themselves but rather that they spend a great part of their time interacting with their parents and other adults. At the age of 3 to 4 month-old [Kuhl, 1991], infants have been shown to imitate the vowels produced by an adult speaker, and therefore imitation is thought to be one of the important pathways to social and vocal development [Meltzoff and Warhol, 1999].

Others psychological studies have shown that infants early come to understand that their vocalizations can be used to get social feedback from their caregiver. For instance, Franklin et al. experimented how motivated the infant is in learning by this social interaction loop [Franklin et al., 2014] with a Face-to-Face/Still-Face/Reunion paradigm with 6 month-old. The mother and her child were in free interaction during the first phase to measure baseline vocalizations number and types. Then the mother was asked to stay looking at the child with no interaction during the second

phase, and they were free again in the third phase. They show an increase of protophones vocalizations from FF to SF phases in all categories measured: full vowels, quasi-vowels, squeals and growls, and not cry or laugh. They argue that by 6 month of age infants have learned that they can re-engage their parent through speech-like vocalizations, which is a step toward a pragmatic use of the perlocutionary effect of their vocalizations, that will be a component of their later communication abilities.

An other mechanism allowing efficient learning from social feedback is that infants may attend to engage in social interactions when they are ready to learn and have a high arousal level. This effect has been suggested in different settings: when infants are vocalizing, or pointing to objects. For instance, Goldstein et al. show that when an experimenter responded to infants looking and vocalizing to an object by naming the object, infants showed stronger label-object association than in a yoked condition where the experimenter only responded when infants were looking at the object and not vocalizing [Goldstein et al., 2010]. Also, in [Begus et al., 2014], infants were more able to learn from the demonstration by the experimenter of the function of an object when they had pointed to before than when they had pointed to a different object. Vocalizing and pointing to objects, as attempts to get information from adults might thus be linked to a focused attentional ready-to-learn state.

Thus infants imitate their parents and if this source of learning is not available, they know how to try to re-engage their parents into social interaction and, furthermore, they learn better when they engaged the interaction and received a natural response.

Those mechanisms of social learning have been modeled by different approaches. We are interested here in models in which the learning agent can autonomously choose when to learn by social interaction and when to learn without interaction, like an infant that sometimes plays alone and sometimes ask parents for interaction, with the hypothesis that the agent knows better than teachers what and when to learn. In robotics settings, NGuyen et al. have developed a learning architecture that can decide what to imitate, how, and from which teacher [Nguyen and Oudeyer, 2012]. The architecture called Socially Guided Intrinsic Motivation with Active Choice of Teacher and Strategy (SGIM-ACTS) estimates the learning progress from each strategy. The different strategies are: learning autonomously, or learning in interaction, by mimicry or by emulation. In those cases, the agent also learns from who to imitate, provided teachers of different skills are available. This learning architecture is assessed on moving objects tasks.

Moulin-Frier et al. have used the SGIM-ACTS framework to implement their model which is learning vocalizations by social interaction [Moulin-Frier et al., 2014a]. At each exploration step, their agent has to first choose either to autonomously explore the sounds produced by its vocal tract, or to engage in social interaction, based on the learning progress estimated for each strategy. If the exploring strategy is chosen, the agent uses the SAGG-RIAC architecture to self-generate goals to try to reach in the sensori space composed of the two first formants of the produced sound. Otherwise, the agent engages in social interaction and receives one of the predifined adult's sounds as demonstration, and tries to imitate that sound.

We will extend our models of hierarchical skill learning to integrate the mechanisms of evaluation of the interest of social guidance of SGIM-ACTS that allow to learn efficiently from different channels of social guidance.

3 Methods

3.1 Exploration architectures

In this section we first give details about models and how they are learned, then define the learning architectures developed in this work, and the control architectures to compare with. We use and extend the Explauto library [Moulin-Frier et al., 2014b] which purpose is to study autonomous exploration in developmental robotics.

3.1.1 Learning a model

In the Explauto framework, a sensorimotor model mapping a motor space to a goal space is learned together with an interest function over the goal space. These 2 functions are considered as slots that can be instanciated by any algorithms, without changing the considered architecture.

In our experiments, the mapping between the motor space and the sensori space will be done by storing all the sensorimotor experience (list of tuples (m, s)), and computing nearest neighbors on that dataset. For instance, to predict the outcome of a given motor command m, we look for the nearest neighbors of m in the motor part of the dataset, and output the corresponding s. On the other hand, to find the motor command that would allow to reach a given sensori experience s, we look for the nearest neighbors of s in the sensori part of the dataset, and output the corresponding m. The nearest neighbor function is the simplest regression algorithm and makes no hypothesis on the mapping, but more complex algorithms could be used like Locally Weighted Regression [Cleveland and Devlin, 1988], that supposes a locally linear mapping.

The interest function evaluates how much interesting exploring a given part of the goal space is. In the motor and goal babbling algorithms, we do not use such a interest function. A motor babbling agent just randomly draw motor commands to execute and stores its sensori input. A goal babbling agent have a constant interest function: it randomly selects goals to be explored.

An iteration of the learning of a model is as follows: the robot selects a point in its goal space according to the interest function and infers parameters to get close to that goal based on the past sensorimotor experience. It also adds some exploration noise to discover new motor configurations near the ones already known. Then it executes the motor trajectory corresponding to the parameters, and observes the consequences in the goal space.

More elaborated interest functions include the SAGG-RIAC procedure [Baranes and Oudeyer, 2010] that splits the goal space in a binary tree and infers the interest to explore on each leaf of the tree. A leaf is splitted when a maximum number of points is reached (100 in our experiments), and the split value is chosen successively on each dimension of the goal space. The value is chosen to maximize the difference of interest in the 2 sub-regions of the leaf:

$$split\_value = argmax(N_1 \times N_2 \times |interest(R_1) - interest(R_2)|),$$

where N_i is the number of points in the subregion i and $interest(R_i)$ the interest of exploring subregion i.

The interest is computed as the absolute value of the competence progress, as a decreasing competence means that something has changed in the corresponding region (e.g. a motor has crashed so unexpected sensori experience happens), that also become interesting. The competence on the task s is then the distance between s and the actually reached sensori experience s' when trying to get as close to s as possible. In our experiments, the interest of a region of the tree is measured as the difference of competence between the firsts and lasts experiments:

$$interest(R_i) = \left| \frac{\sum_{i=3n/4}^{i=n} competence(i) - \sum_{i=1}^{i=n/4} competence(i)}{n/4} \right|$$

3.1.2 Control architectures

We compare our architectures to the control one where the robot learns directly a mapping between M and S, either with a competence-based intrinsic motivation (Algorithm 2) or a fully random motor babbling (Algorithm 1).

Algorithm 1 Motor Babbling Flat Architecture

Require: Motor space $M = \{m_1, ... \ m_m\}$
Require: Sensor space S
`sm_model` $\leftarrow$ SMModel(M, S) ▷ Initializes the sensorimotor model
loop
  $m \leftarrow$ Random(M) ▷ Draws a random motor command
  $s \leftarrow$ Environment(m) ▷ Applies it and observes the effect on the environment
  Update `sm_model` with (m,s)
end loop

Algorithm 2 Goal-Space Flat Architecture

Require: Motor space $M = \{m_1, ... \ m_m\}$
Require: Sensor space S
`sm_model` $\leftarrow$ SMModel(M, S)
`im_model` $\leftarrow$ IMModel(S)
loop
  $s_g \leftarrow$ Choose(`im_model`) ▷ The interest model generates a goal in S
  $m \leftarrow$ Infer(`sm_model`, s_g) ▷ The sensorimotor model infers a command to reach this goal
  $s \leftarrow$ Environment(m)
  Update `sm_model` with (m, s)
  Update `im_model` with (s, s_g)
end loop

3.1.3 Simple hierarchical architecture

In the case of 2 sensorimotor models, with reusing the other, we can define a simple hierarchical architecture that explores first the first model and then the second. The agent explores first the mapping between M and S_1 and then a mapping between S_1 and S_2. When exploring the second one, the agent chooses a point in S_1 to produce and uses the first mapping to select the best motor configuration to obtain this subtask.

We define the Motor Babbling Simplest First Hierarchical Architecture (Algorithm 3) as a control to compare with the Goal-Space Simplest First Hierarchical Architecture (Algorithm 4), in which an interest model, possibly a random goal babbling or the SAGG-RIAC-like tree, chooses the goals to explore.

Algorithm 3 Motor Babbling Simplest First Hierarchical Architecture

Require: Motor space $M = \{m_1, ... \ m_m\}$
Require: Sensor spaces $S_1, \ S_2$
Require: Total number of iterations n
$\texttt{sm\_model1} \leftarrow$ SMModel$(M, \ S_1)$
$\texttt{sm\_model2} \leftarrow$ SMModel$(S_h, \ S_2)$
for $n/2$ iterations **do**
    $m \leftarrow$ Random(M)
    $s_1, \ s_2 \ \leftarrow$ Environment(m)
    Update $\texttt{sm\_model1}$ with $(m, \ s_1)$
    Update $\texttt{sm\_model2}$ with $(s_1, \ s_2)$
end for
for $n/2$ iterations **do**
    $s_1 \leftarrow$ Random(S_1) ▷ Draws a random hand movement
    $m \leftarrow$ Infer($\texttt{sm\_model}$, s_1)
    $s'_1, \ s_2 \ \leftarrow$ Environment(m)
    Update $\texttt{sm\_model1}$ with $(m, \ s'_1)$
    Update $\texttt{sm\_model2}$ with $(s'_h, \ s_2)$
end for

3.1.4 Strategic choice

As the switch between the exploration of the two models might be difficult to set by hand, another approach is to let the agent choose at each iteration to explore the model that yields the highest competence progress (Algorithm 5). This strategic learning approach has first been proposed to allow an agent to choose between different learning modalities [Nguyen and Oudeyer, 2012].

Those first algorithms are defined in the case of the learning of 2 sensorimotor models, but the extension to an arbitrary hierarchy of sensorimotor models is straghtforward: we explore the models from the lower layer to the higher.

3.1.5 Interacting models in hierarchy

The previous hierarchical learning architectures considered models as independents building blocks that each explores a mapping with its own policy. However, optimally exploring the different models of the hierarchy does not imply that we are optimally exploring the overall hierarchy, as parts of the sensori space of an intermediate model might be of no relevance to the complex skills being learned, and others might be of critical importance. For instance, if a first model learns how to move the hand of the robot at different positions, and a second model learns the consequencies of pushing on certain buttons in front of the robot, then learning to acurately to put hand of the robot behind it for the first model is not useful to refine the other models. On the other hand, being able to robustly reach the different buttons is of major interest for the second model and so could be more explored by the first model. We thus define here 2 types of architectures where the models do not learn independently but communicate information what seems useful for them to learn.

Algorithm 4 Goal-Space Simplest First Hierarchical Architecture

Require: Motor space $M = \{m_1, ... \ m_m\}$
Require: Sensor spaces $S_1, \ S_2$
Require: Total number of iterations n
`sm_model1` $\leftarrow$ SMModel($M, \ S_1$)
`sm_model2` $\leftarrow$ SMModel($S_1, \ S_2$)
`im_model1` $\leftarrow$ IMModel(S_1) ▷ Initializes the interest models
`im_model2` $\leftarrow$ IMModel(S_2)
for $n/2$ iterations **do**
    $s_1 \leftarrow$ Choose(`im_model1`) ▷ The first interest model generates a goal in S_1
    $m \leftarrow$ Infer(`sm_model1`, s_1) ▷ The first sm infers a command to reach this goal
    $s'_1, \ s_2 \ \leftarrow$ Environment(m)
    Update `sm_model1` with ($m, \ s'_1$)
    Update `sm_model2` with ($s'_1, \ s_2$)
    Update `im_model1` with ($s'_1, \ s_1$)
end for
for $n/2$ iterations **do**
    $s_2 \leftarrow$ Choose(`im_model2`) ▷ The second interest model generates a goal in S_2
    $s_1 \leftarrow$ Infer(`sm_model2`, s_2) ▷ The second sm model infers a hand movement
    $m \leftarrow$ Infer(`sm_model1`, s_1) ▷ The first sm model infers a command
    $s'_1, \ s_2 \ \leftarrow$ Environment(m)
    Update `sm_model1` with ($m, \ s'_1$)
    Update `sm_model2` with ($s'_1, \ s'_2$)
    Update `im_model2` with ($s'_2, \ s_2$)
end for

Algorithm 5 Goal-Space Strategic Choice Hierarchical Architecture

Require: Motor space $M = \{m_1, ... \ m_m\}$
Require: Sensor spaces $S_1, \ S_2$
`sm_model1` $\leftarrow$ SMModel($M, \ S_1$)
`sm_model2` $\leftarrow$ SMModel($S_1, \ S_2$)
`im_model1` $\leftarrow$ IMModel(S_1)
`im_model2` $\leftarrow$ IMModel(S_2)
`models_interest` $\leftarrow$ MAB(`im_model1`, `im_model2`) ▷ Initializes a Multi-Armed Bandit on models
loop
    `model_to_train` $\leftarrow$ Choose(`models_interest`)
    if `model_to_train` = `im_model1` **then**
        $s_1 \leftarrow$ Choose(`im_model1`)
        $m \leftarrow$ Infer(`sm_model1`, s_1)
        $s'_1, \ s_2 \ \leftarrow$ Environment(m)
        Update `sm_model1` with $(m, \ s'_1)$
        Update `sm_model2` with $(s'_1, \ s_2)$
        Update `im_model1` with $(s'_1, \ s_1)$
        Update `models_interest` with Interest(`im_model1`) ▷ The MAB is updated with the current interest of the first interest model
    else if `model_to_train` = `im_model2` **then**
        $s_2 \leftarrow$ Choose(`im_model2`)
        $s_1 \leftarrow$ Infer(`sm_model2`, s_2)
        $m \leftarrow$ Infer(`sm_model1`, s_1)
        $s'_1, \ s_2 \ \leftarrow$ Environment(m)
        Update `sm_model1` with $(m, \ s'_1)$
        Update `sm_model2` with $(s'_1, \ s'_2)$
        Update `im_model2` with $(s'_2, \ s_2)$
        Update `models_interest` with Interest(`im_model2`) ▷ The MAB is updated with the current interest of the second interest model
    end if
end loop

Top-Down Drive

Here, we let the highest model in the hierarchy draw goals in S_2, then infer an interesting hand's movement in S_1 to try in order to reach that goal (Algorithm 6). The lowest model in the hierarchy is now given a certain amount of iterations to try motor configurations to obtain this hand's movement. It can do so either by sampling around its best already known motor configuration for that hand's movement, or with a more sophisticated black-box optimization technique like the Covariance Matrix Adaptation Evolution Strategy [Hansen, 2006].

Algorithm 6 Goal-Space Top-Down Drive Hierarchical Architecture

Require: Motor space $M = \{m_1, ... \ m_m\}$
Require: Sensor spaces $S_1, \ S_2$
1: `sm_model1` $\leftarrow$ SMModel($M, \ S_1$)
2: `sm_model2` $\leftarrow$ SMModel($S_1, \ S_2$)
3: `im_model1` $\leftarrow$ IMModel(S_1)
4: `im_model2` $\leftarrow$ IMModel(S_2)
5: **loop**
6: $s_2 \leftarrow$ Choose(`im_model2`)
7: $s_1 \leftarrow$ Infer(`sm_model2`, s_2)
8: Explore_Around(`sm_model1`, s_1) ▷ The first model can explore for a few iterations in order to reach s_1
9: $m \leftarrow$ Infer(`sm_model1`, s_1) ▷ Now the best motor command is chosen
10: $s'_1, \ s_2 \ \leftarrow$ Environment(m)
11: Update `sm_model1` with ($m, \ s'_1$)
12: Update `sm_model2` with ($s'_1, \ s'_2$)
13: Update `im_model2` with ($s'_2, \ s_2$)
14: **end loop**

We also explicit an exemple with 2 lower models and 1 higher model in the hierarchy (Algorithm 7). This algorithm can be extended to a tree hierarchy i.e. a hierarchy where each model is connected to at most 1 higher level model. If a model *mod* is connected to 2 higher level models, each giving *mod* a goal to explore, then the choice of the goal to explore for *mod* is not straightforward. A possibility to extend this algorithm to other hierarchies (where only one model is not reused) is derived in the next paragraph.

Algorithm 7 Goal-Space Top-Down Drive 2-low/1-high Hierarchical Architecture

Require: Motor spaces $M_1,\ M_2$
Require: Sensor spaces $S_1,\ S_2,\ S_3$
1: `sm_model1` $\leftarrow$ SMModel($M_1,\ S_1$)
2: `sm_model2` $\leftarrow$ SMModel($M_2,\ S_2$)
3: `sm_model3` $\leftarrow$ SMModel($S_1S_2,\ S_3$)
4: `im_model1` $\leftarrow$ IMModel(S_1)
5: `im_model2` $\leftarrow$ IMModel(S_2)
6: `im_model3` $\leftarrow$ IMModel(S_3)
7: **loop**
8: $s_3 \leftarrow$ Choose(`im_model3`)
9: $s_1,\ s_2 \leftarrow$ Infer(`sm_model3`, s_3)
10: Explore_Around(`sm_model1`, s_1) ▷ The first model can explore for a few iterations in order to reach s_1
11: Explore_Around(`sm_model2`, s_2) ▷ The second model can explore for a few iterations in order to reach s_2
12: $m_1 \leftarrow$ Infer(`sm_model1`, s_1)
13: $m_2 \leftarrow$ Infer(`sm_model1`, s_2)
14: $s'_1,\ s'_2,\ s'_3 \leftarrow$ Environment(m_1m_2)
15: Update `sm_model1` with ($m_1,\ s'_1$)
16: Update `sm_model2` with ($m_2,\ s'_2$)
17: Update `sm_model3` with ($s'_1s'_2,\ s'_3$)
18: Update `im_model3` with ($s'_3,\ s_3$)
19: **end loop**

Merging interest models for hierarchy

Another possibility for a higher model to guide the exploration of a lower is to compute the interest of the second model in its motor space S_1 as the inverse of the interest distribution (the distribution of goals to be chosen in S_2, by the sensorimotor mapping function). The inverse of the distribution can be computed through Monte-Carlo sampling. The lower model then gets a guided interest distribution in its sensori space S_1, and merges this distribution with its own interest distribution when it has to generate goals in S_1. If a model is connected to more than 1 higher models, it has to merge its own interest distribution with each of the guided output distributions from the higher models that are connected to it. If a model is connected to more than 1 lower model, it outputs one distribution for each connected lower model, computed by the same inversion mechanism. This algorithm has not yet been implemented.

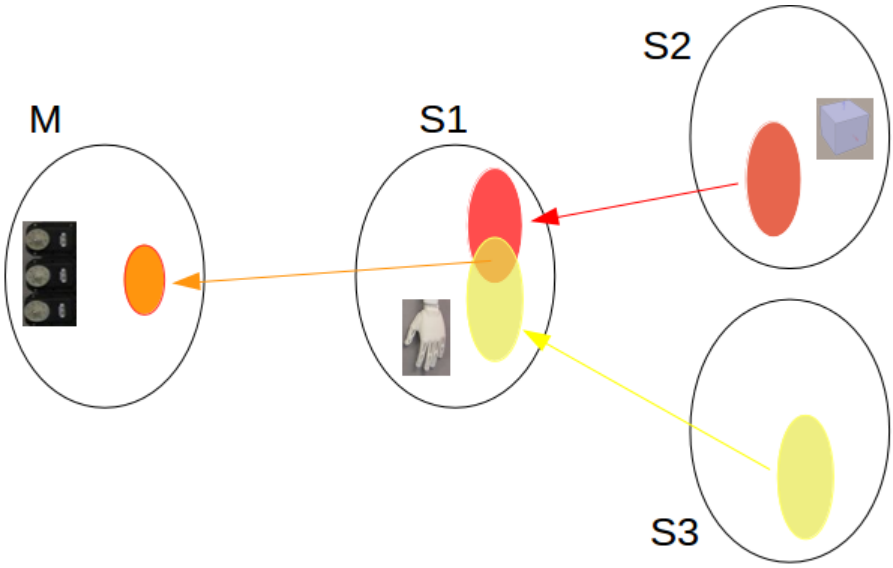


Figure 1: Top-Down Drive to guide exploration of lower level models

3.1.6 Exploration of sensorimotor models of increasing complexity

In the previous learning algorithms developed, the hierarchy of models to learn is given to the agent as if it already knew the combinations of actions to explore in order to learn the successive tasks. We describe here an algorithm (Algorithm 8) that autonomously generates different hierarchies of sensorimotor models and decides which of them to explore. To explore a hierarchy, it then uses the previously described algorithms that explore in a fixed hierarchy (e.g. Top-Down Drive, Sec. 3.1.5).

Let $O = \{O_1, ... O_p\}$ be a set of operators allowed to combine the actions to reuse, for instance the sequence operator, that takes 2 actions and return the action that executes those 2 actions in sequence. Also, the parallel operator takes 2 actions and return the action that executes those 2 actions in parallel, when possible, and if the 2 actions take different values on some motor, one of them is randomly chosen.

Given a set of motor spaces and sensori spaces (for instance vocal and arm motor spaces, visual and audio sensori spaces), the algorithm explores first the simplest sensorimotor models (e.g. M_1S_1, M_1S_2, M_2S_1, M_2S_2), and when making too small progress on those models complexifies the hierarchy of models. It then combine known models with the allowed operators to form a new more complex one (e.g. a sequence of an action in S_1 with the M_1S_1 model and an action in $M2$, with S_1 as sensorimotor model). The algorithm increases the complexity of the hierarchies step by step, the complexity being the number of sensorimotor models reused plus the number of sensori spaces used.

To decide which hierarchy to explore at each learning time step, a Multi-Armed Bandit algorithm monitors interest to explore (e.g. the competence progress) on each of them. The interest of a hierarchy is computed as the competence progress over the last experiments on the models of that hierarchy (with a sliding window).

Each model can be used by several hierarchies that do not duplicate it but share it. Exploring a given hierarchy can thus benefits the other hierarchies. This algorithm has not yet been implemented.

Algorithm 8 Exploration of Sensorimotor Models of Increasing Complexity

Require: Motor spaces $M = \{M_1, ... M_m\}$
Require: Sensor spaces $S = \{S_1, ... S_n\}$
Require: Operators $O = \{O_1, ... O_p\}$
`models_MAB` $\leftarrow$ MAB(list of models $(M_i S_j)$ with $i \leq m$ and $j \leq n$) ▷ Initialize a Multi-Armed Bandit with models of complexity 0
loop
  if all models in `current_models` are not interesting **then**
    `actions` $\leftarrow M \cup$ `models_MAB`
    `combined_actions` $\leftarrow$ O(`actions` $\times$ `actions`)
    `possible_models` $\leftarrow$ (`actions` $\cup$ `combined_actions`)$\times P(S)$
    $c \leftarrow$ max_complexity(`current_models`)
    if some model of `models_MAB`, has complexity c **then**
      `new_model` $\leftarrow$ a model of complexity c from `possible_models`
    else
      `new_model` $\leftarrow$ a model of complexity $c + 1$ from `possible_models`
    end if
    add `new_model` to `models_MAB`
  else
    `model` $\leftarrow$ Choose(`models_MAB`)
    Train(`model`)
    Update `models_MAB` with Interest(`model`)
  end if
end loop

3.1.7 Formulation as a Directed Acyclic Graph

The hierarchies of models can be formalized more efficiently, with a directed acyclic graph (DAG) that links motor and sensori domains and intermediate and top-level models (the models that are not reused by higher level models). Each model could then receive top-down drive from all the models from different hierarchies that reuse it, which would be more efficient that exploring independently the hierarchies with the Top-Down Drive algorithm. See Fig. 2 for an exemple of such hierarchies. In order to choose which model to explore, a tradeoff can be done for each model between autonomous interest and top-down drive and social guidance interests. The model to explore would then be the one with max tradeoff interest.

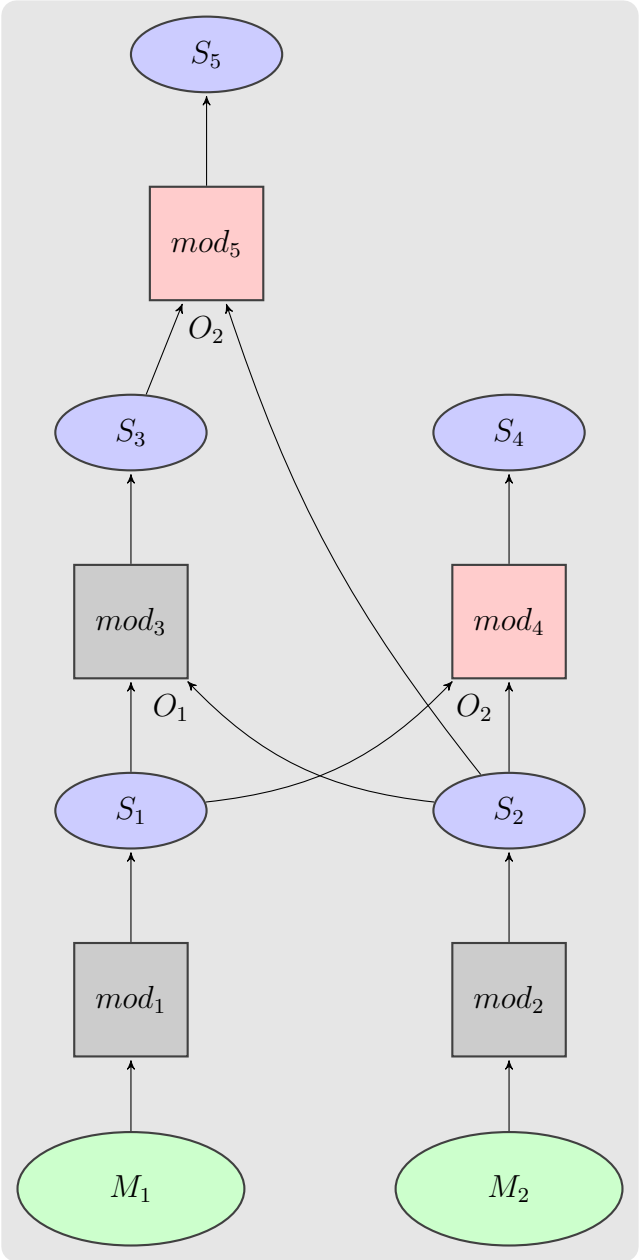


Figure 2: Exemple of 2 hierarchies sharing models. Ellipses represent domains. Green ellipses: motor domains. Blue ellipses: sensori domains. Squares represent models. Gray squares: intermediate models. Red squares: top-level domains. First hierarchy is composed of intermediate models mod_1, mod_2, mod_3, and top-level model mod_5. Second hierarchy is composed of intermediate models mod_1, mod_2, and top-level model mod_4.

3.1.8 Integrating social interaction

We describe here different ways to integrate social interaction to those hierarchical learning architectures. The following one is inspired from the SGIM-ACTS learning architecture [Nguyen and Oudeyer, 2012]. In SGIM-ACTS, the learning progress from each strategy are estimated. The agent can learn autonomously, or learn in interaction. When learning by interaction, the agent also learns from who to imitate, if teachers of different skills are available. For each model, we could thus estimate the interest of asking for teacher input (sensori tasks) on that model. When the agent explores a given model, it then first chooses by which strategy to explore it: self-generation of goals, or teacher input, and from which teacher to get the input that will serve as socially guided goal to

imitate.

Another possibility is to merge the input from social peers with the interest distribution of the models. When generating a goal to explore for a given model, the interest distribution of the model is merged with the top-down drive interest distributions from higher models plus social interest given as a another distribution.

Also, if the social input is not specified at the goal spaces level, but more abstract, it could be integrated at the level of the choice of model or choice of hierarchy. When choosing which hierarchy (in Algo. 8 in the Multi-Armed-Bandit) or which model (in Algo. of Section 3.1.7) to explore, the agent could merge the interest with a social value or reinforcement for those hierarchies or models. The implementation and evaluation of those different algorithms remains to be done.

3.2 Measures

3.2.1 Exploration

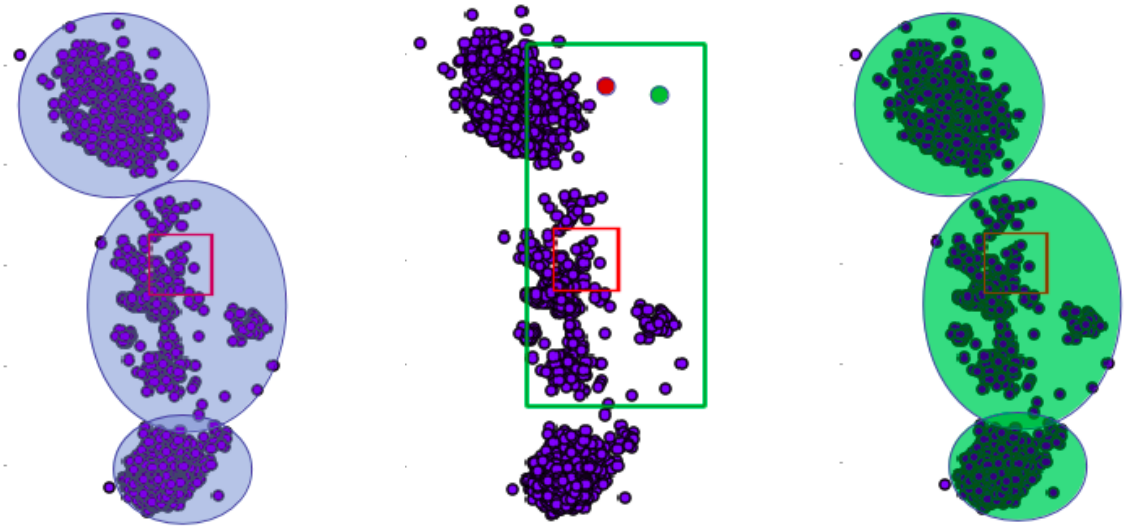
Exploration is a measure of how well the agent has found diverse outcomes in its task space. It makes no assumptions about the interest of exploring given regions of the task space from the viewpoint of the engineer, neither if we want the robot to learn specific tasks or if we know that certain tasks will be more useful later for the robot. To compute the quantity of exploration we consider the task space as a grid with small cells and count the number of cells that have been reached.

3.2.2 Competence

Given a task region where we want to evaluate the competence of the agent, we draw random goals in that region and measure the distance of the point reached from that goal.

3.2.3 Competent exploration

As the agent self-generates goals in the task space, it might not yet have explored that evaluated part of the environment, or in another run, have explored it very early, so it is hard to interpret this competence as a general progress. We thus define an intermediate measure that evaluates how well competent is the robot on the explored part of the task space. The competent exploration is thus the number of cells explored (as in the exploration measure) but that the robot can reach again, with some margin, when asked for it.



(a) (b) (c)

Figure 3: Measures of exploration. Blue points are 2D sensori points reached during exploration. (a) Exploration measures approximately the reached area (approx. the blue areas). (b) Competence measures the ability to reach random goals in a task area (green rectangle) e.g. the green point is a goal, the red point is the reached position, leading to a high error for that goal. (c) Competent exploration measures the ability to reach goals in the explored area (approx. green areas).

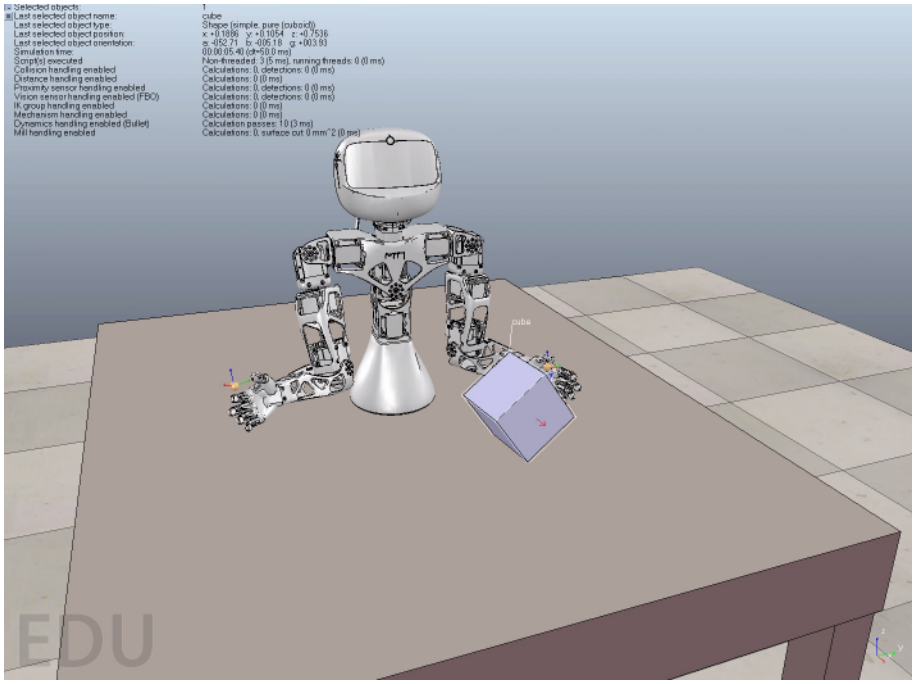
3.3 Environments

3.3.1 V-rep simulated physical arm

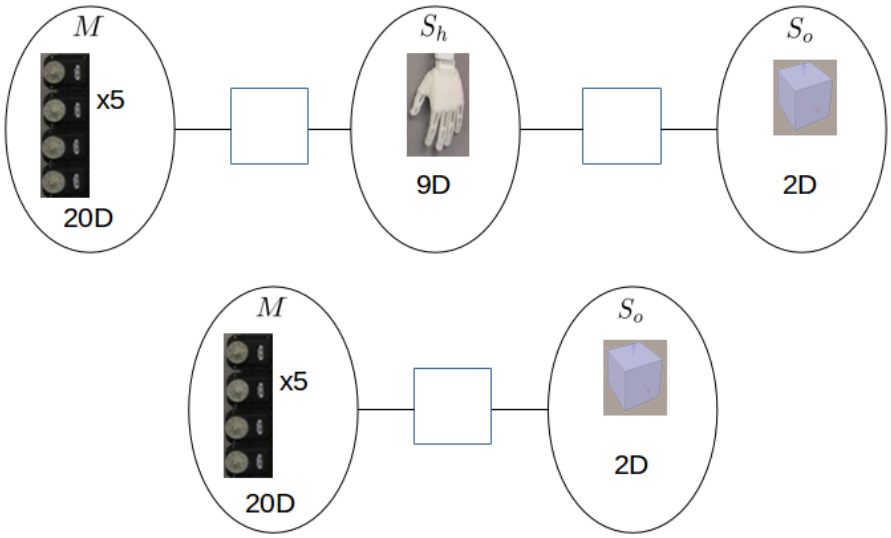
The experimental setup is composed of a robotic arm with 4 degrees of freedom from the Poppy robot [Lapeyre et al., 2014] simulated with the V-REP simulator based on the Bullet physics engine. We use Dynamical Movement Primitive [Ijspeert et al., 2013] to control the arm's movement as this framework permits the production of a diversity of arm's trajectories with few parameters. Each arm's degree of freedom (DOF) is controlled by a DMP with a starting and a goal position equal to the rest position of the motor. Each DMP is parameterized by one weight on each of 5 basis functions whose centers are distributed homogeneously throughout the movement of duration $4s$. The weights are bounded in the interval [-400,400] which allow each motor to cover its standard angle interval [-1,1] during the movement. Each DMP outputs a series of angle positions that represents a sampling of the trajectory of one motor during the movement. A block is placed near the robot's hand and can be moved in two dimensions in different complex ways, e.g. with the hand pushing on the top of the block or on a side, in one or two steps. We voluntarily kept the number of basis functions small for the agent to learn more easily the trajectory space but still being able to move the object in different complex ways.

Let M be the $20D$ space of the motor parameters and S_h the $9D$ space representing the movement of the hand encoded by the projection of the $3D$ movement of the hand on DMPs with 3 basis functions. S_o is the $2D$ space representing the position of the object at the end of the simulation. As we are interested in how an architecture has explored the different locations where the object can be pushed to, we define S_o as the goal space where exploration will be evaluated. Thus, to evaluate the exploration of an architecture we compute the number of reached cells in a grid of $2cm$ width. And to measure the competent exploration, we draw one sample in each explored cell of the object space and use its sensorimotor model to infer a motor configuration that leads the object close to

the given object goal position. We then run the movement in the simulator and check if the object has been moved less than $2cm$ away from the goal.



(a)



(b)

Figure 4: (a) Poppy Torso in the V-Rep simulator moving a block. (b) Top: Hierarchical architecture. Bottom: Control architecture.

3.3.2 Simple geometric arm

A simple geometric arm is defined as a 3 DOF arm with a total length of 1 unit, and the first part of length 4/7 units, the second 2/7 and the last 1/7. The sensori space has 2 dimensions being the X and Y position of the end of the arm.

We also define new sensori objects in this environment as sliders that output a linear value between 0 and 1 if the end of the arm is close to the slider, and 0 if it is at a distance greater than 0.15. For

instance, in Fig. 5, in (a) the 2 sliders output 0, and in (b), the first slider outputs 0 and the second outputs 0.8.

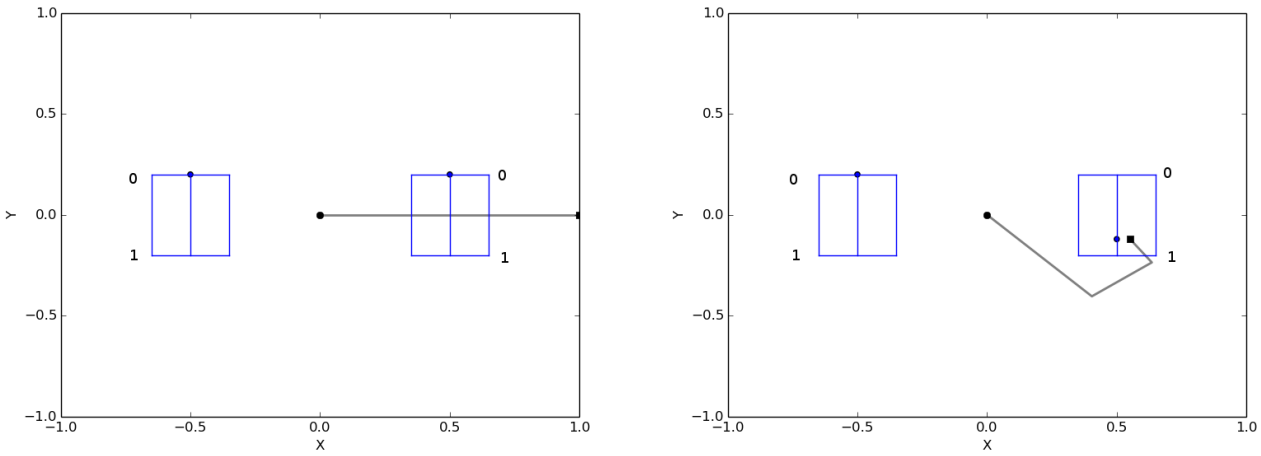
This environment will also be used in a sequence of 2 arm positions, with a new sensori value s_3 depending on s_1 and s_2 at the 2 timesteps:

$$r1 = (s_1\_1 + s_2\_1)/2$$

$$r2 = (s_1\_2 + s_2\_2)/2$$

$$s3 = (r1 + r2)/2 \; if \; r1 > 0 \; and \; r2 > 0 \; else \; 0$$

The new value s_3 then outputs a strictly positive value only if the arm has touched the 2 sliders in sequence.



(a) (b)

Figure 5: 2 positions of the simple arm and corresponding values on the 2 sliders.

3.3.3 The DIVA synthesizer

The DIVA synthesizer is a vocal synthesizer that outputs the formants of the sounds produced given the positions on 13 motor values. As in [Moulin-Frier et al., 2014a], we use only the first 7 articulators, set the articulators 8 to 10 to 1 to ensure phonation, and articulators 11 to 13 to 0. We keep only the first and second formants as sensori output as they encode relatively well the vowels produces.

We also add a slider as in the arm environment, as a third output value, between the sounds /o/ and /y/ with margin 0.3 octave.

A combined environment is also defined where the arm with one slider s_1 and the vocal synthesizer with one slider s_2 are put together. A third output value is then computed as

$$s_3 = 4 \times (s_1 - 0.5) \times (s_2 - 0.5) \; if \; s_1 > 0.5 \; and \; s_2 > 0.5 \; else \; 0$$

3.3.4 Combination of arm and vocal movements

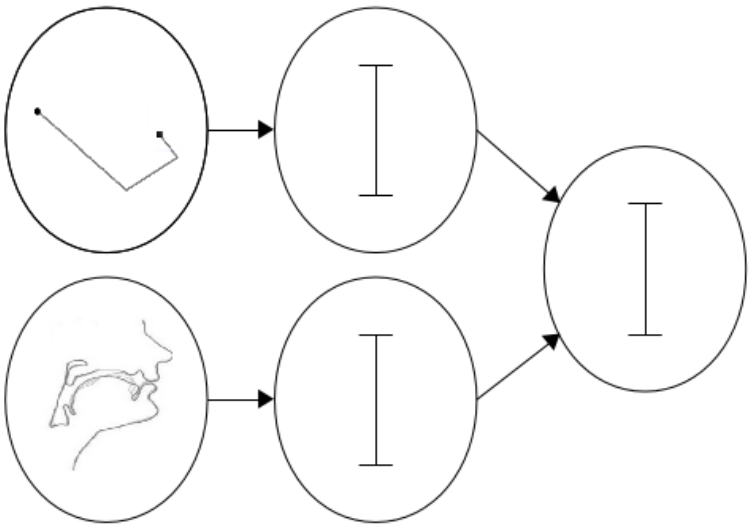


Figure 6: Multi-modal environment

3.4 Experiments

3.4.1 V-rep arm pushing a block

In order to compare the first hierarchical architecture implemented, we test them on a simple hierarchical task where the robot arm simulated in V-rep has to push the block to different locations. We compare here the flat control architectures, the Simplest First, and the Top-Down guiding architectures, with goal babbling and SAGG-RIAC-like interest models. We measure the exploration and the competent exploration with a grid size of $0.02m$.

3.4.2 Sequence of actions

In order to compare the hierarchical architecture in a more complex hierarchical task with 2 lower models to learn before learning the high one, we use simple arm environment with a sequence of 2 actions and the new output value s_3 depending on the value of the 2 sliders in the 2 actions. In hierarchical architectures, the first model learns a mapping between the first 3D M action and the value on the first slider touched, the second model learns a mapping between the second 3D M action and the value on the second slider touched, and the high third model learns a mapping between the 2 sliders touched values and the s_3 new value. Flat architectures learn a mapping between the 2 3D actions and the 2 sliders and the new value, so a model of $M^2S_1S_2S_3$. We compare here the flat control architectures, the Simplest First, and the Top-Down guiding architectures, with goal babbling and SAGG-RIAC-like interest models. We measure the exploration on S_3 with a grid size of $0.001m$.

3.4.3 Combination of arm and vocal movements

In a similar mathematical problem but with a multi-modal combination of actions, we use the environment composed of a simple arm and a slider plus the DIVA synthesizer plus its slider. In hierarchical architectures, the first model learns a mapping between the first 3D M action and the value on the slider, the second model learns a mapping between the DIVA motor space and the value on the slider in formant space, and the high third model learns a mapping between those 2 sliders values and the s_3 new value. Flat architectures learn a mapping between the multi-modal

10D action and the 2 sliders and the new value, so a model of $M_1 M_2 S_1 S_2 S_3$. We compare here the flat control architectures, the Simplest First, and the Top-Down guiding architectures, with goal babbling and SAGG-RIAC-like interest models. We measure the exploration on S_3 with a grid size of $0.001m$.

4 Results

4.1 V-rep arm pushing a block

Preliminary results show comparable amounts of exploration for hierarchical and flat architectures with the same SAGG-RIAC interest models (Fig. 7). MS1 means that one single model learns the mapping between M and S_h. MS2 means that one single model learns the mapping between M and S_o (flat control architecture). S1S2 means that one single model learns the mapping between S_h and S_o. SEQ means that a model learns MS_h in the first half and an other learns $S_h S_o$ in the second half of the iterations (Simplest first strategy). TOP-DOWN-CMA means that the CMA evolution strategy is used in the model MS_h to try to reach a hand movement s_h chosen by the model $S_h S_o$. TOP-DOWN-RANDOM means that around the s_h chosen by $S_h S_o$, the first model makes 5 random explorations (taken into account in the 10,000 iterations).

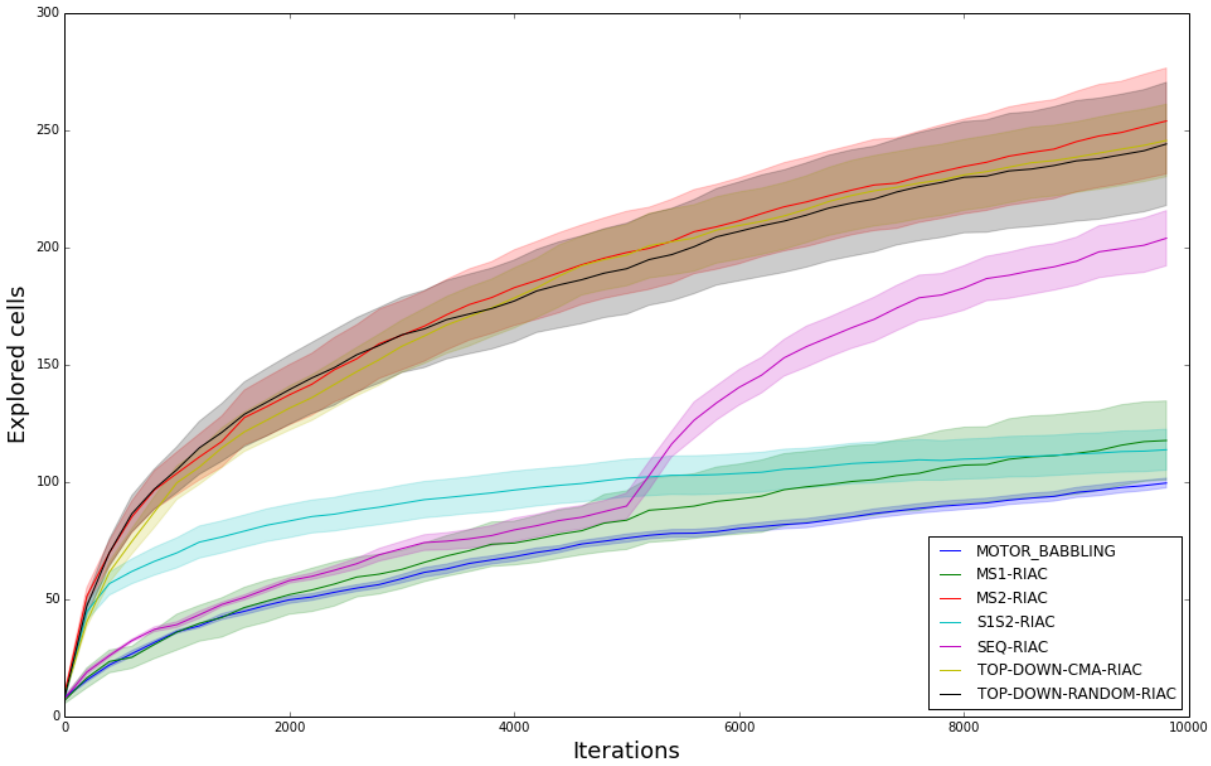
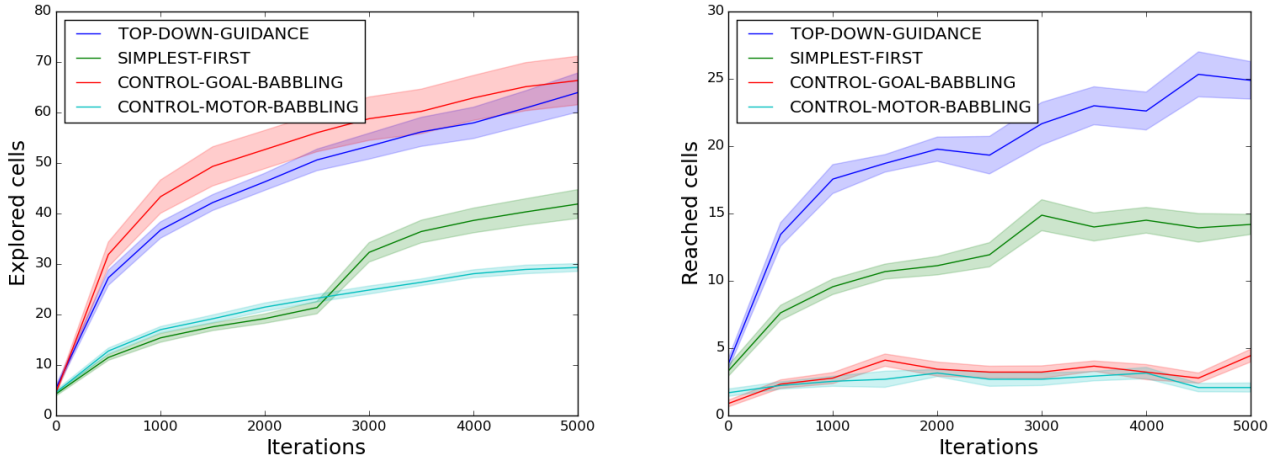


Figure 7: Exploration in the object's space S_o using the SAGG-RIAC interest model. Mann-Whitney U tests at the end of the experiments show that the exploration measures in the MS2, SEQ and TOP-DOWN conditions are significatively greater ($p < 0.05$) than in the MOTOR-BABBLING, MS1 and S1S2 conditions.

As a very high variability results of the collision between the hand and the object, only a few explored cells, were in fact competent, in other words when we ask the robot to go to those cells,it succeeds in only a few. We measured the exploration together with the competent exploration in

another run with a grid of size 0.1 (Fig. 8). The competent measure is the number of cells where the robot succeeds in putting again the block at that place.

Mann-Whitney U tests at the end of the experiments show that the exploration measures in the CONTROL-GOAL-BABBLING and TOP-DOWN-GUIDANCE conditions are significatively greater ($p < 0.5$) than in the SEQ condition, where it is significatively greater than in the CONTROL-MOTOR-BABBLING condition. On the other hand, tests on the competent exploration measure show that it is significatively greater in the TOP-DOWN-GUIDANCE condition than in the SEQ condition where it is significatively greater than in the CONTROL conditions.



(a) (b)

Figure 8: (a) Exploration. (b) Competent exploration. CONTROL-MOTOR-BABBLING: random motor parameters in M are explored for all iterations. CONTROL-GOAL-BABBLING: one single model learns the mapping between M and S_o. SEQ: a model learns MS_h in the first half and an other learns S_hS_o in the second half of the iterations (Simplest first strategy). TOP-DOWN-GUIDANCE: the CMA evolution strategy is used in the model MS_h to try to reach a hand movement s_h chosen by the model S_hS_o.

Results show comparable amounts of exploration for the top-down guidance and the control (goal babbling) conditions, whereas the competent exploration measure shows that the architectures learning an intermediate hand's movement representation allow the agent to put again the object on much more diverse locations. The intuition for this result is that when exploring around given motor parameters, the control architecture will modify some motor's angle trajectories and will produce more unstable interactions with the object than a hierarchical architecture that modify directly hand's Cartesian trajectory. A posssibilty to test this hypothesis is to choose an explored object's position s_o in the condition TOP-DOWN-GUIDANCE, then infer a hand movement s_h and then a motor set of parameters m that should lead close to s_o. If the hypothesis is true, then given a variation m' of m, and an infered m'' of a variation s'_h of s_h, the variance of the reached object position when repeating m' should be higher than when repeating m''.

Fig. 9 shows the diversity of explored block position for the same experiment but several independant trials. This illustrates why measuring the competence (Section 3.2.2) does not make sense on a fixed subregion of the goal space as different runs do not explore the same parts of the space.

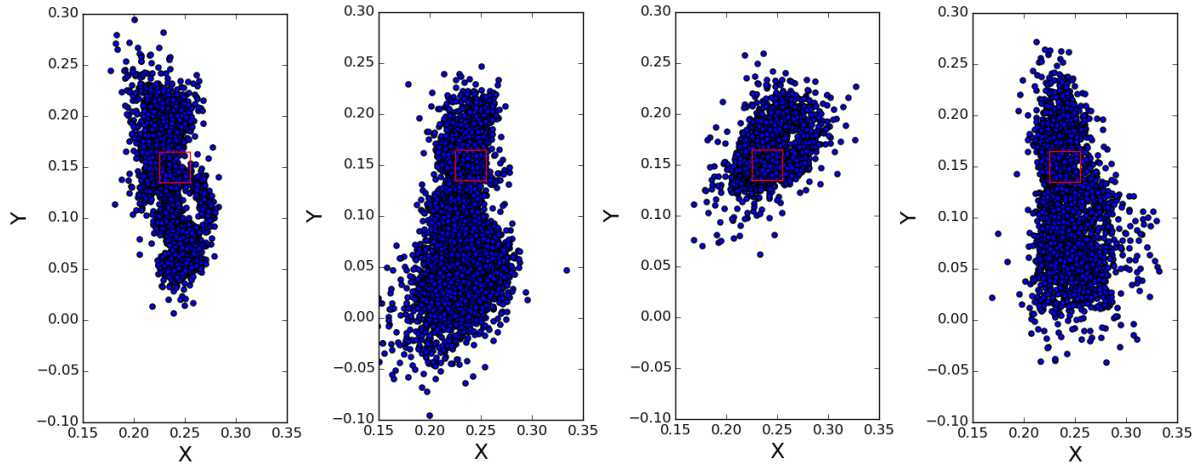


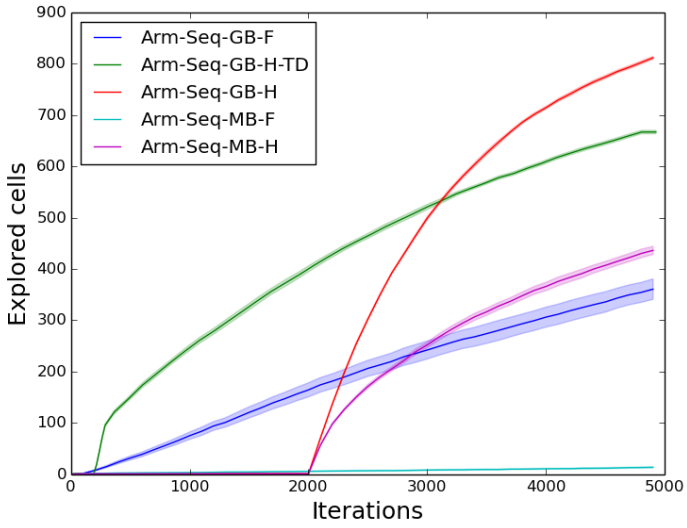
Figure 9: Variability in exploration in 2D space for several trials of condition TOP-DOWN-CMA-tree (SAGG-RIAC). Red square: initial position of the block.

4.2 Sequence of actions

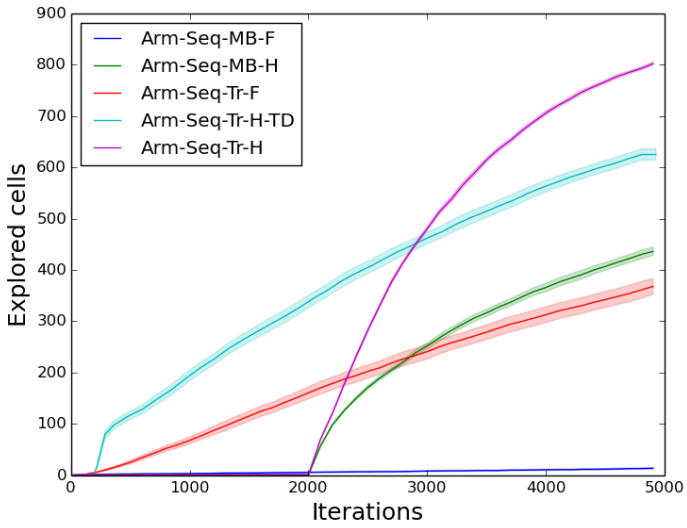
Fig. 10 shows the results of the second experiment, with a sequence of 2 arm movements. Results show that the hierarchical Simplest First architecture explores better than the hierarchical Top-Down approach, which explores better than the flat architectures, either with goal babbling or SAGG-RIAC interest models. The motor babbling Simplest First hierarchical architecture explores also better than flat architectures.

4.3 Combination of arm and vocal movements

Fig. 11 shows the exploration in the multi-modal arm plus vocal synthesizer environment. Results show that the hierarchical Simplest First architecture explores better than the flat architectures, either with goal babbling or SAGG-RIAC interest models. The motor babbling Simplest First hierarchical architecture explores less than flat architectures.



(a)



(b)

Figure 10: Exploration. (a) Goal Babbling versus Motor Babbling. (b) SAGG-RIAC versus Motor Babbling. GB: goal babbling is used to explore models. MB: motor babbling is used. Tr: SAGG-RIAC interest model. F: flat architecture with 1 model. H: hierarchical architecture with the Simplest First strategy with 3 models. GB-H-TD: GB is used to explore the high model, and TOP-DOWN-RANDOM to guide the 2 lower models. Tr-H-TD: SAGG-RIAC is used to explore the high model, and TOP-DOWN-RANDOM to guide the 2 lower models. Mann-Whitney U tests at the end of the experiments show that the exploration measures in the GB-H and Tr-H conditions are significatively greater ($p < 0.001$) than in the GB-H-TD condition where it is significatively greater ($p < 0.01$) than in the Tr-H-TD condition, where it is significatively greater ($p < 0.001$) than in the MB-H condition, where it is significatively greater ($p < 0.01$) than in the Tr-F and GB-F conditions, where it is significatively greater ($p < 0.001$) than in the MB-F condition.

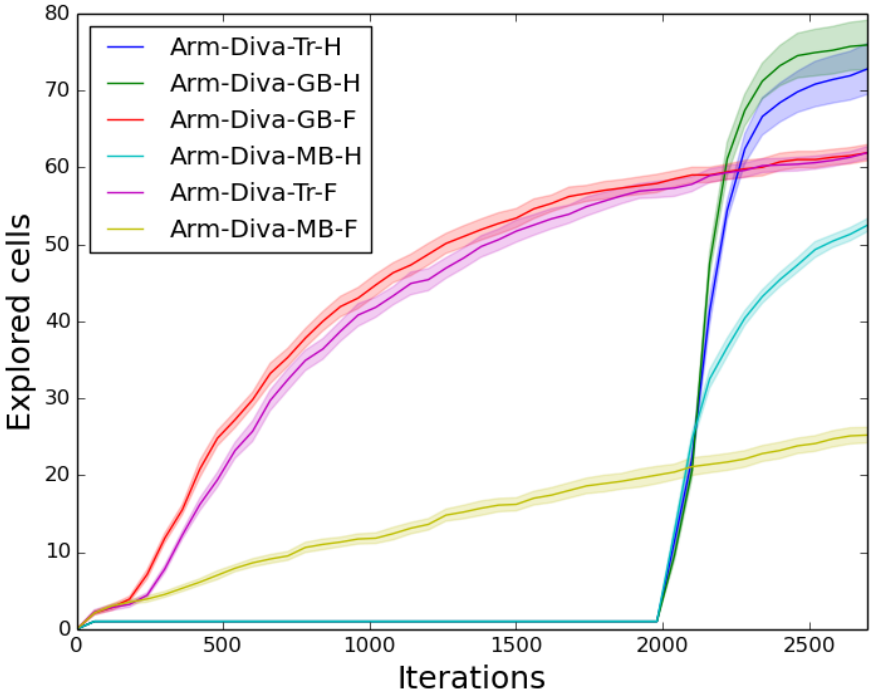


Figure 11: Exploration in the multi-modal environment. GB: goal babbling is used to explore models. MB: motor babbling is used. Tr: SAGG-RIAC interest model. F: flat architecture with 1 model. H: hierarchical architecture with the Simplest First strategy with 3 models. Mann-Whitney U tests at the end of the experiments show that the exploration measures in the Tr-H and GB-H conditions are significatively greater ($p < 0.05$) than in the GB-F and Tr-F conditions, where it is significatively greater ($p < 0.01$) than in the MB-H condition, where it is significatively greater ($p < 0.001$) than in the MB-F condition.

5 Discussion

The hierarchical architectures composed of 2 models tested in the V-rep simulator show equivalent exploration results but better competent exploration than flat architectures.

The results on the environment composed of a sequence of 2 simple arm commands and the multi-modal environment with an arm and a vocal synthesizer show that our exploration architectures get more useful as the hierarchy of sensorimotor models gets more complex.

The development of a parallel combination operator that allows to run different actions at the same time is not straightforward as a motor could receive two commands at the same time. In that case, a first idea is to randomly choose one of the commands. An interesting approach could make use of Probabilistic Movement Primitives [Paraschos et al., 2013] that represent distributions of probabilities of movements on the different articulators. PMPs are shown to handle easily sequence and parallel combinations of movements as operations on distributions of probabilities, for instance the product of probability distributions for the parallel operator.

In the architectures developed (e.g. Section 3.1.6), the sensori spaces represent environmental states that are received as input to the agent at the end of the executed actions. In order to handle

environments with hidden states, where sequences of actions have combinatorial effects, it should be beneficial to take the context of the action into account, thus having contextual sensorimotor models relating an action A in a sensori context S to the next sensori state S'. The dimensionalities would then be much higher, which might require dimensionality reduction techniques.

6 Graph implementation

I'm trying to implement a first version of a graph representation of multiple hierarchies' learning. Before that, I have to be sure about the behavior of the SAGG-RIAC algorithm and its parameters as it will be used as a sub-component in the multi-level graph representation.

6.1 SAGG-RIAC

Here I want to study the important parameters of the SAGG-RIAC implementation (from Explauto, with my implementation of RIAC in explauto.interest_model.tree) and test them first on very simple environments. The goal is to get a sound progress output, that could be compared with other models' progresses.

Parameters

- sensorimotor model (KNN -default-, WNN, LWLR)
- exploration noise (gaussian noise amplitude -default 1/30-)
- competence measure ($-distance$ or $exp(-distance)$, distance min -default=0.01-)
- progress measure (initial progress value, progress time-scale)
- leaf sampling mode (greedy, eps-greedy, softmax -default-), weighting by volume, no multi-scale sampling here
- constaints on tree: depth -default=6-, max number of points per region -default=100-
- split mode (middle -default-, median, best interest difference)

Questions

- how does the progress measure evolve for very simple forward models ? (constant, piece-wise, linear, random)
- does progress change when split occurs ? (weighting by volume ?)

First Tests

- **Constant forward model**. We have one motor dimension $m \in [0, 1]$ and one sensory dimension always equal to 0 ($\in [0, 1]$). With default RIAC settings, the progress measure does not converge to 0, even if the distance min of the competence measure is not 0 (typically 0.01, which means that if the competence is better than 0.01 in the region, that region will become non-interesting). The reason seems to be that given a region not containing 0 (e.g. $s \in [0.5 - 0.75]$), all the drawn goals will have a low competence but varying between the competence of distances 0.5 to 0.75, with no amelioration with time. One idea to fix this problem is to set a maximum goal-to-reached distance (e.g. the size of the region) corresponding to a

minimum competence, which will be the always obtained competence in our example, with no fluctuating progress. This idea works in this example, and is beneficial for the same reasons in the case of a **Random forward model**. However, in the case of a non-stationary forward model, a new kinematics too far from the learned one might not be computed as interesting (it might be at the beginning due to the time window). EDIT: That does not prevent adaptation to new forward models (See Fig. 15).

- **Linear forward model**. We have one motor dimension $m \in [0, 1]$ and one sensory dimension $s = m$. The progress fluctuates between 0.1 and 1 for some hundreds iterations (learning phase) and then fluctuates at very low values ($10^{-2\ to\ -5}$) (stochastic changes). Setting a minimal distance in the competence measure (e.g. 0.01) makes the progress go to stricly 0 instead.

- Does progress change when split occurs ? In order to have progress outputs comparable as possible between learning modules, we should have a progress value that depends only on the new incoming data and not on a change in data representation. In the SAGG-RIAC algorithm, a split is decided when a region has too much points (> 100 here). The progress of the 2 sub-regions is then recomputed with the competences of the 2 sub-sets of the points, separately. Assuming that the 2 sub-regions and the parent one have similar progresses, we have now 2 leaves with the same progress. In the sampling process, when we choose the leaf from which to sample a point with a softmax function applied to the interests, then the 2 leaves will weigh 2 times more than the parent one, while representing the same region. It might be interesting to explore more in splitted regions that need resampling, but that leads to a focuse on such regions that will thus be resplitted and so on, with too deep trees. I thus included a possibility to weight the interest of regions by their volume, which allow a more stable sampling process.

6.2 Hierarchical exploration

I implemented the hierarchical exploration of models with a model-first sampling (instead of a hierarchy-first). That mean I consider all the models to learn, see their learning progress, and sample one model with high progress (with a softmax of relatively low temperature -0.5-). Then that model goal babbles, chooses a motor configuration to try (add random noise), which can be in the sensory space of a lower-model level, and execute the motor command (if no lower-level dependency) or give that goal to lower-level models to be executed now (with no exploration iterations budget, but allowing motor noise in lower-level models). A motor command m is finally executed, and a sensory input s observed. Each model is then updated with the dimensions of ms it is concerned with, which can be different from what it asked to lower-level models. I also implemented the possibility to add an exploration budget around the top-down guided point, but I don't use it here as results are not clear.

Parameters

- model sampling (greedy, softmax -default-)

- sampling models (-default-) or first sampling the hierarchy to train (root of the DAG) and then the model in the hierarchy.

- add top-down guided exploration budget

Questions

- In a simple hierarchy, do we see expected exploration behaviors ? e.g. if a model is harder to learn, is it babbling more often ? If a model is reusing another one, does a learning schedule autonomously emerge with the higher-level model explored significantly after ?
- For non-stationary forward models, the sensorimotor model does not forget old points. Should we weight also by time ?

First Tests

I use here identity function as forward models. There are 2 low-level models (mod1 and mod2) and one higher level that only rely upon mod1. Thus $s1 = m1, s2 = m2, s3 = s1$. In a second condition, the forward model of mod3 is changed at 5000 iterations with $s3 = s1^2$.

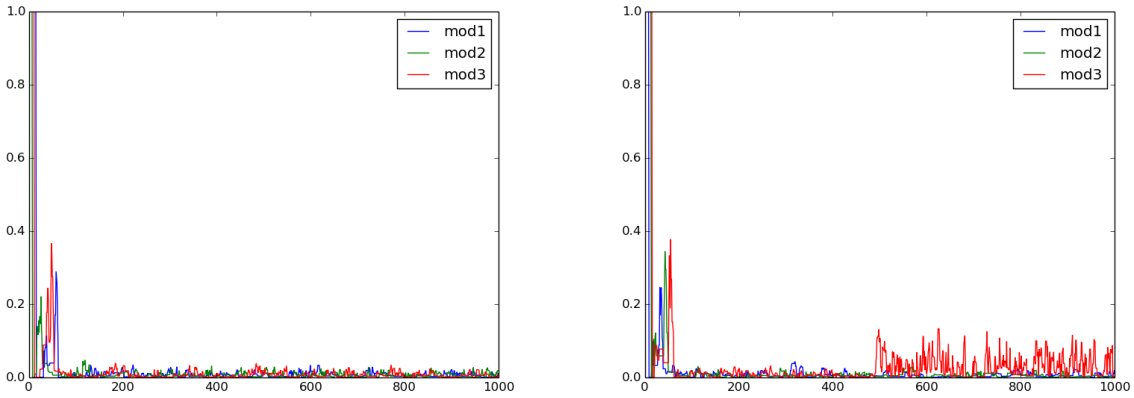


Figure 12: Exploration

Remarks

- In the first condition, the 3 modules learn quickly their function, and then alternate. In tests with more complex functions including random noise, the results are less clear (depend on max distance).
- In the second condition, at 5000, the 3rd module is seeing an increase in its progress but is not learning the change in its forward model. This is due to the problem described before, of non forgetting old sensorimotor points.
- In tests with a more complex hierarchy (7 models with some of them reusing several others), and a change in a given forward model, the results are similar.
- The progress or interest of a SAGG-RIAC tree is here computed as the progress on the last k (here $k = 10$) babbling points on the overall tree, taking the points of all leaves into account (default). Another way to compute the progress is to get the maximum of the progress of each leaf. This does not change qualitatively the results.

6.3 More on Non-Stationary Forward Models

To cope with non-stationary forward models, with have to somehow forget old points and consider more the newest ones. The linear regression in LWLR is weighted by the distance of the points in

the dataset. We can also weight them according to their timestamps (iterations at which the points were added). Given x, we want to compute $f(x)$, using data points $\{x_i, y_i\}$. First idea: we compute the weights of the Locally-Weighted Linear Regression with:

$$w_i = K_g(dist(x_i, x), \sigma^2) * ((iteration(i) - min\_iteration + 1)/max\_iteration)^2$$

where K_g is a gaussian kernel with $\sigma = 0.05$.

The results (Fig. 13) do not show a stabilization of the progress towards 0 after the change in forward model at iteration 500 for module 3.

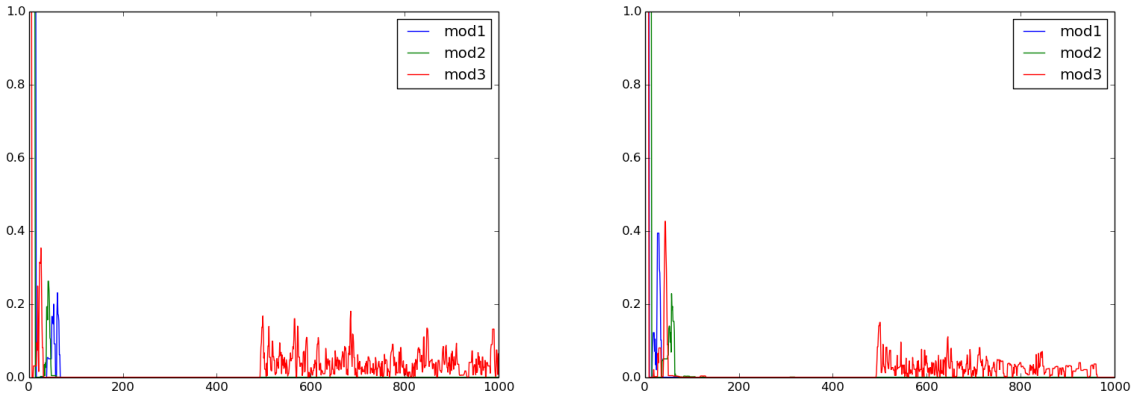


Figure 13: Progress of the 3 modules. The weights of the LWLR regression are weight with iterations' timestamps. Left: k=5 points used in regression, Right: k=20 points. Progress is measured as the max of the progress of each leaf.

Second idea: we use a gaussian kernel also for time weighting (see Fig. 14, left).

$$w_i = K_g(dist(x_i, x), \sigma^2) * K_g(current\_iteration - iteration(i),\ \sigma_t^2)$$

with $\sigma_t = 100$ iterations.

The result is shown in Fig. 14 left. That was neither a fast adaptation so I closer looked into the progress depending on the leaves. I saw that the non-zero progress was due to the initial guess point of the inverse optimization (with algo L-BFGS-B) that was set as the x corresponding to the nearest neighbor of the goal y_g in the dataset. After the change in forward model, the guess point is often corresponding to the old forward model, and the optimization starting with that points is often failing, returning uncertain or 0 values giving a not real progress to the leaf. I changed the number of initial guess points (goal's nearest neighbors) to try with inverse optimization to $k = 10$ (Fig. 14 right). Now, if points corresponding to the new forward model do exist near the goal, they are tested and optimization fail much seldom. However, the computation time gets too long.

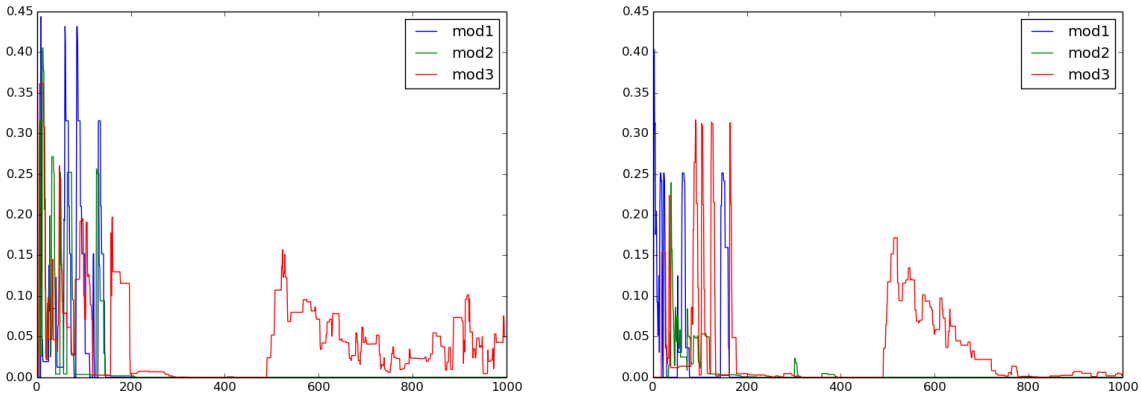


Figure 14: Progress of the 3 modules. The weights of the LWLR regression are weight with a gaussian time kernel. Left: 1 point is used as initial guess. Right: 10 points are used. Progress is measured as the progress on the last points of the global tree.

An idea is to try only different points (ideally one old point and one new point) from the dataset as initial guess to the inverse optimization. I did that with a k-means clustering of the 50 nearest neighbors of the goal in the dataset, to get 2 clusters. The inverse optimization is feeded with the 2 clusters' centroids as initial guess. The results (Fig. 15) show the same adaptation as the $k = 10$ guess points (Fig. 14 right) but is much faster.

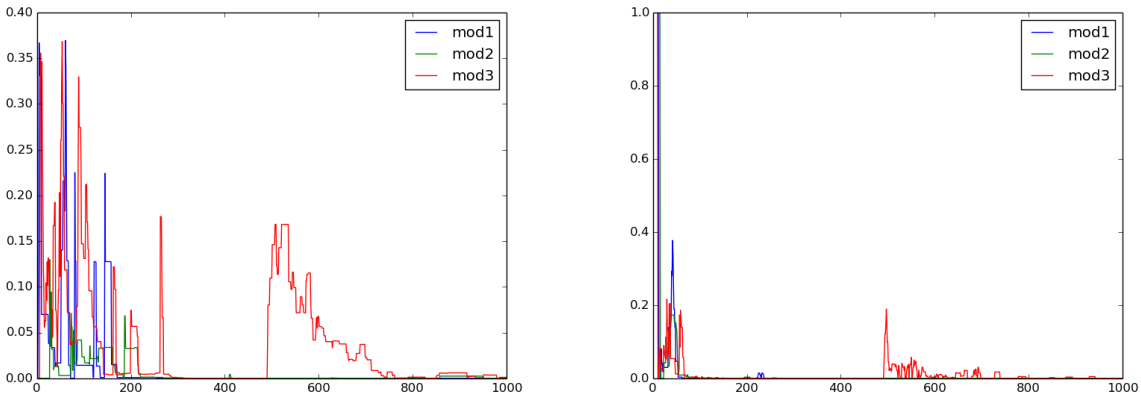


Figure 15: K-means clustering of 50 initial guesses in 2 clusters to try inverse optimization. Left: Progress is measured as the progress on the last points of the global tree. Right: Progress is measured as the max of the progress of each leaf.

There is still some noise due to BFGS failed optimizations that sometimes sticks to the domain bounds, which gives competence noise in the SAGG-RIAC tree's lowest leaf (e.g. with bounds [0-0.125] -tree max depth is 3 here). The adaptation is satisfying, given that the time kernel size is of 100 iterations. In Fig. 16, the gaussian time kernel has size 20 iterations, and different forward model's transitions are tested. The transition from a harder to a simpler forward model seems to need longer adaptation, but it might not be significant.

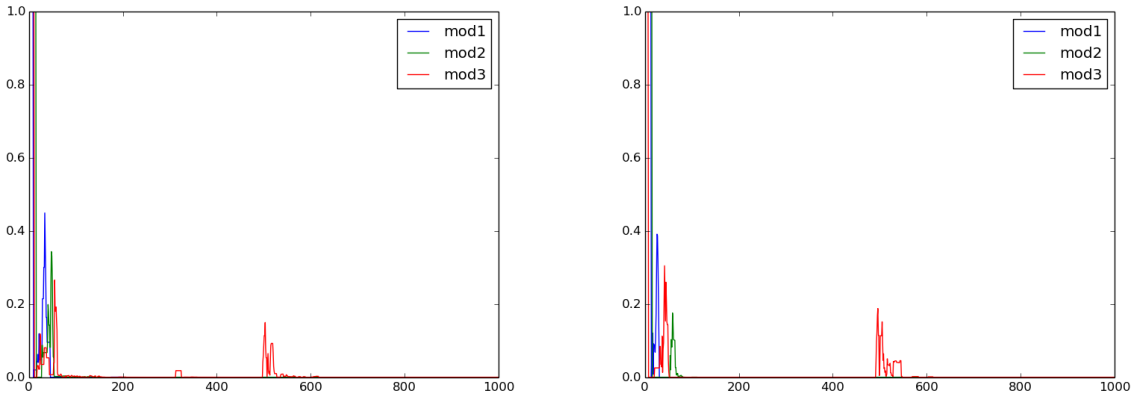


Figure 16: Progress on the 3 modules. LWLR weighted with a gaussian time kernel of size 20 iterations. Left: forward model of module 3 moves from $x \to x^2$ to $x \to x$. Right: forward model of module 3 moves from $x \to x$ to $x \to x^2$.

When one module see a change in its forward model, all the models that reuse it will not have their sensorimotor model modified, but will have their interest model perturbated. For instance, let's say module 1 learns the mapping from m_1 to s_1 and becomes wrong, and module 2 learns a mapping from s_1 to s_2 and reuses module 1. Then when module 2 is babbling, it wants to produce s_2 so infers a s_1 to try, and feeds it to module 1 that fails at producing s_1 but produces instead s'_1 that leads to s'_2 far from s_2. Thus both modules see a decrease in competence and thus an increase in interest.

As all models reusing the perturbated one gets a perturbated interest (interest that do not actually correspond to a change in the competence of the model's mapping), thus a lot of exploration iterations might be wasted on those models while they do not necessarily need training. With 4 models in a linear hierarchy where model $i+1$ reuses model i, Fig. 17 left shows the results after a change in the forward model of module 1 at iteration 200. Indeed, the 4 modules have a non-zero interest and are sampled, from iterations 200 to 300, whereas only module 1 needs sampling. An idea to avoid this waste of exploration is to weight the interest of each module (before a softmax choice) depending on the level of modules in the hierarchy. I used the following weights in Fig. 17 right:

$$w_{mod_i} = p_i * f^{max\_level - level(i)}$$

where p_i is the progress of module i, f a power factor, $level(i)$ is the level of module i in the hierarchy: from 0 for module 1 to 3 for module 4.

We can see in Fig. 17 right that even if modules 3 and 4 have sampled a few points between iterations 225 and 400, most of the babbling is done by module 1 that corrects its own sensorimotor model. This idea to first train on more basic skills seems ecological and can be applied to more complex hierarchies, and can also be combined with other mechanisms to choose modules to train.

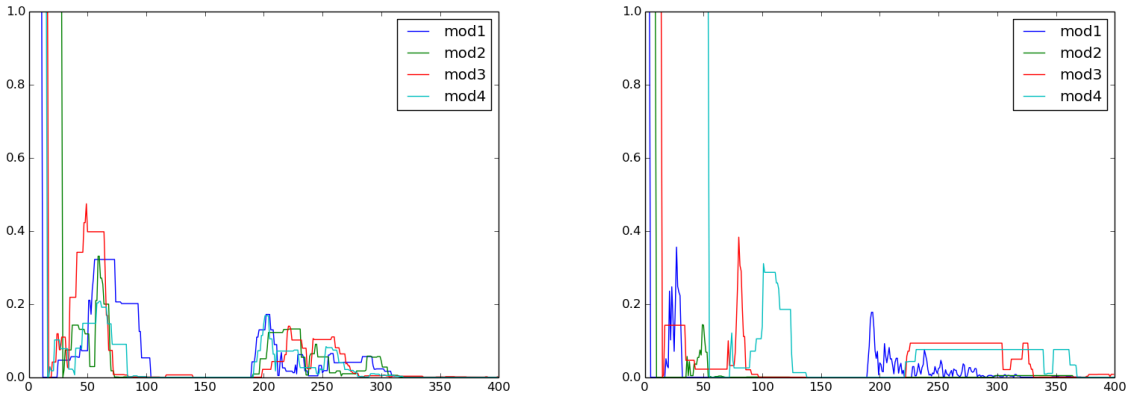


Figure 17: Progress on the 4 modules of a linear hierarchy. Left: $f = 0$. Right: $f = 10$.

6.4 Creation of modules/tasks

Here we allow the creation of modules when all the progresses of current modules are below a threshold (e.g. 1e-2).

Module creation An operator is randomly chosen among the concurrence and sequence operators. Then, 2 input connexions are chosen among the output of other modules plus primitive motor spaces. A sensory space is randomly chosen as a subset of all possible sensory spaces. In case of the concurrence operator, there are constraints on the input and output spaces: that the 2 input's set of controled variables must have a void intersection, and that the motor spaces can't be in the sensory spaces. We could remove the constraints and let the agent understand that this module might be useless, but it could be a waste of iterations.

In the following experiments (see Fig. 18), we use the previous linear hierarchy as initial models (from module 1 to module 4), and we use the constraints above for the concurrence operator. As we have only one motor space, the concurrence operator constraints can't be fulfilled, so the sequence operator only will be chosen.

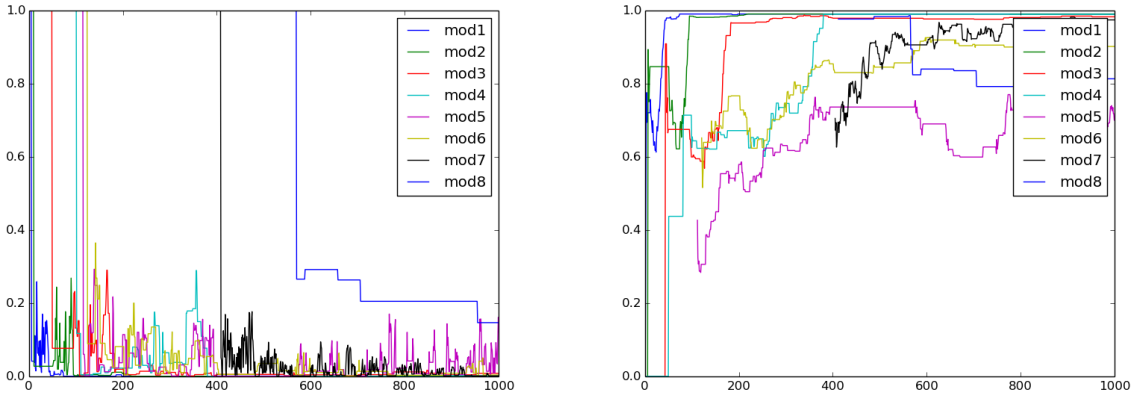


Figure 18: Left: Progress of modules. Right: IM Competence of modules: competence to reach self-generated goals, which depends on the sensorimotor competences of involved modules.

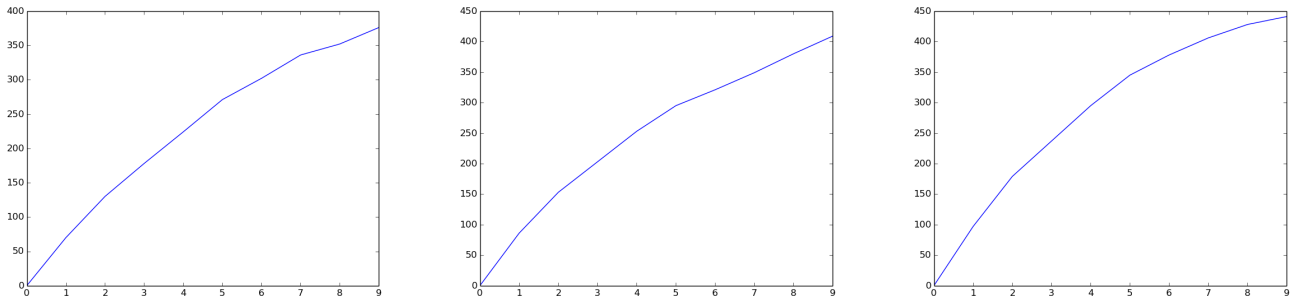
6.5 More on Top-Down Drive

The idea of Top-Down Drive is that higher modules in the hiearchy have relevant information about the utility of the exploration of the different regions of the common representation space. Different ideas might allow to take this information into account. In the experiments with the V-rep arm pushing a block (4.1), I suggested 2 possibilities (See Section 3.1.5). One is that each module asked to reach a goal, explores with a small budget of experiments around that goal before infering a motor command to reach that goal. A problem with that approach is that in a complex hierarchy, it would take to much time to infer a motor command for a goal containing a lot of nested sub-goals, furthermore as each module could receive multiple goals. The second approach, not implemented was thus to feedback a top-down interest model in the common space to lower models that will have to balance the self-computed interest map with Top-Down interest map (and even social interest map). The drawback now is that a lot of maps represented for instance with GMMs have to be handled, recomputed, multiplicated and sampled.

Weighting by the density of Top-Down goals. An other approach in-between would be to store, for each module, all the goals that have been asked from higher models in a specific KDTree (different from the SM model that contains all points, and from IM model that contains the self-generated points). The information of interest to explore a point would be retrieved as the density of points in the kdtree around the given point.

If we use SAGG-Random, instead of sampling random goals, we could directly sample the density map most of the time (one parameter from 0 to 1), and randomly sample sometime.

Fig. 19 shows the results of the exploration of a 1D space with a 2-modules hierarchy: first modules learns $s1 = m1$ and second module learns $s2 = 2 * s1$ with $m1, s1, s2 \in [0 - 1]$. The idea is that for the second module to progress, only half of the motor space ($m1 \in [0 - 0.5]$) is useful to explore. The Top-Down guidance of module 2 on module 1 thus makes it focus more on that part. The results show that the larger the TD weight, the larger the exploration of $S2 = [0 - 1]$ (divided into 500 small parts). Also, other tests show as expected that the smaller the useful area of M1 for module 2, the larger the benefit of Top-Down guidance on the exploration of S2.



(a) (b) (c)

Figure 19: (a) no Top-Down Guidance. (b) Top-Down Guidance with weight 0.5. (c) Top-Down Guidance with weight 1.

In the case we use SAGG-RIAC, and in order to balance self-computed interest with Top-Down interest, we could include the Top-Down information into the softmax choice of a leaf of the interest model's tree to explore. By default, the weights of the leaves are the competence progress on the points of each leaf. An idea is to also weight the interest of the leaves by the number of TD goals in that leaf. I've also implemented that algorithm, which shows also a benefit from SAGG-RIAC.

Taking into account the information of the error between desired and actual motor command. With the previous algorithm based on the density on Top-Down asked goals, there still exists information not handled but which might be useful. Indeed, when a module asks for a desired motor command m_d, if lower modules do not reach that goal but a m_r potentially far from m_d, then the distance $e_m = ||m_r - m_d||$ represents the error of lower models and might be useful to drive their exploration. The density idea already uses the information of the point m_d, but not the associated e_m which could lead to a quantity of progress if derived. This learning progress is exactly the learning progress computed in a leaf of the interest model of lower modules, but there it is updated only when that sub-module has babbled, not when a higher module has asked for a goal. If we were to update also the interest models when modules have not babbled, I guess that would mess up the competence derivative computation, leading to wrong estimates. So I don't know if we can use this supplementary information.

Taking into account the progress in the leaf of the remote module that has babbled. In the density algorithm above, when a higher module has asked for a goal s_g, that point is added to the density map of Top-Down goals, saying that this point is an interesting location to further explore around. But overall, some of those asked goals will lead to a progress in a leaf of the remote module that has babbled, and some of them will not. Thus it might be useful to use that information to add or remove interest in the given region of the map of Top-Down goals depending on the progress that has been remotely elicited. To this purpose, we could inspire from the framework of RL and particularly the $TD(\lambda)$ algorithm that makes use of eligibility traces that stores the influence of each state on the future reward. We could see the progress computed in the leaf that has babbled as a reward which will be added to the Top-Down interest of the leaves that have been used in the given command (with accumulating or replacing eligibility traces).

This idea is also inspired by the RiARiT algorithm that stores a table of competence levels required by each activity on each knowledge component. Each knowledge component thus knows the influence it has on each activity. In our setting, knowledge components are themselves activities, and that table would be estimated on-the-fly and at the leaf scale.

The implementation would use the mean of (possibly time-decaying) rewards in each leaf as a weight to be balanced with self-computed leaves' interests before a leaf softmax choice.

6.6 Implementation of multiple options to explore a given sensori space

I have implemented the possibility to have different modules that learn a mapping towards the same sensori space. A higher module that uses that space as a motor space will choose which lower module to use to explore a point in that space with the goal to maximize the competence around that point. I have tested that idea using SAGG-Random modules instead of SAGG-RIAC ones. I compute the progresses (to choose the module that will babble) as the derivative of the competence of last goal babbled points. The competence around a point (to choose the module that will reach the goal with the maximum precision) is computed as the mean of the competences of the $k = 20$ nearest neighbors.

The hierarchy used is the following: mod1 learns from $m1$ to $s1$, mod2 learns from $m2$ to $s1$, and mod3 learns from $s1$ to $s2$. Mod3 can thus reuse either mod1 or mod2. All spaces are 1-dimensional with range $[0 - 1]$. Here are the forward models:

$$s1 = \frac{2 * m1 + m2}{3}$$

$$s2 = s1^2$$

Thus, if mod3 reuses mod1, it will reach a wider range of outputs than with mod2. Fig. 20 shows the results of 2000 iterations with this setup. Mod3 quickly (100 iterations) gets to use mod1 (around 90% of babbling iterations) instead of mod2. In the computation of competences, I also cut the competences to a minimal competence value (or maximal error distance of 0.1) in order to avoid big random fluctuations of interest (absolute value of progress) when a module wants to explore a far unreachable point, which entails a very high interest (see Sec. 6.1).

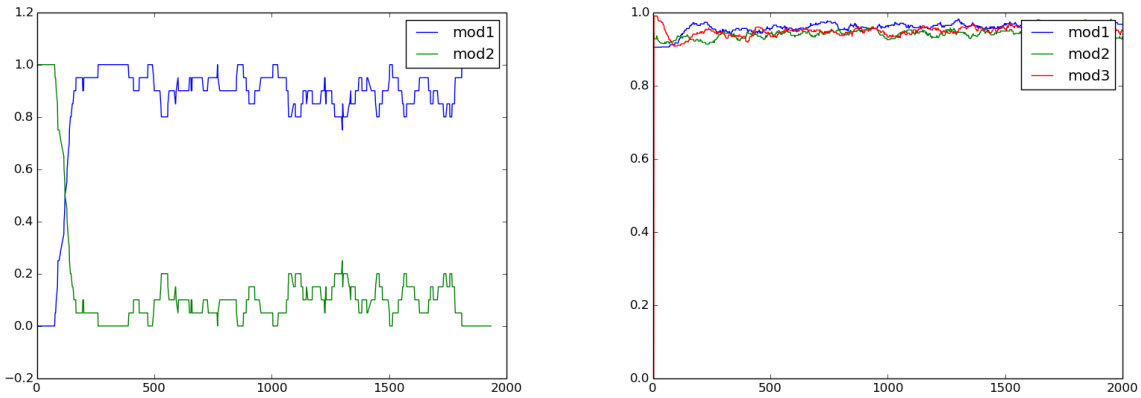


Figure 20: Progress on the 3 modules of a hierarchy with 2 choices of sub-modules for module 3. Left: Proportion of choices of module 3 (with a sliding window). Right: Competences (with a sliding window) of babbled points of each modules.

6.7 ZPDES

I've tested Benjamin's code of the ZPDES algorithm [Clement et al., 2014] to choose which module to explore at each iterations. The algorithm has mainly 4 parameters:

- The rate α of integration of new information about average reward of bandits: $bandit\_value = (1 - \alpha) * old\_bandit\_value + \alpha * reward$. I used $\alpha = 0.5$.
- The activation threshold: if the mean (recent) competence of activated modules exceeds that threshold, then a new module will be activated. I used 0.5.
- The deactivation threshold: if one module have its mean recent competence exceeding that threshold, it will be deactivated. I used 0.6 here.
- A window size to compute the means of recent competences and progress: I used 20 iterations.

See Fig. 21 for an exemple of execution of ZPDES in my setup. There are 3 modules with the same mathematical functions as forward models as in the previous section. We can see that module 1 gets deactivated early when its mean competence exceeds 0.6. Module 3 only gets activated when the mean competence of modules 1 and 2 exceeds 0.5 around 200 iterations.

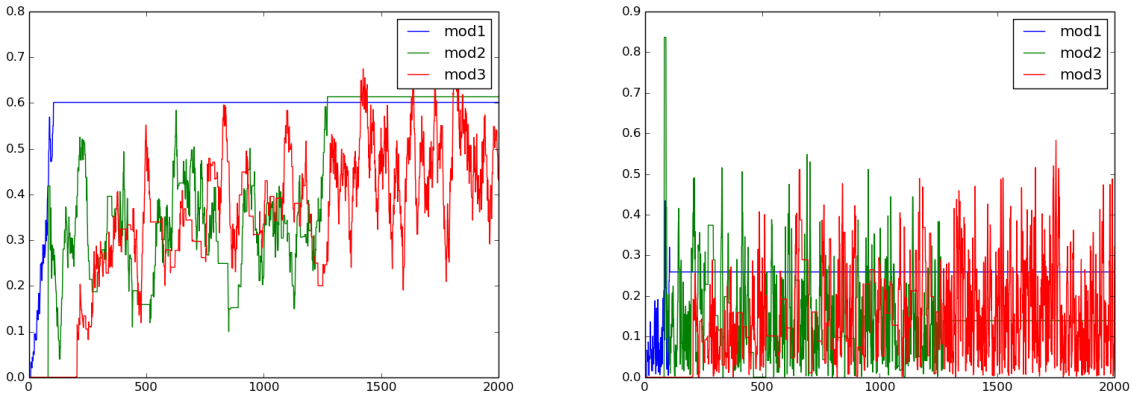


Figure 21: Competences and progresses using ZPDES.

Remarks The activation threshold allows to avoid spending useless trials on too complex tasks, provided the algorithm knows a sequence of modules sorted by increasing complexity or level in the hierarchy. Also, the deactivation threshold allows to stop sampling tasks that are already mastered. However, if a task can't be mastered, for instance if a large part of the sensori space is not reachable (which is often the case), then its competence might never exceed the deactivation threshold, which is not really a problem, and might even not exceed the activation threshold, at least if we use random goal babbling that might often set unreachable goals. That could be a problem as only non progressing modules might be activated. One could lower the activation threshold but its benefit will be lower also. If SAGG-RIAC is used, the apparent competence will be the one on sampled points in progressing subregions of the tree, thus mostly in reachable parts, so that might not be an issue.

7 Summary

We have implemented and tested some of the described hierarchical exploration architectures in different environments (See Table 1).

Problem	**Algorithm**	**Specificities**	**Implem. ?**
Exploring autonomously a fixed hierarchy	Exploring models in a hierarchy	Fixed learning cv	Yes
		Dynamic learning cv	Yes
	Top-Down Drive	Models used by max 1 model	Yes
		Other hierarchies	No
Exploring tasks of increasing complexity	Creating modules	Given M, S, O	Some
Learning with Social Guidance	Integration of Social Guidance	Task-space level	No
		Model level	
		Hierarchy level	

Table 1: Summary of developed algorithms

8 Roadmap

The next important steps are to properly formulate, analyze and illustrate the different questions of this report, first in abstract settings before real experimental setups. The first question is how to explore a given, fixed hierarchy of tasks, and how the different algorithms behave. In those algorithms, the hierarchy of models to learn is given to the agent as if it already knew the combinations of actions to explore in order to learn the successive tasks. This is a strong hypothesis as infants have first to discover what types of actions to combine to get a specific outcome. The second question is thus how and when to autonomously combine the previously learned actions in a relevant hierarchy of skills. Then, we will study the integration of social guidance into those algorithms.

In future work, we also plan to make an experiment in which a robot has to autonomously discover that sounds can influence a social peer (for example making him manipulate objects in the environment), using the extensions of our hierarchical exploration architectures that integrate social interaction (Section 3.1.8). Experiments will be conducted both through simulation of robotic environments and real robot experiments, in particular using the open-source humanoid platform Poppy. Then, we will work to allow for hierarchical reinforcement learning techniques in the model [Botvinick, 2012], in particular option theory [Sutton et al., 1999]). Also, we will study how techniques of multi-modal deep learning [Ngiam et al., 2011] can be used to select useful manifold in high-dimensional sensorimotor flows (i.e. find useful lower dimensional abstractions of the flows) on which skill acquisition techniques can be applied.

References

[Baranes and Oudeyer, 2010] Baranes, A. and Oudeyer, P.-Y. (2010). Intrinsically motivated goal exploration for active motor learning in robots: A case study. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1766–1773. IEEE.

[Baranes and Oudeyer, 2013] Baranes, A. and Oudeyer, P.-Y. (2013). Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73.

[Begus et al., 2014] Begus, K., Gliga, T., and Southgate, V. (2014). Infants learn what they want to learn: responding to infant pointing leads to superior learning. *PloS one*, 9(10).

[Botvinick, 2012] Botvinick, M. M. (2012). Hierarchical reinforcement learning and decision making. *Current opinion in neurobiology*, 22(6):956–962.

[Cangelosi et al., 2010] Cangelosi, A., Metta, G., Sagerer, G., Nolfi, S., Nehaniv, C., Fischer, K., Tani, J., Belpaeme, T., Sandini, G., Nori, F., et al. (2010). Integration of action and language knowledge: A roadmap for developmental robotics. *Autonomous Mental Development, IEEE Transactions on*, 2(3):167–195.

[Clement et al., 2014] Clement, B., Oudeyer, P.-Y., Roy, D., and Lopes, M. (2014). Online optimization of teaching sequences with multi-armed bandits. In *Educational Data Mining 2014*.

[Cleveland and Devlin, 1988] Cleveland, W. S. and Devlin, S. J. (1988). Locally weighted regression: an approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83(403):596–610.

[Csikszentmihalyi, 1990] Csikszentmihalyi, M. (1990). *Flow: The Psychology of Optimal Experience*. Perennial Modern Classics. Harper & Row.

[Fabisch and Metzen, 2014] Fabisch, A. and Metzen, J. H. (2014). Active contextual policy search. *The Journal of Machine Learning Research*, 15(1):3371–3399.

[Fenson et al., 1994] Fenson, L., Dale, P. S., Reznick, J. S., Bates, E., Thal, D. J., Pethick, S. J., Tomasello, M., Mervis, C. B., and Stiles, J. (1994). Variability in early communicative development. *Monographs of the society for research in child development.*

[Forestier, 2014] Forestier, S. (2014). Workshop notes: First interdisciplinary symposium on information-seeking, curiosity and attention. inria bordeaux sud-ouest, november 6-7, 2014.

[Franklin et al., 2014] Franklin, B., Warlaumont, A. S., Messinger, D., Bene, E., Nathani Iyer, S., Lee, C.-C., Lambert, B., and Oller, D. K. (2014). Effects of parental interaction on infant vocalization rate, variability and vocal type. *Language Learning and Development*, 10(3):279–296.

[Goldstein et al., 2010] Goldstein, M. H., Schwade, J., Briesch, J., and Syal, S. (2010). Learning while babbling: Prelinguistic object-directed vocalizations indicate a readiness to learn. *Infancy*, 15(4):362–391.

[Greenfield, 1991] Greenfield, P. M. (1991). Language, tools and brain: The ontogeny and phylogeny of hierarchically organized sequential behavior. *Behavioral and Brain Sciences*, 14:531–551.

[Guenther et al., 2006] Guenther, F. H., Ghosh, S. S., and Tourville, J. A. (2006). Neural modeling and imaging of the cortical interactions underlying syllable production. *Brain and language*, 96(3):280–301.

[Hansen, 2006] Hansen, N. (2006). The CMA evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pages 75–102. Springer.

[Higuchi et al., 2009] Higuchi, S., Chaminade, T., Imamizu, H., and Kawato, M. (2009). Shared neural correlates for language and tool use in broca's area. *Neuroreport*, 20(15):1376–1381.

[Howard and Messum, 2011] Howard, I. S. and Messum, P. (2011). Modeling the development of pronunciation in infant speech acquisition. *Motor Control*, 15(1):85–117.

[Ijspeert et al., 2013] Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., and Schaal, S. (2013). Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373.

[Kidd et al., 2012] Kidd, C., Piantadosi, S. T., and Aslin, R. N. (2012). The goldilocks effect: Human infants allocate attention to visual sequences that are neither too simple nor too complex. *PLoS One*, 7(5):e36399.

[Kocsis and Szepesvári, 2006] Kocsis, L. and Szepesvári, C. (2006). Discounted ucb. 2nd PASCAL Challenges Workshop.

[Kuhl, 1991] Kuhl, P. K. (1991). Human adults and human infants show a "perceptual magnet effect" for the prototypes of speech categories, monkeys do not. *Perception & psychophysics*, 50(2):93–107.

[Lapeyre et al., 2014] Lapeyre, M., Rouanet, P., Grizou, J., Nguyen, S., Depraetre, F., Le Falher, A., and Oudeyer, P.-Y. (2014). Poppy Project: Open-Source Fabrication of 3D Printed Humanoid Robot for Science, Education and Art. In *Digital Intelligence 2014*, page 6, Nantes, France.

[Mangin and Oudeyer, 2012] Mangin, O. and Oudeyer, P.-Y. (2012). Learning to recognize parallel combinations of human motion primitives with linguistic descriptions using non-negative matrix factorization. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 3268–3275. IEEE.

[Meltzoff and Warhol, 1999] Meltzoff, A. N. and Warhol, J. (1999). Born to learn: What infants learn from watching us. *The role of early experience in infant development*, pages 145–164.

[Metzen and Kirchner, 2013] Metzen, J. H. and Kirchner, F. (2013). Incremental learning of skill collections based on intrinsic motivation. *Frontiers in Neurorobotics*, 7.

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

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

[Mugan and Kuipers, 2009] Mugan, J. and Kuipers, B. (2009). Autonomously learning an action hierarchy using a learned qualitative state representation.

[Ngiam et al., 2011] Ngiam, J., Khosla, A., Kim, M., Nam, J., Lee, H., and Ng, A. Y. (2011). Multimodal deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 689–696.

[Nguyen and Oudeyer, 2012] Nguyen, S. and Oudeyer, P.-Y. (2012). Active choice of teachers, learning strategies and goals for a socially guided intrinsic motivation learner. *Paladyn*, 3(3):136–146.

[Oudeyer et al., 2007] Oudeyer, P.-Y., Kaplan, F., and Hafner, V. V. (2007). Intrinsic Motivation Systems for Autonomous Mental Development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286.

[Oudeyer and Smith, 2014] Oudeyer, P.-Y. and Smith, L. (2014). How evolution may work through curiosity-driven developmental process.

[Paatero and Tapper, 1994] Paatero, P. and Tapper, U. (1994). Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126.

[Paraschos et al., 2013] Paraschos, A., Daniel, C., Peters, J., and Neumann, G. (2013). Probabilistic movement primitives. In *Advances in Neural Information Processing Systems*, pages 2616–2624.

[Sutton et al., 1999] Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211.

[Vigorito and Barto, 2010] Vigorito, C. M. and Barto, A. G. (2010). Intrinsically motivated hierarchical skill learning in structured environments. *Autonomous Mental Development, IEEE Transactions on*, 2(2):132–143.

[Warlaumont et al., 2013] Warlaumont, A. S., Westermann, G., Buder, E. H., and Oller, D. K. (2013). Prespeech motor learning in a neural network using reinforcement. *Neural Networks*, 38:64–75.