Intrinsically Motivated Goal Exploration Processes with Automatic Curriculum Learning

Sébastien Forestier, Rémy Portelas, Yoan Mollard, Pierre-Yves Oudeyer

Intrinsically Motivated Goal Exploration Processes with Automatic Curriculum Learning

Sébastien Forestier seb@forest.bio
Inria Bordeaux Sud-Ouest
200 avenue de la Vieille Tour, 33405 Talence, France

Rémy Portelas remy.portelas@inria.fr
Inria Bordeaux Sud-Ouest
200 avenue de la Vieille Tour, 33405 Talence, France

Yoan Mollard yoan.mollard@inria.fr
Inria Bordeaux Sud-Ouest
200 avenue de la Vieille Tour, 33405 Talence, France

Pierre-Yves Oudeyer pierre-yves.oudeyer@inria.fr
Inria Bordeaux Sud-Ouest
200 avenue de la Vieille Tour, 33405 Talence, France

Editor:

Abstract

Intrinsically motivated spontaneous exploration is a key enabler of autonomous lifelong learning in human children. It enables the discovery and acquisition of large repertoires of skills through self-generation, self-selection, self-ordering and self-experimentation of learning goals. We present an algorithmic approach called Intrinsically Motivated Goal Exploration Processes (IMGEP) to enable similar properties of autonomous learning in machines. The IMGEP architecture relies on several principles: 1) self-generation of goals, generalized as parameterized fitness functions; 2) selection of goals based on intrinsic rewards; 3) exploration with incremental goal-parameterized policy search and exploitation with a batch learning algorithm; 4) systematic reuse of information acquired when targeting a goal for improving towards other goals. We present a particularly efficient form of IMGEP, that uses a population-based policy and an object-centered modularity. We provide several implementations of this architecture and demonstrate their ability to automatically generate a learning curriculum within several experimental setups including a real humanoid robot exploring multiple spaces of goals with several hundred continuous dimensions. While no particular target goal is provided to the system, this curriculum allows the discovery of skills that act as stepping stones for learning more complex skills, e.g. nested tool use.

Keywords: intrinsic motivations, goal exploration, curiosity-driven learning, automatic generation of goals, curriculum learning

1. INTRODUCTION

An extraordinary property of natural intelligence in humans is their capacity for lifelong autonomous learning. Processes of autonomous learning in infants have several properties that are fundamentally different from many current machine learning systems. Among them is the capability to spontaneously explore their environments, driven by an intrinsic motivation to discover and learn new tasks and problems that they imagine and select by themselves (Berlyne, 1966; Gopnik et al., 1999). Crucially, there is no engineer externally imposing one target goal that they should explore, hand providing a curriculum for learning, nor providing a ready-to-use database of training examples. Rather, children self-select their objectives within a large, potentially open-ended, space of goals they can imagine, and

they collect training data by physically practicing these goals. In particular, they explore goals in an organized manner, attributing to them values of interestingness that evolve with time, and allowing them to self-define a learning curriculum that is called a developmental trajectory in developmental sciences (Thelen and Smith, 1996). This self-generated learning curriculum prevents infants from spending too much time on goals that are either too easy or too difficult, and allows them to focus on goals of the right level of complexity at the right time. Within this process, the new learned solutions are often stepping stones for discovering how to solve other goals of increasing complexity. Thus, while they are not explicitly guided by a final target goal, these mechanisms allow infants to discover highly complex skills. For instance, biped locomotion or tool use would be extremely difficult to learn by focusing only on these targets from the start as the rewards for those goals are typically rare or deceptive.

An essential component of such organized spontaneous exploration is the intrinsic motivation system, also called curiosity-driven exploration system (Kidd and Hayden, 2015; Oudeyer and Smith, 2016; Gottlieb and Oudeyer, 2018). In the last two decades, a series of computational and robotic models of intrinsically motivated exploration and learning in infants have been developed (Oudeyer and Kaplan, 2007; Baldassarre and Mirolli, 2013; Bazhydai et al., 2020), opening new theoretical perspectives in neuroscience and psychology (Gottlieb et al., 2013). Two key ideas have allowed to simulate and predict important properties of infant spontaneous exploration, ranging from vocal development (Moulin-Frier et al., 2013; Forestier and Oudeyer, 2017), to object affordance and tool learning (Forestier and Oudeyer, 2016a,b). The first key idea is that infants might select experiments that maximize an intrinsic reward based on empirical learning progress (Oudeyer et al., 2007). This mechanism would generate automatically developmental trajectories (e.g. learning curricula) where progressively more complex tasks are practiced, learned and used as stepping stones for more complex skills. The second key idea is that beyond selecting actions or states based on the predictive learning progress they provide, a more powerful way to organize intrinsically motivated exploration is to select goals, i.e. self-generated fitness functions, based on a measure of competence progress, i.e. a measure of progress in learning to produce diverse and controlled behavioral features (Baranes and Oudeyer, 2010b, 2013). Here, the intrinsic reward is the empirical improvement towards solving self-selected goals (Oudeyer and Kaplan, 2007; Forestier and Oudeyer, 2016a), happening through lower-level policy search mechanisms that generate physical actions. The efficiency of such goal exploration processes relies on a form of hindsight learning that leverages the fact that the data collected when targeting a goal can be informative to find better solutions to other goals (for example, a learner trying to achieve the goal of pushing an object on the right but actually pushing it on the left fails to progress on this goal, but learns as a side effect how to push it on the left). These general ideas have been instantiated and studied in the context of population-based learning architectures (e.g. Baranes and Oudeyer (2013); Péré et al. (2018)), as well as more recently in goal-conditioned reinforcement learning architectures (e.g. Colas et al. (2018a); Nair et al. (2018)).

Beyond neuroscience and psychology, we believe these models open new perspectives in artificial intelligence. In particular, algorithmic architectures for intrinsically motivated goal exploration were shown to allow the efficient acquisition of repertoires of high-dimensional motor skills with automated curriculum learning in several robotics experiments (Baranes and Oudeyer, 2013; Forestier and Oudeyer, 2016a). This includes for example learning omnidirectional locomotion or learning multiple ways to manipulate complex soft objects (Rolf et al., 2010; Baranes and Oudeyer, 2013).

In this article, we make several contributions:

- We present a formalization of Intrinsically Motivated Goal Exploration Processes (IMGEP), that is both more compact and more general than these previous models. In particular, it considers a generalized definition of the concept of goals, construed as abstract parameterized fitness functions that can express arbitrary objectives over full behavioural trajectories and include constraints. This enables to express a diversity of goal exploration algorithms in the same framework, including Quality-Diversity algorithms that were not previously formalized as goal exploration algorithms.

- We present a new population-based IMGEP algorithmic architecture, called MPOP-IMGEP, implementing two forms of object-centered modularity. First, we introduce spatial modularity: each object of the environment is associated to a goal space. Second, we introduce temporal modularity: the temporal structure of objects' movement is leveraged for more efficient leveraging of discovered stepping-stones in goal exploration, through a stepping-stone preserving mutation operator (SPM). We also present various instantiation
- We present a systematic experimental study of this new IMGEP algorithm in diverse environments providing opportunities for discovering complex skills like tool use, as well as including complex distractors: a 2D simulated environment, a Minecraft environment, and a real humanoid robotic setup. We compare several variants of IMGEP algorithms, including ablations, in terms of sample efficiency to discover a diversity of behavioral features. We also compare IMGEPs algorithms with algorithms exploring only one target object: we show that letting agents self-organize exploration of diverse objects is vastly more efficient for discovering how to control the target object than channeling the agent to explore only this object. We also compare the exploration resulting from the self-organized learning curriculum of intrinsically motivated agents with the exploration following a curriculum designed by hand with expert knowledge of the task, showing a similar exploration efficiency.

2. INTRINSICALLY MOTIVATED GOAL EXPLORATION PROCESSES

We define a framework for the intrinsically motivated exploration of multiple goals, where the data collected when exploring a goal give some information to help reach other goals. This framework considers that when the agent performed an experiment, it can compute the fitness of that experiment for achieving any goal, not only the one it was trying to reach. Importantly, it does not assume that all goals are achievable, nor that they are of a particular form, enabling to express complex objectives that do not simply depend on the observation of the end policy state but might depend on several aspects of entire behavioral trajectories (see Box on features, goals and goal spaces). Also, the agent autonomously builds its goals but does not know initially which goals are achievable or not, which are easy and which are difficult, nor if certain goals need to be explored so that other goals become achievable.

2.1 NOTATIONS AND ASSUMPTIONS

Let's consider an agent that executes continuous **actions** $a \in \mathcal{A}$ in continuous **states** $s \in \mathcal{S}$ of an **environment** E. We consider policies producing time-bounded rollouts through the dynamics $\delta_E(\boldsymbol{s_{t+1}} \mid \boldsymbol{s_{t_0:t}}, \boldsymbol{a_{t_0:t}})$ of the environment, and we denote the corresponding behavioral trajectories $\tau = \{s_{t_0}, a_{t_0}, \cdots, s_{t_{end}}, a_{t_{end}}\} \in \mathbb{T}$.

We assume that the agent is able to construct a goal space $\mathcal{G}$ parameterized by g, representing fitness functions f_g giving the fitness $f_g(\tau)$ of an experimentation τ to reach a goal g (see Box on features, goals and goal spaces). Also, we assume that given a trajectory τ, the agent can compute $f_g(\tau)$ for any $g \in \mathcal{G}$.

Given these spaces $\mathcal{S}$, $\mathcal{A}$, $\mathcal{G}$, the agent explores the environment by sampling goals in $\mathcal{G}$ and searching for good solutions to those goals, and learns a **goal-parameterized policy** $\Pi(\boldsymbol{a_{t+1}} \mid \boldsymbol{g}, \boldsymbol{s_{t_0:t+1}}, \boldsymbol{a_{t_0:t}})$ to reach any goal from any state.

We can then evaluate the agent's exploration and learning efficiency either by observing its behavior and estimating the diversity of its skills and the reached stepping-stones, or by "opening" agent's internal models and policies to analyze their properties.

Generalized Goals and Goal Spaces

In the general case, the agent has algorithmic tools to construct any goal as any function f_g (parameterized by vector g), taking as input a state-action trajectory τ, and returning the fitness of τ for achieving the goal. As a shorthand, we call the goal g (vector of parameters), but the full representation of the goal consists in the combination of the parameters g *and* the function or program computing the fitness function[a].

Formally, given a behavioural trajectory $\tau = \{s_{t_0}, a_{t_0}, \cdots, s_{t_{end}}, a_{t_{end}}\}$, we assume the agent can compute a set of **behavioural features** $\varphi_1(\tau), ..., \varphi_n(\tau)$, also called **outcomes**, over the full trajectory. Those features are vectors that encode any static or dynamic property of the environment or the agent itself. Examples include the average speed of an object, an encoding of an object's trajectory or relation to other objects, the frequency of a movement, or the result of a test checking whether a movement or an object fulfills some properties specified in a sub-part of vector g. The features may be given to the agent, or learned, for instance using generative models (see Péré et al. (2018); Nair et al. (2018); Laversanne-Finot et al. (2018); Reinke et al. (2019)).

We also assume that computational tools, in the form of mathematical operators[b], are available to the agent for constructing goals, i.e. for constructing fitness functions f_g using these features. Examples include:

- $f_g(\tau) = \varphi_g(\tau)$: the goal is to produce a trajectory τ that maximizes feature $\varphi_g(\tau)$, where the goal parameter vector g is a simple one dimensional index of the target behavioural feature. Sampling the space of goals thus amounts to sampling which feature to maximize, e.g. maximize agent's speed or number of objects collected.

- $f_g(\tau) = -||\varphi_{\psi_1(g)}(\tau) - \psi_2(g)||$: the goal g is to produce a trajectory τ so that its features $\varphi_{\psi_1(g)}(\tau)$ are as close as possible to the target vector $\psi_2(g)$, using a measure $||.||$, e.g. move the hand or a ball to a particular 3D position (Baranes and Oudeyer, 2013; Péré et al., 2018), or produce a sound with a particular target spectrum (Moulin-Frier et al., 2014).

- $f_g(\tau) = \varphi_i(\tau)$ `if` $\varphi_j(\tau) \in \psi_3(g)$ `else` 0: the goal g is to produce a trajectory τ which maximizes feature i while keeping feature vector j inside a local region specified by $\psi_3(g)$, e.g. maximize agent's speed while displaying a certain type of pattern of leg movement. When goal sampling is made in this form of function space, IMGEPs correspond to quality-diversity algorithms (Pugh et al., 2016) such as MAP-Elite (Cully et al., 2015).

- $f_g(\tau) = f_{g_1}(\tau)$ `if` $f_{g_2}(\tau) < f_{g_3}(\tau)$ `else` $f_{g_4}(\tau)$: goals can be combined to form more complex constrained optimization problems, e.g. move the ball to follow a target while not getting too close to the walls and holes and minimizing the energy spent.

A **goal space** is a set of goals (fitness functions) parameterized by a vector g. Diverse forms of structured parameterization can be used, as shown by the examples above, and corresponding to diverse types of goals. In the experiments of this paper, we define several goal spaces, each with a parameterization representing the target trajectory of positions or sound or light of an object in the environment. Each object k and modality m defines a goal space $\mathcal{G}^{k,m}$ containing goals g of the form $f_g^{k,m}(\tau) = -||\varphi_{\psi_1^{k,m}(g)}(\tau) - \psi_2^{k,m}(g)||$ where $\psi_2^{k,m}(g)$ denotes the target trajectory of object k in modality m (positions, sound or light), and $\varphi_{\psi_1^{k,m}(g)}(\tau)$ denotes the features of trajectory τ of the same object and modality.

a. The term "fitness function" is here a synonym for "goal-achievement reward function" sometimes used in related work.

b. These operators are pre-defined in existing work and in this paper, but they could in principle be learned.

Architecture 1 Intrinsically Motivated Goal Exploration Process (IMGEP)

Require: Action space $\mathcal{A}$, State space $\mathcal{S}$
Initialize knowledge $\mathcal{E} = \varnothing$
Initialize goal space $\mathcal{G}$ and goal policy Γ
Initialize policies Π and Π_ϵ
Launch asynchronously the two following loops:
loop ▷ Exploration loop
  Choose goal g in $\mathcal{G}$ with Γ
  Execute a roll-out of Π_ϵ, observe trajectory τ
  ▷ From now on $f_{g'}(\tau)$ can be computed to estimate the fitness of the current experiment τ for achieving any goal $g' \in \mathcal{G}$
  Compute the fitness $f = f_g(\tau)$ associated to goal g
  Compute intrinsic reward $r_i = IR(\mathcal{E}, g, f)$ associated to g
  Update exploration policy Π_ϵ with $(\mathcal{E}, g, \tau, f)$ ▷ e.g. fast incremental algo.
  Update goal policy Γ with $(\mathcal{E}, g, \tau, f, r_i)$
  Update knowledge $\mathcal{E}$ with (g, τ, f, r_i)
loop ▷ Exploitation loop
  Update policy Π with $\mathcal{E}$ ▷ e.g. batch training of deep NN, SVMs, GMMs
  Update goal space $\mathcal{G}$ with $\mathcal{E}$
return Π

2.2 ALGORITHMIC ARCHITECTURE

We present Intrinsically Motivated Goal Exploration Processes (IMGEP) as an algorithmic architecture that can be instantiated into many particular algorithms sharing several general principles (see pseudo-code in Architecture 1):

- The agent autonomously builds and samples goals as fitness functions, possibly using intrinsic rewards,
- Two processes are running in parallel: 1) an exploration loop samples goals and searches for good solutions to those goals with the exploration policy; 2) an exploitation loop uses the data collected during exploration to improve the goal-parameterized policy and the goal space,
- The data acquired when exploring solutions for a particular goal is reused to extract potential solutions to other goals.

2.3 GOAL EXPLORATION

In the exploration loop, the agent samples a goal g, executes its exploration policy Π_ϵ, and observes the resulting trajectory τ. This new experiment τ can be used to:

- compute the fitness associated to goal g,
- compute an intrinsic reward evaluating the interest of the choice of g,
- update the goal policy (sampling strategy) using this intrinsic reward,
- update the exploration policy Π_ϵ with a fast incremental learning algorithm,
- update the learning database $\mathcal{E}$.

Then, asynchronously, this learning database $\mathcal{E}$ can be used to learn a target policy Π with a slower or more computationally demanding algorithm, but on the other end resulting in a more accurate policy. The goal space may also be updated based on this data.

2.4 INTRINSIC REWARDS

In goal exploration, a goal $g \in \mathcal{G}$ is chosen at each iteration. $\mathcal{G}$ may be infinite, continuous and of high-dimensionality, making the choice of goal important and non-obvious. Indeed, even if the fitness function $f_{g'}(\tau)$ may give information about the fitness of a trajectory τ to achieve many goals $g' \in \mathcal{G}$, the policy leading to τ has been chosen with the goal g to solve in mind, thus it may not give as much information about other goals than the execution of another policy chosen when targeting other goals.

Intrinsic rewards provide a mean for the agent to self-estimate the expected interest of exploring particular goals for learning how to achieve all goals in $\mathcal{G}$. An intrinsic reward signal r_i is associated to a chosen goal g, and based on a heuristic (denoted IR) such as outcome novelty, progress in reducing outcome prediction error, or progress in competence to solve problems (Oudeyer and Kaplan, 2007).

In the experiments of this paper, we use intrinsic rewards based on measuring the competence progress towards the self-generated goals, which has been shown to be particularly efficient for learning repertoires of high-dimensional robotics skills (Baranes and Oudeyer, 2013). Figure 1 shows a schematic representation of possible learning curves and the exploration preference of an agent with intrinsic rewards based on learning progress.

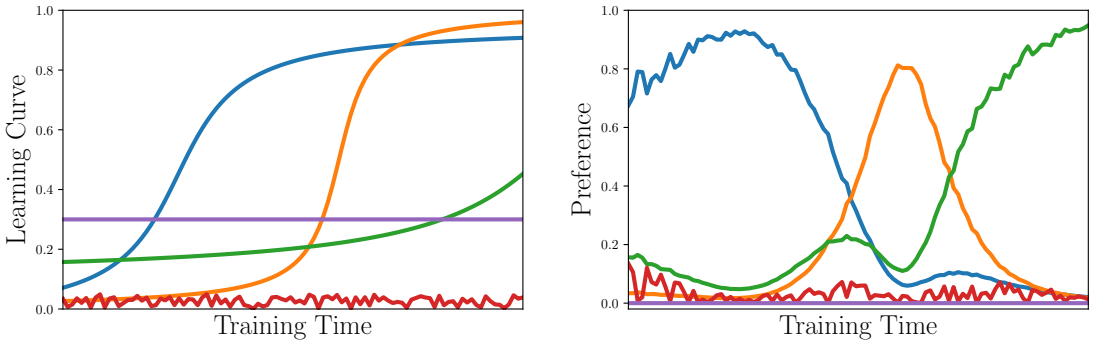


Figure 1: Schematic representation of possible learning curves for different goals and the associated exploration preference for an agent with intrinsic rewards based on learning progress. Left: schematic learning curves associated to 5 imaginary goals: the y axis represents the competence of the agent to achieve the goal (1 is perfect, 0 is chance level), and the x axis is training time on a goal. The blue, orange and green curves represent achievable goals, for which agent's competence increases with training, at different rates, and saturates after a long training time. The purple curve represents a goal on which the agent always has the same competence, with no progress. The red curve is the learning curve on an unreachable goal, e.g. moving an uncontrollable object. Right: exploration preference of an agent with a learning progress heuristic (competence derivative) to explore the 5 goals defined by the learning curves. At the beginning of exploration, the agent makes the most progress on goal blue so it prefers to train on this one, and then its preference will shift towards goals orange and green. The agent is making no progress on goal purple so will not choose to explore it, and goal red has a noisy but low estimated learning progress.

3. MODULAR POPULATION-BASED IMGEP ARCHITECTURE

In the previous section, we defined the most general IMGEP architecture without specifying the implementation of its components such as goals and policies. We define here a particular IMGEP architecture, used in the experiments of this paper, where the goal-parameterized policy Π is based on

a population of solutions, the goal space $\mathcal{G}$ is constructed in a modular manner from a set of objects and the exploration mutations is temporally modular through taking into account the movement of those objects. This particular architecture is called Modular Population-Based IMGEP, and we detail its ingredients in the following sections. Figure 2 summarizes the different components of the architecture, and the pseudo-code is provided in Architecture 2.

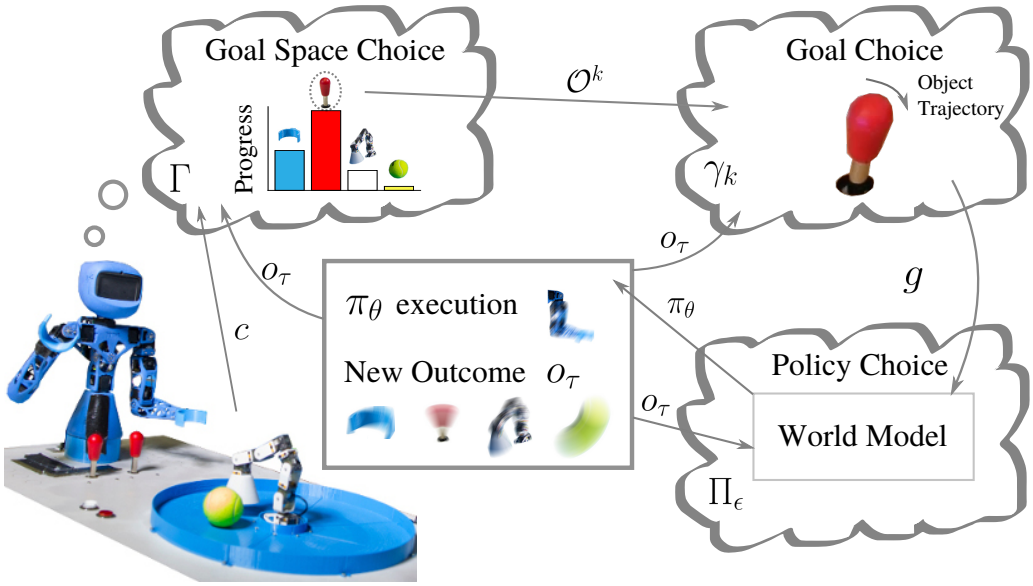


Figure 2: Summary of our modular population-based IMGEP implementation. At each iteration, the agent observes the current context c and chooses a goal space (an object) to explore based on intrinsic rewards (the learning progress to move each object) with Γ. Then a particular goal g for the chosen object is sampled with γ_k, for instance to push the left joystick to the right. The agent chooses the best policy parameters θ to reach this goal, with the exploration meta-policy Π_ϵ, and using an internal model of the world implementing object-centered modularity in goals and mutations. The agent executes policy π_θ, observes the trajectory τ and compute the outcome o_τ encoding the movement of each object. Finally, each component is updated with the result of this experiment.

3.1 POPULATION-BASED META-POLICIES Π AND Π_ϵ

In this version of the IMGEP framework, the goal-parameterized policy Π is based on a population of low-level policies. We consider that the starting state s_{t_0} of a trajectory is characterized by a feature vector c called **context**. The policy Π is built from a set of low-level policies π_θ parameterized by $\theta \in \Theta$, and from a meta-policy $\Pi(\boldsymbol{\theta} \mid \boldsymbol{g}, \boldsymbol{c})$ which, given a goal and context, chooses the best policy π_θ to achieve the goal g. The policies $\pi_\theta(\boldsymbol{a_{t+1}} \mid \boldsymbol{s_{t_0:t+1}}, \boldsymbol{a_{t_0:t}})$ can be implemented for instance by stochastic black-box generators or small neural networks (see experimental section).

During the goal exploration loop, the main objective consists in collecting data that covers well the space of goals: finding θ parameters that yield good solutions to as many goals as possible. The **exploration meta-policy** $\Pi_\epsilon(\boldsymbol{\theta} \mid \boldsymbol{g}, \boldsymbol{c})$ is learned and used to output a distribution of policies π_θ that are interesting to execute to gather information for solving in context c the self-generated goal g and goals similar to g. To achieve the objective of collecting interesting data, the exploration meta-policy Π_ϵ must have fast and incremental updates. As the aim is to maximize the coverage

Architecture 2 Modular Population-Based IMGEP

Require: Action space $\mathcal{A}$, State space $\mathcal{S}$, Context space $\mathcal{C}$, Outcome space $\mathcal{O}$

Initialize knowledge $\mathcal{E} = \varnothing$
Initialize goal space $\mathcal{G}$, goal policies γ_k and goal space policy Γ
Initialize meta-policies Π and Π_ϵ
Launch asynchronously the two following loops:
loop ▷ Exploration loop
    Observe context c
    Choose goal space $\mathcal{G}^k$ with Γ
    Choose goal g in $\mathcal{G}^k$ with γ_k
    Choose policy parameters θ to explore g in context c with Π_ϵ
    Execute a roll-out of π_θ, observe trajectory τ
    Compute outcome o_τ from trajectory τ
    ▷ From now on, $f_{g'}(\tau)$ can be computed to estimate the fitness of the experiment τ for achieving any goal $g' \in \mathcal{G}$
    Compute the fitness $f = f_g(\tau)$ associated to goal g
    Compute intrinsic reward $r_i = IR(\mathcal{E}, c, g, \theta, o_\tau, f)$ associated to g in context c
    Update exploration meta-policy Π_ϵ with $(\mathcal{E}, c, \theta, \tau, o_\tau)$ ▷ e.g. fast incr. algo.
    Update goal policy γ_k with $(\mathcal{E}, c, g, o_\tau, f, r_i)$
    Update goal space policy Γ with $(\mathcal{E}, c, k, g, o_\tau, f, r_i)$
    Update knowledge $\mathcal{E}$ with $(c, g, \theta, \tau, o_\tau, f, r_i)$
loop ▷ Exploitation loop
    Update meta-policy Π with $\mathcal{E}$ ▷ e.g. batch training of DNN, SVM, GMM
return Π

of the space of goals, being very precise when targeting goals is less crucial than the capacity to update the meta-policy quickly and incrementally. In our experiments, the exploration meta-policy $\Pi_\epsilon(\boldsymbol{\theta} \mid \mathbf{g}, \mathbf{c})$ is implemented as a fast memory-based nearest neighbor search with a kd-tree.

On the contrary, the purpose of the **target meta-policy** Π is to be used in exploitation mode: later on, it can be asked to solve as precisely as possible some goals g with maximum fitness. As the training of this meta-policy can be done asynchronously from data collected by the goal exploration loop, this allows the use of slower training algorithms, possibly batch, that might generalize better, e.g. using Gaussian mixture models, support vector regression or (deep) neural networks. These differences justify the fact that IMGEP uses in general two different representations and learning algorithms for Π_ϵ and Π. This two-level learning scheme has similarities with the Complementary Learning Systems Theory used to account for the organization of learning in mammalian brains (Kumaran et al., 2016).

3.2 OBJECT-CENTERED MODULAR GOAL CONSTRUCTION

In the IMGEP architecture, the agent builds and samples goals autonomously. Here, we consider the particular case where the agent builds several goal spaces that correspond to moving each object in the environment. Each goal space represents features of the movement of an object in the environment, such as its end position in τ or its full trajectory.

We define the **outcome** $o_\tau \in \mathcal{O}$ of an experiment τ as the features of the movement of all objects, so that $\mathcal{O} = \prod_k \mathcal{O}^k$ where $o_\tau^k \in \mathcal{O}^k$ are the features of object k. Those features come from a perceptual system that may be given to the agent or learned by the agent. From feature space $\mathcal{O}^k$, the agent can autonomously generate a corresponding goal space $\mathcal{G}^k$ that contains fitness functions of the form $f_g(\tau) = -||g - o_\tau^k||_k$. The norm $||.||_k$ is a distance in the space O^k, which can be normalized to be

able to compare the fitness of goals across goal spaces. The goal space is thus modular, composed of several object-related subspaces: $\mathcal{G} = \bigcup_k \mathcal{G}^k$.

With this setting, goal sampling is hierarchical in the sense that the agent first chooses a goal space $\mathcal{G}^k$ to explore with a goal space policy Γ and then a particular goal $g \in \mathcal{G}^k$ with the corresponding goal policy γ_k. Those two levels of choice can make use of self-computed intrinsic rewards r_i (see Sec. 2.4).

Given an outcome o_τ, the fitness $f_g(\tau)$ can thus be computed by the agent for all goals $g \in \mathcal{G}$ and at any time after the experiment τ. For instance, if while exploring the goal of moving object A to the left, object B moved to the right, that outcome can be taken into account later when the goal is to move object B.

3.3 OBJECT-CENTERED TEMPORAL MODULARITY: STEPPING-STONE PRESERVING MUTATIONS

When targeting a new goal g, the internal model (a memory-based nearest neighbor search in our experiments) infers the best policy π_θ to reach the goal g. The exploration meta-policy Π_ϵ then performs a mutation on θ in order to explore new policies. The mutation operator could just add a random noise on the parameters θ, however, those parameters do not all have the same influence on the execution of the policy, in particular with respect to time. In our implementations, the parameters are sequenced in time, with some parameters influencing more the beginning of the policy roll-out and some more the end of the trajectory. However, in the context of tool use, the reaching or grasping of a tool is necessary for executing a subsequent action on an object. A random mutation of policy parameters, irrespective of the moment when the tool is grasped, can result in an action where the agent do not grasp the tool and thus cannot explore the corresponding object.

The Stepping-Stone Preserving Mutation operator (`SSPMutation`) analyzes the trajectory of the target object while the previous motor policy π_θ was run, to find the moment when the object started to move. The operator does not change the variables of θ concerning the movement before the object moved and mutates the variables of θ concerning the movement after the object moved (see an example mutation in Fig. 15). When the goal of the agent is to move the tool and it already succeeded to move the tool in the past with policy π_θ, then the application of this mutation operator changes the behavior of the agent only when the tool start to move, which makes the agent explore with the tool once grasped and avoid missing the tool. Similarly, when the goal of the agent is to move a toy controlled by a tool, the mutation changes the behavior only when the toy starts to move, which makes the agent grasp the tool and reach the toy before exploring new actions, so that the agent do not miss the tool nor the toy. The idea of this stepping-stone preserving operator is similar to the Go-Explore approach (Ecoffet et al., 2019).

3.4 VARIOUS IMPLEMENTATIONS OF MODULAR POPULATION-BASED IMGEP

The Modular Population-Based IMGEP architecture gives a high-level description of the learning agent with a population-based policy and an object-centered modularity in goals and mutations. Each component of this architecture may have several particular implementations, such that various implementations the learning agent can be derived. For instance, in the main loop of Architecture 2, many aspects are not constrained such as how the goal is chosen, how the parameters θ are computed, how the policies π_θ are implemented, how the intrinsic rewards are defined. In Section 4.2, we detail three implementations of the Modular Population-Based IMGEP architecture.

4. EXPERIMENTS

In this section, we evaluate the Modular Population-Based IMGEP architecture by designing several algorithmic implementations and several environments suitable for curriculum learning, where the exploration of a task brings information to later solve other tasks. In particular, we study environments where agents discover objects that can be used as tools to move other objects. A good exploration of a tool and of its functioning will yield a better exploration of the objects on which this tool can act. Those tasks provide the opportunity for an intrinsically motivated agent to build on the skills it has learned to explore and learn new skills on its own.

Here, we first describe three tool-use learning environments and we detail our implementations of IMGEP and of several control conditions. Then, we study the behavior of the different agents in the different environments depending on the learning architecture. We investigate in particular the benefits of a modular representation of the sensory feedback with goals based on objects, and how the exploration mutations can take into account the movement of the goal object. We further examine how and in which conditions the intrinsic motivation component of the IMGEP architecture improves the learning of skills that can be reused, such as using a tool to move an object.

4.1 TOOL-USE ENVIRONMENTS

We design three tool-use environments. The first one is a 2D simulated robotic arm with 3 joints and a gripper that can grab sticks and move toys. It is a simple environment with no physics and only 2D geometric shapes so very fast to execute. The second environment is a Minecraft scene where an agent is able to move, grab and use tools such as a pickaxe to break blocks. The third one is a real robotic setup with a Torso robot moving its arm that can reach joysticks controlling a toy robot. This setup has complex high-dimensional motor and sensory spaces with noise both in the robot physical arm and in the interaction between objects such as its hand and the joysticks. It is a high-dimensional and noisy environment with a similar stepping-stone structure as the robotic environments but with a completely different sensorimotor setup. The code of the different environments and experiments is available on GitHub[1].

4.1.1 2D SIMULATED TOOL-USE ENVIRONMENT

In the 2D Simulated Environment (see Fig. 3), the learning agent controls a robotic arm with a gripper, that can grab one of two sticks, one with a magnet at the end and one with Velcro, that can themselves be used to move several magnets or Velcro toys. Some other objects cannot be moved, they are called static distractors, and finally a simulated cat and dog are randomly moving in the scene, they are random distractors.

The 2D robotic arm has 3 joints that can rotate from $-\pi$ *rad* to π *rad*. The length of the 3 segments of the arm are 0.5, 0.3 and 0.2 so the length of the arm is 1 unit. The starting position of the arm is vertical with joints at position 0 *rad* and its base is fixed at position $(0, 0)$. The gripper gr has 2 possible positions: *open* ($gr \geqslant 0$) and *closed* ($gr < 0$). The robotic arm has 4 degrees of freedom represented by a vector in $[-1, 1]^4$.

Two sticks of length 0.5 can be grasped by the handle side (orange side) in order to catch an out-of-reach object. The magnetic stick can catch magnetic objects (in blue), and the other stick has a Velcro tape to catch Velcro objects (in green). If the gripper closes near the handle of one stick, this stick is considered grasped and follows the gripper's position and the orientation of the arm's last segment until the gripper opens. If the other side of a stick reaches a matching object (magnetic or Velcro), the object then follows the stick. There are three magnetic objects and three Velcro objects, but only one of each type is reachable with its stick. A simulated cat and dog are following a random walk, they have no interaction with the arm nor with other object. Finally, four

1. Code of the IMGEP experiments: https://github.com/sebastien-forestier/IMGEP

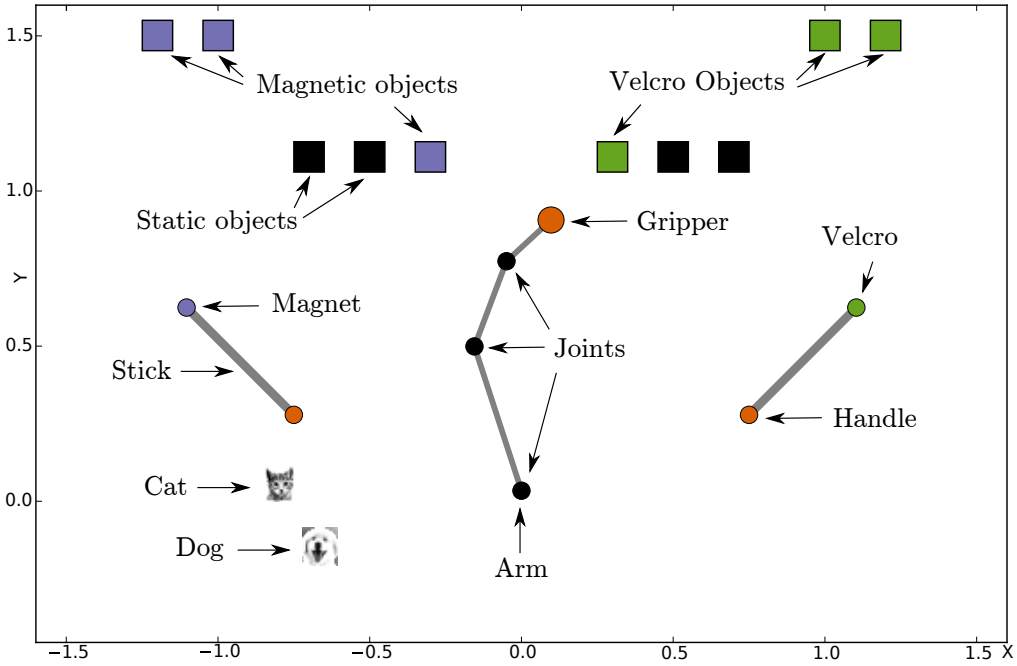


Figure 3: 2D Simulated Tool-Use Environment. A simulated robotic arm with a gripper can grab sticks and move toys. The gripper has to close near the handle of a stick to grab it. One magnetic toy and one Velcro toy are reachable with their corresponding stick. Other toys cannot be moved (static or too far away). The cat and the dog are distractors: they move randomly, independently of the arm.

static black squares have also no interaction with other objects. The arm, tools and other objects are reset to their initial state at the end of each iteration of 50 steps.

The agent receives a sensory feedback representing the result of its actions. This feedback (or outcome) is either composed of the position of each object at 5 time points during the 50 steps trajectory, or just the end state of each object, depending on the experiments. First, the hand is represented by its X and Y position and the aperture of the gripper (1 or -1). The sticks are represented by the X and Y position of their tip. Similarly, each other object is represented by their X and Y positions. Each of the 15 objects defines a sensory space S_i. The total sensory space S has either 155 dimensions if trajectories are represented, or 31 dimensions if only the end state of each object is represented.

4.1.2 MINECRAFT MOUNTAIN CART

The Minecraft Mountain Cart (MMC) extends the famous Mountain Car control benchmark in a 3D environment with a multi-goal setting (see Fig. 4).

In this episodic task, the agent starts on the left of the rectangular arena and is given ten seconds (40 steps) to act on the environment using 2 continuous commands: *move* and *strafe*, both using values in $[-1; 1]$. *move(1)* moves the agent forward at full speed, *move(-0.1)* moves the agent slowly backward, etc. Similarly *strafe(1)* moves the agent left at full speed and *strafe(-0.1)* moves it slowly to the right. Additionally, a third binary action allows the agent to use the currently handled tool.

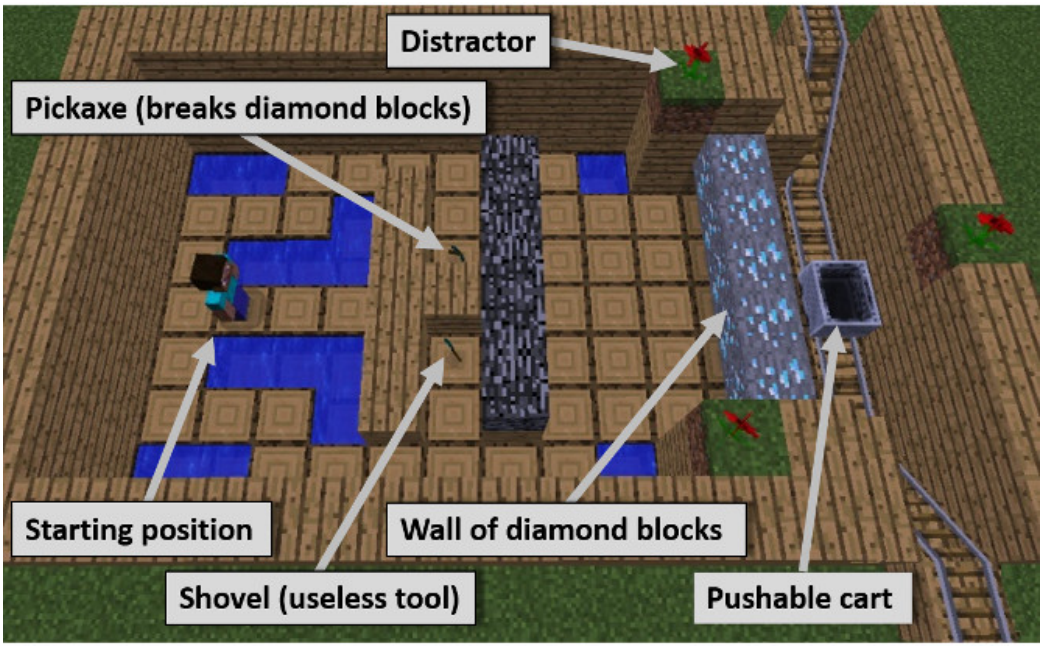


Figure 4: Minecraft Mountain Cart Environment. If the agent manages to avoid falling into water holes it may retrieve and use a pickaxe to break diamond blocks and access the cart. A shovel is also located in the arena and serves as a controllable distractor.

The first challenge of this environment is to learn how to navigate within the arena's boundaries without falling in water holes (from which the agent cannot get out). Proper navigation might lead the agent to discover one of the two tools of the environment: a shovel and a pickaxe. The former is of no use but the latter enables to break diamond blocks located further ahead in the arena. A last possible interaction is for the agent to get close enough to the cart to move it along its railroad. If given enough speed, the cart is able to climb the left or right slope. The height and width of these slopes were made in such a way that an agent simply hitting the cart at full speed will not provide enough inertia for the cart to climb the slope. The agent must at least partially support the cart along the track to propel it fast enough to fully climb the slope.

The outcome of an episode is a vector composed of the end position of the agent (2D), shovel (2D), pickaxe (2D), cart (1D) and 3 distractors (2D each) positions along with a binary vector (5D) encoding the 5 diamond blocks' states.

This environment is interesting to study modular IMGEP approaches since it is composed of a set of linked tasks of increasing complexity. Exploring how to navigate will help to discover the tools and, eventually, will allow to break blocks and move the cart.

4.1.3 ROBOTIC TOOL-USE ENVIRONMENT

In order to benchmark different learning algorithms in a realistic robotic environment with high-dimensional action and outcome spaces, we designed a real robotic setup composed of a humanoid arm in front of joysticks that can be used as tools to act on other objects (see Fig. 5). We recorded a video of an early version of the experimental setup[2].

2. Early version of the experimental setup: https://youtu.be/NOLAwD4ZTW0

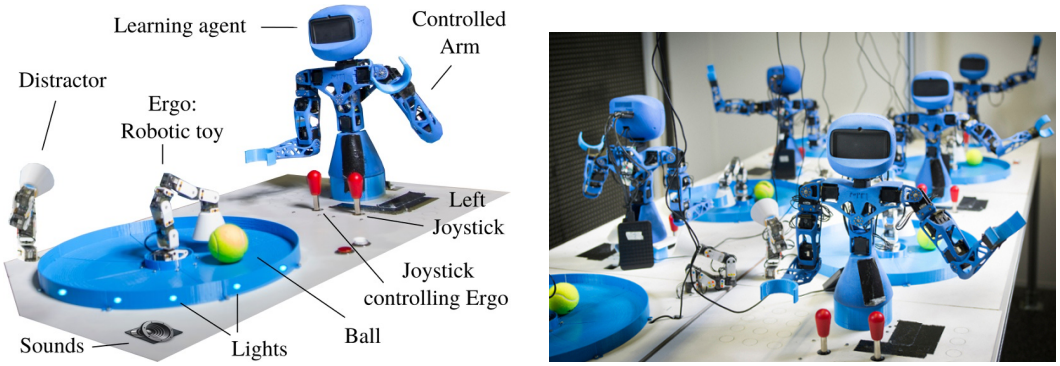


Figure 5: Robotic Tool-Use Environment. Left: a Poppy Torso robot (the learning agent) is mounted in front of two joysticks that can be used as tools to act on other objects: a Poppy Ergo robotic toy and a ball that can produce light and sound. Right: 6 copies of this setup are running in parallel to gather more data. Several Ergo robots are placed between robots: they act as distractors that move randomly, independently of the agents.

A Poppy Torso robot (the learning agent) is mounted in front of two joysticks and explores with its left arm. A Poppy Ergo robot (seen as a robotic toy) is controlled by the right joystick and can push a ball that controls some lights and sounds. Poppy is a robust and accessible open-source 3D printed robotic platform (Lapeyre et al., 2014).

The left arm has 4 joints, with a hook at the tip of the arm. A trajectory of the arm is here generated by radial basis functions with 5 parameters on each of the 4 degrees of freedom (20 parameters in total).

Two analogical joysticks (Ultrastick 360) can be reached by the left arm and moved in any direction. The right joystick controls the Poppy Ergo robotic toy, and the left joystick do not control any object. The Poppy Ergo robot has 6 motors, and moves with hardwired synergies that allow control of rotational speed and radial extension.

A tennis ball is freely moving in the blue arena which is slightly sloped so that the ball comes close to the center at the end of a movement. The speed of the ball controls (above a threshold) the intensity of the light of a LED circle around the arena. Finally, when the ball touches the border of the arena, a sound is produced and varied in pitch depending on ball position.

Several other objects are included in the environment, with which the agent cannot interact. Two Ergo robots (2D objects) are moving randomly, independently of the agent. Six objects are static: the right hand (3D) of the robot that is disabled in this experiment, the camera recording the ball trajectory (3D), the blue circular arena (2D), an out-of-reach yellow toy (2D), the red button also out-of-reach (2D) and the lamp (2D). All distractor objects are reset after each roll-out.

The context c of this environment represents the current configuration of objects in the scene. In practice, since only the Ergo and ball are not reset after each roll-out, this amounts to measuring the rotation angle of the Ergo and of the ball around the center of the arena.

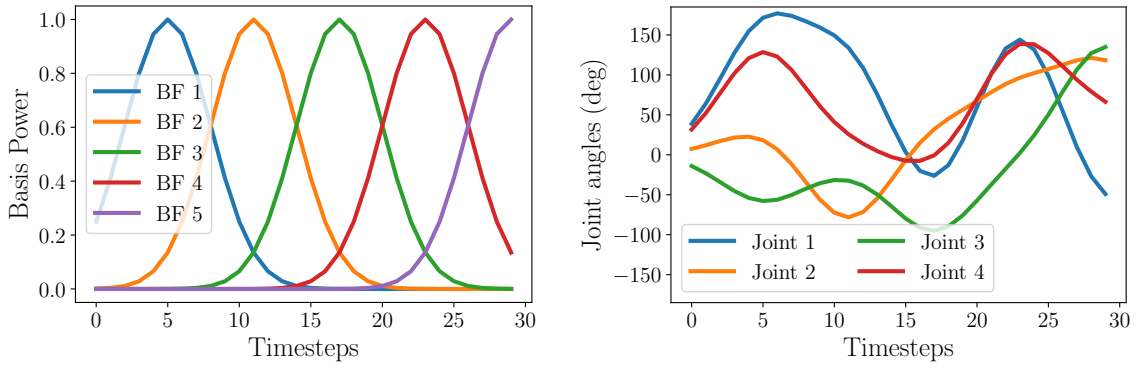
The agent is given a perceptual system providing sensory feedback that represents the trajectories of all objects in the scene. First, the 3D trajectory of the hand is computed through a forward model of the arm as its x, y and z position. The 2D states of each joystick and of the Ergo are read by sensors, and the position of the ball retrieved through the camera. The states of the 1D intensity of the light and the 1D pitch of the sound are computed from the ball position and speed. Each of the 15 objects defines a sensory space S_i representing its trajectory. The total sensory space S has 310 dimensions.

4.2 IMPLEMENTATION OF THE MODULAR POPULATION-BASED IMGEP ARCHITECTURE

In the following subsections, we detail our implementations of the algorithmic parts of the modular population-based IMGEP architecture (see architecture 2).

4.2.1 MOTOR POLICY π_θ

In the 2D Simulated environment and the Robotic environment, we implement the motor policies with Radial Basis Functions (RBF). We define 5 Gaussian basis functions with the same shape ($\sigma = 5$ for a 50 steps trajectory in the 2D environment and $\sigma = 3$ for 30 steps in the Robotic environment) and with equally spaced centers (see Fig. 6). The movement of each joint is the result of a weighted sum of the product of 5 parameters and the 5 basis. The total vector θ has 20 parameters, in both the 2D Simulated and the Robotic environment. In the 2D environment, the fourth joint is a gripper that is considered open if its angle is positive and closed otherwise.



(a) Basis Functions shape

(b) Example Joints Trajectory

Figure 6: Implementation of motor policies π_θ through Radial Basis Functions. (a) 5 Gaussian bases with different centers but same shape. (b) the movement of each joint is the result of a weighted sum of the product of 5 parameters and the 5 basis. The total vector θ has 20 parameters, in both the 2D Simulated and the Robotic environment.

In the Minecraft Mountain Cart environment, trajectories are sampled in a closed-loop fashion using neural networks. The observation vector has the same structure as the outcome vector: it provides the current positions of all objects normalized in $[-1; 1]$ (18D). Each neural network is composed of one hidden layer of 64 Relu units and a 3D output with tanh activation functions. The 1411 policy parameters are initialized using the initialization scheme of He et al. (2015).

4.2.2 STEPPING-STONE PRESERVING MUTATIONS

The Stepping-Stone Preserving Mutation operator (`SSPMutation`) does not change the variables of θ concerning the movement before the object moved and modifies the variables of θ concerning the movement after the object moved. `SSPMutation` adds a Gaussian noise around those values of θ in the 2D simulated environment ($\sigma = 0.05$) and in Minecraft Mountain Cart ($\sigma = 0.3$), or adds the Gaussian noise around the previous motor positions (in the robotic environment with joysticks). In the experimental section we compare it to the `FullMutation` operator that adds a Gaussian noise to θ irrespective of the moment when the target object moved.

4.2.3 GOAL SPACE POLICY Γ

The agent estimates its learning progress globally in each goal space (or for each model learned). At each iteration, the context c is observed, a goal space k is chosen by Γ and a random goal g is sampled by γ_k in $\mathcal{G}^k$ (corresponding to a fitness function f_g). Then, in 80% of the iterations, the agent uses $\Pi_\epsilon(\boldsymbol{\theta} \mid g, c)$ to generate with exploration a policy θ and does not update its progress estimation. In the other 20%, it uses Π, without exploration, to generate θ and updates its learning progress estimation in $\mathcal{G}^k$, with the estimated progress in reaching g. To estimate the learning progress r_i made to reach the current goal g, the agent compares the outcome o_τ with the outcome o'_τ obtained for the previous context and goal (g', c') most similar (Euclidean distance) to (g, c): $r_i = f_g(\tau) - f_g(\tau')$. Finally, Γ implements a non-stationary bandit algorithm to sample goal spaces. The bandit keeps track of a running average r_i^k of the intrinsic rewards r_i associated to the current goal space $\mathcal{G}^k$. With probability 20%, it samples a random space $\mathcal{G}^k$, and with probability 80%, the probability to sample $\mathcal{G}^k$ is proportional to r_i^k in the 2D Simulated and Minecraft environments, or $exp\big(\frac{r_i^k}{\sum_k r_i^k}\big)$ if $r_i^k > 0$ and 0 otherwise, in the Robotic environment.

4.2.4 CONTROL CONDITIONS

We design several control conditions.In the Random Model Babbling (RMB) condition, the choice of goal space is random: $\Gamma(\mathbf{k} \mid \mathbf{c})$, and $\gamma_k(\mathbf{g} \mid \mathbf{c})$ for each k are always uniform distributions. Agents in the Single Goal Space (SGS) condition always choose the same goal space, of high interest to the engineer: the magnet toy in the 2D Simulated environment, and the ball in the robotic environment. The Fixed Curriculum (FC) condition defines Γ as a curriculum sequence engineered by hand: the agents explore objects in a sequence from the easiest to discover to the most complex object while ignoring distractors. The conditions SGS and FC are thus extrinsically motivated controls. We define the Flat Random Goal Babbling (FRGB) condition with a single outcome/goal space containing all the variables of all objects, to compare modular and non-modular representations of the environment. The agents in this condition choose random goals in this space, and use the `FullMutation` operator. Finally, agents in the Random condition always choose random motor policies θ.

4.3 RESULTS

In this section we show the results of several experiments with the three environments and the different learning conditions. We first study in details the Active Model Babbling (AMB) learning algorithm, a modular implementation of the IMGEP architecture. Then, in order to understand the contribution of the different components of this learning algorithm, we compare it to several controls (or ablations): without a modular representation of goals, without the goal sampling based on learning progress, or without the stepping-stone preserving mutation operator. In those experiments, goals are sampled in spaces representing the sensory feedback from the environment. We thus compare several possible encodings of the feedback: with the trajectory of each object or with only the end point of the trajectories. We included distractors that cannot be controlled by the learning agent in the three tool-use environments. We also test the learning conditions with and without distractors to evaluate their robustness to distractors.

4.3.1 EXPLORATION MEASURE AND SUMMARY RESULTS

Table 1 shows a summary of the exploration results at the end of the runs, in all conditions in all spaces of all environments, We give the 25, 50 and 75 percentiles of the exploration results of all seeds. Exploration measures the percentage of reached cells in a discretization of each goal space. The best condition in each space is highlighted in bold, based on Welch's t-tests (with threshold $p < 0.05$): if several conditions are not significantly different, they are all highlighted. In the 2D Simulated environment, there are 100 seeds for each condition, and the exploration measures the

number of cells reached in a discretization of the 2D space of the end position of each object with 100 bins on each dimension. In the Minecraft environment, there are 20 runs for conditions Random, SGS, FRGB, FC and 42 for AMB and RMB. The exploration metric for the agent, pickaxe and shovel spaces is the number of reached cells in a discretization of the 2D space in 450 bins (15 on the x axis, 30 on the y axis). The same measure is used for the block space, which is discrete with 32 possible combinations. For the cart space we measure exploration as the number of different outcomes reached. In the Robotic environment, there are 6 runs with different seeds for condition SGS, 8 for FRGB, 16 for RMB, 23 for AMB, 12 for FC and 6 for Random, and the exploration also measures the number of cells reached in a discretization of the space of the end position of each object with 1000 bins in 1D, 100 bins on each dimension in 2D, and 20 bins in 3D.

Env, Space \ Condition		Rdm	SGS	Flat	RMB	AMB	FC
2D Simulated Environment	Magnet Tool	0,0,0	0,0,0	8.0,11,13	33,36,39	57,**61**,65	61,**67**,70
	Magnet Toy	0,0,0	0,0,0	0,0,0	0,0,5.0	0,**3.0**,16	0,**3.0**,19
Minecraft Mountain Cart	Agent Pos.	28,29,30	29,29,30	34,36,40	48,50,54	55,58,61	59,**63**,67
	Shovel	5,5,6	5,6,7	8,11,13	25,27,30	32,34,37	34,**37**,42
	Pickaxe	6,6,7	6,7,8	11,15,19	33,35,39	41,45,48	43,**51**,61
	Blocks	3,3,3	3,3,3	3,11,19	69,77,84	73,**84**,93	100,**100**,100
	Cart	0,0,0	0,0,0	0,0,1	5,162,409	56,360,886	386,**787**,1207
Robotic Environment	Hand	24,24,25	18,19,20	20,21,22	22,24,25	22,23,24	21,22,23
	L. Joystick	4.2,4.7,5.9	1.9,3.3,4.6	0.1,0.1,0.3	15,18,19	20,**22**,26	23,**26**,29
	R. Joystick	0.6,0.9,1.0	0.3,0.4,0.5	0,0,0	10,11,13	16,**18**,22	15,17,18
	Ergo	0.2,0.3,0.4	0.1,0.1,0.2	0,0,0	1.2,**1.5**,1.7	1.5,**1.7**,1.8	1.7,**1.7**,1.9
	Ball	0,0,0.1	0,0,0	0,0,0	0.8,1.0,1.0	0.9,**1.1**,1.2	0.9,**0.9**,1.0
	Light	0.1,0.1,0.1	0.1,0.2,0.2	0.1,0.1,0.1	0.8,1.8,3.0	2.0,**3.6**,4.9	1.8,**2.2**,3.7
	Sound	0.1,0.1,0.1	0.1,0.1,0.1	0.1,0.1,0.1	0.8,1.1,2.6	1.7,**2.8**,3.6	1.2,1.6,2.3

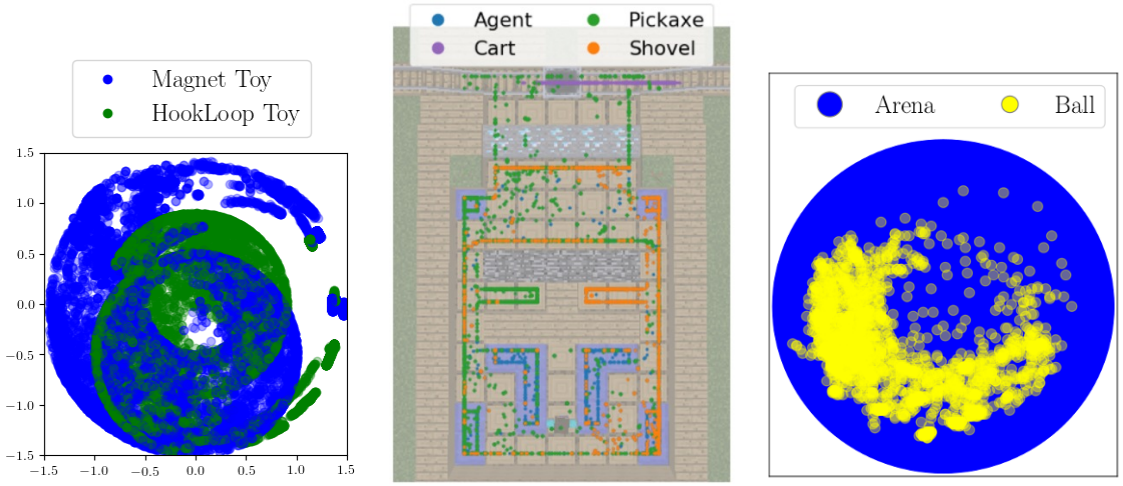
Table 1: Exploration results in all environments and conditions.

4.3.2 INTRINSICALLY MOTIVATED GOAL EXPLORATION

Here we study in detail the Active Model Babbling (AMB) learning algorithm. AMB agents encode the sensory feedback about objects with a modular representation: each object is associated with one independent learning module. At each iteration, they first select an object to explore, then a particular goal to reach for this object. They execute a motor policy to reach this goal, and observe the outcome. The selection of the object to explore is based on a self-estimation of the learning progress made to move each object according to chosen goals. The AMB algorithm is thus a modular implementation of the IMGEP architecture.

Exploration Maps We first plot examples of exploration results as cumulative exploration maps, one per environment. Those maps show all the positions where one AMB agent succeeded to move objects.

Fig. 7(a) shows the position of the reachable toys of the 2D simulated environment at the end of each iteration in one trial of intrinsically motivated goal exploration. The reachable area for those two toys is the inside the circle of radius 1.5 and center 0. We can see that in 100k iterations, the agent succeeded to transport the toys in many places in this area. The experiments with other seeds are very similar. Fig. 7(b) shows an exploration map of a typical run in Minecraft Mountain Cart after 40k iterations. As you can see the agent successfully managed to (1) navigate within the arena boundaries, (2) move the pickaxe and shovel, (3) use the pickaxe to break blocks and (4) move the cart located behind these blocks. An example in the robotic environment is shown in Fig. 7(c) where



(a) 2D Simulated Environment (b) Minecraft Mountain Cart (c) Robotic Environment

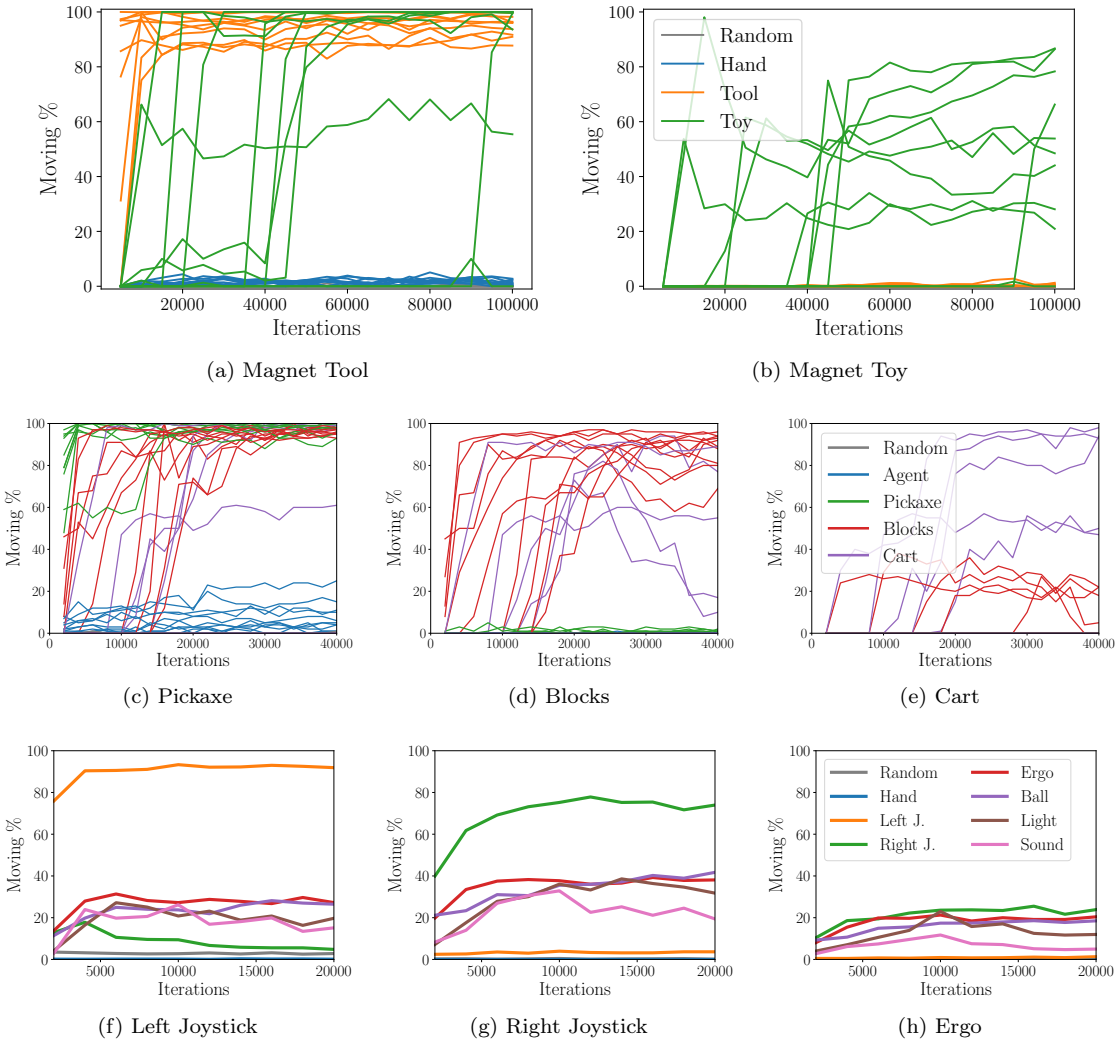
Figure 7: Examples of exploration map of one IMGEP agent in each environment. (a) in the 2D Simulated Environment, we plot the position of the reachable magnet toy at the end of each iteration with a blue point, and the Velcro toy in green. (b) in Minecraft Mountain Cart we plot the end position of the agent, the agent with pickaxe, the agent with shovel, and the cart. (c) in the Robotic environment, the position of the ball is plotted when it moved in the arena.

we plot the position of the ball when it moved in the first $10k$ iterations of the exploration of one agent.

Overall, they show that IMGEP agents discovered how to use the different tools in each environment within the time limit: the sticks to grab the toys in the 2D simulated environment, the pickaxe to mine blocks to reach the cart in Minecraft Mountain Cart, the joysticks to move the toy and push the ball in the robotic experiment.

Discoveries In order to understand the tool-use structure of the exploration problem in each environment, we can look in more details how agents succeeded to move objects while exploring other objects. Indeed, to the agents starting to explore, tools are objects like any other object (e.g. the hand, the stick and the ball have the same status). However, if a tool needs to be used to move another object, then this tool will be discovered before that object, so the exploration of this tool is a stepping-stone giving more chances to discover novelty with that object than the exploration of any other object. To quantify these dependencies between objects in our tool-use environments, we show the proportion of movements where an object of interest has been moved depending on the currently explored object.

Concerning the 2D simulated environment, Fig. 8 shows the proportion of the iterations with a goal in a given space that allowed to move (a) the magnet tool, (b) the magnet toy, in 10 runs with different seeds. First, random movements of the arm have almost zero chances to reach the magnet tool or toy. Exploring movements of the hand however have about 1.5% chances to move the magnet tool, but still almost zero chances to reach the toy. Exploring the magnet tool makes this tool move in about 93% of the iterations, and makes the toy move in about 0.1% of movements. Finally, exploring the toy makes the tool and the toy move with a high probability as soon as the toy was discovered. Those results illustrate the stepping-stone structure of this environment, where each object must be well explored in order to discover the next step in complexity (Hand → Tool → Toy).



(a) Magnet Tool (b) Magnet Toy

(c) Pickaxe (d) Blocks (e) Cart

(f) Left Joystick (g) Right Joystick (h) Ergo

Figure 8: Stepping-stone structure of the three environments. In the 2D Simulated environment, we show the proportion of iterations that allowed to (a) move the magnet tool, (b) move the magnet toy, depending on the currently explored goal space (or random movements), for 10 IMGEP agents. The fastest way to discover the tool is to explore the hand and to discover the toy is to explore the tool. In the Minecraft Mountain Cart environment, we show the proportion of iterations that allowed to (c) move the pickaxe, (d) mine diamond blocks, and (e) move the cart, depending on the currently explored goal space (or random movements), for 10 agents with different seeds. Exploring the agent space helps discover the pickaxe, exploring the pickaxe helps discover the blocks, and exploring the blocks helps discover the cart. In the Robotic environment, we show the proportion of iterations that allowed to (f) reach the left joystick, (g) reach the right joystick, and (h) move the Ergo robot, depending on the currently explored goal space (or random movements), averaged for 11 IMGEP agents with different seeds. Exploring random movements or the Hand space helps discover the left joystick, exploring the left joystick helps discover the right one, which helps discover the Ergo toy.

	Pickaxe goals	Cart goals
FC	39,49,55	12,17,25
AMB	41,45,49	8,11,18
RMB	37,40,43	6,9,15
SGS	N/A	0,0,0

Table 2: Competence results in Minecraft Mountain Cart. We give the 25, 50 and 75 percentiles of the competence results of all seeds.

In Minecraft Mountain Cart (see Fig. 8(c,d,e)), random exploration with neural networks in this environment is extremely challenging. An agent following random policies has 0.04% chances to discover the pickaxe, 0.00025% chances to break a single block and it never managed to move the cart (over 800k episodes). IMGEP agents reach better performances by leveraging the sequential nature of the environment: when exploring the agent space there is around 10% chances to discover the pickaxe, and exploring the pickaxe space has around 1% chances to break blocks. Finally, exploring the block space has about 8% chances to lead an agent to discover the cart.

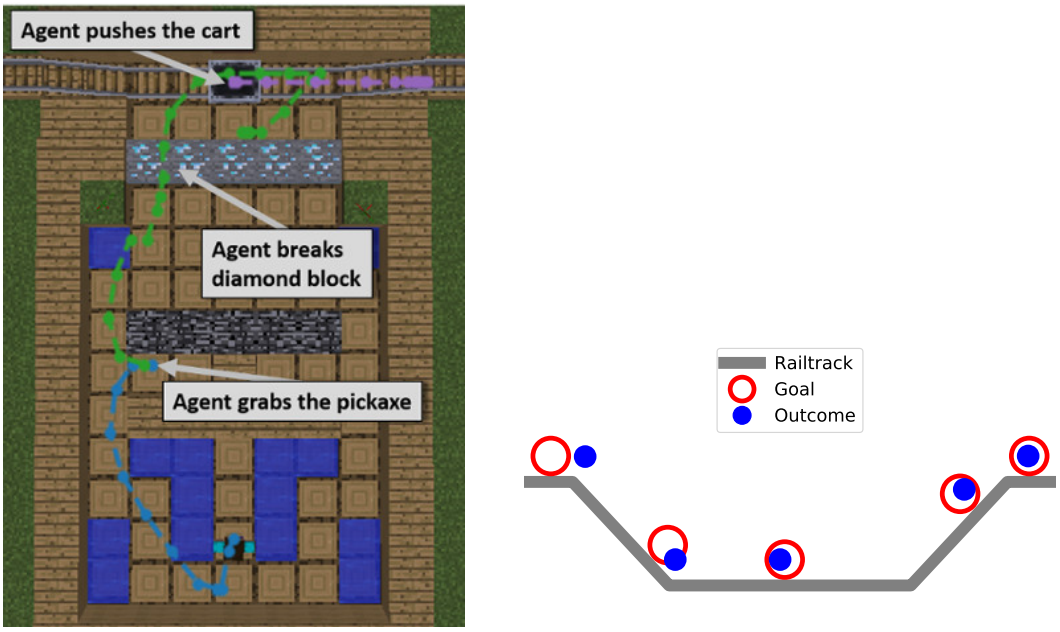
In the Robotic environment, a similar stepping-stone exploration structure is displayed (see Fig. 8(f,g,h)): in order to discover the left joystick, the robots needs to do random movements with its arm, which have about 2.9% chances to makes the left joystick move, or explore its hand (0.2% chance). To discover the right joystick, the agent has to explore the left joystick, which gives a probability of 3.3% to reach the right one. To discover the Ergo (the white robotic toy in the center of the blue arena), the exploration of the right joystick gives 23% chances to move it, whereas the exploration of the Hand, the left joystick or random movements has a very low probability to make it move.

4.3.3 LEARNED SKILLS

In Minecraft Mountain Cart we performed post-training tests of competence in addition of exploration measures. Using modular approaches allows to easily test competence on specific objects of the environment. Fig. 9(b) shows an example in the cart space for an AMB agent. This agent successfully learned to move the cart close to the 5 queried locations.

For each of the RMB, AMB and FC runs we performed a statistical analysis of competence in the cart and pickaxe spaces using 1000 and 800 uniformly generated goals, respectively. We were also able to test SGS trials for cart competence as this condition has the cart as goal space. A goal is considered reached if the Euclidean distance between the outcome and the goal is lower than 0.05 in the normalized space (in range $[-1, 1]$) for each object. Since the pickaxe goal space is loosely defined as a rectangular area around the environment's arena, many goals are not reachable. Results are shown in Table 2. SGS agents never managed to move the cart for any of the given goals. AMB appears to be significantly better than RMB on the pickaxe space ($p < 0.01$ on Welch's t-tests). However it is not in the cart space ($p = 0.09$), which might be due to the stochasticity of the environment. FC is not significantly better than AMB on the cart and pickaxe spaces.

Intrinsic Rewards based on Learning Progress The IMGEP agents self-evaluate their learning progress to control each object. When they choose a goal for an object, they monitor what is the actual movement given to the object and compare it to the goal. If the distance between the goals and the actual reached movements decrease over time on average, this tells the agents it is making progress to control this object. This signal is used as an intrinsic reward signal that the agent will seek to maximize by choosing to explore objects that yield a high learning progress. We can analyze this signal to understand at which point the agent perceived progress to control each object and how its exploration behavior changed over time.



(a) Agent moving the cart

(b) 5 cart goals

Figure 9: Example of learned skills in the Minecraft Mountain Cart. (a) One AMB agent's trajectory for a single cart goal. (b) Five final cart positions reached by an AMB agent when tasked to reach five different targets. This agent successfully learned to push the cart along the track.

Fig. 10 (top) shows the intrinsic rewards of two agents (different seeds) to explore each object in the 2D simulated environment, computed by the agents as the average of intrinsic rewards based on learning progress to move each object. We can see that the intrinsic reward of the hand increases first as it is the easiest object to move. Then, when the sticks are discovered, the agents start to make progress to move them in many directions. Similarly, while exploring the sticks, they discover the reachable toys, so they start making progress in moving those toys. However, the static objects can't be moved so their learning progress is strictly zero, and the objects moving randomly independently of the agent (cat and dog) have a very low progress.

Fig. 10 (middle) shows the intrinsic reward of two agents in the Minecraft Mountain Cart environment. Both agents first explore the simpler agent space and then quickly improves on the shovel and pickaxe spaces. Exploring the pickaxe space leads to discover how to progress in the block space. Finally, after some progress in the block space, the cart is discovered after 14k episodes for the first agent (left figure) and 26k episodes for the other (right figure). The 3 distracting flowers have an interest strictly equal to zero in both runs.

Fig. 10 (bottom) shows the intrinsic reward of two agents in the Robotic environment. The first interesting object is the robot's hand, followed by the left joystick and then the right joystick. The left joystick is the easiest to reach and move so it gets interesting before the right one in most runs, but then they have similar learning progress curves. However, the right joystick can be used as a tool to control other objects, so that one will be touched more often. Then, the agent can discover the Ergo and Ball while exploring the joysticks. Finally, some agents also discover that the ball can be used to make light or sound. Here also, the progress of static objects is zero and the one of random objects is low.

Overall, the evolution of those interests show that evaluating the learning progress to move objects allows agents to self-organize a learning curriculum focusing on the objects currently yielding the most progress and to discover stepping stones one after the other.

4.3.4 INFLUENCE OF GOAL MODULARITY

In this section, we study several algorithms with a different goal space structure in order to evaluate the influence of the modularity of the goal space. We compare the Active Model Babbling condition to other conditions. In the Flat Random Goal Babbling (FRGB) condition, the goal space is not modular and contains all variables of all objects. With this non-modular sensory representation, agents choose goals in the whole sensory space, which corresponds to all objects: a goal could mean for instance push *toy1* to the left and *toy2* to the right at the same time, which might be unfeasible. This exploration dynamics results in exploring the most diverse global sensory states, which is akin to novelty search algorithms (Lehman and Stanley, 2011a). We also test the Random control where agents always choose random actions.

In the 2D simulated environment, we run 100 trials of each condition with different random seeds. We measure the exploration of one stick and its corresponding toy as the cumulative number of reached cells in a discretization of the 2D space of the position of each objects at the end of movements. Fig. 11 shows the evolution of the exploration of the stick and the toy in the 100 trials of each condition. We plot in bold the median over the 100 trials in each condition. We can see that the modularity of the goal space helps exploration: the median exploration after 100k iterations is about 60% of the magnet tool space for condition AMB vs about 10% for condition FRGB. The agents in condition AMB succeeded to reach the magnet toy, with a substantial variance between the 100 trials. Some AMB agents explored very well the magnet toy (up to 60%) and some did not (very low exploration). Finally, completely random agents did not even manage to explore the magnet tool.

Fig. 13 shows exploration results in the Minecraft Mountain Cart environment for 20 trials of all conditions except for AMB and RMB which were run 42 times. When looking at the median exploration in the pickaxe space, FRGB does not manage to reach more than 15% exploration when AMB and RMB reached 45% and 35%, respectively. Modular approaches significantly outperform FRGB across all goal spaces (Welch's t-tests at 40k iterations, $p < 0.001$). Random agents did not manage to explore the block and cart spaces.

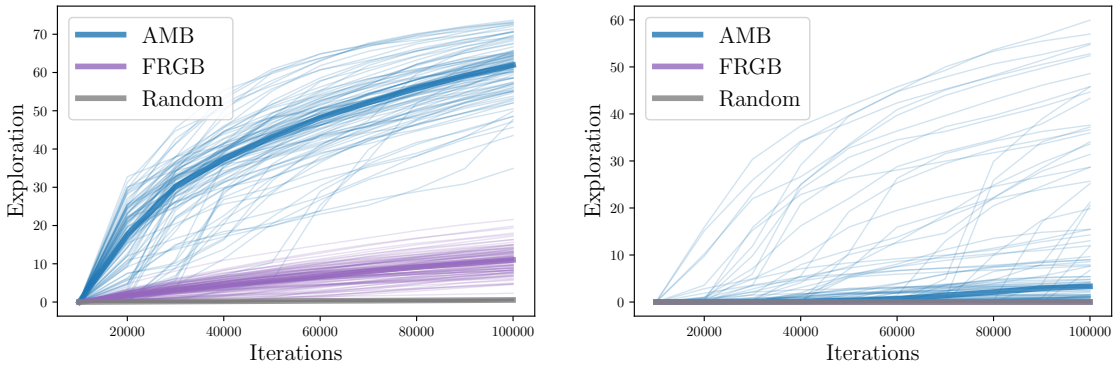
In the robotic environment (see Fig. 14), agents with the flat (intricate) representation of the sensory feedback (FRGB) do not explore objects other than the hand.

The modular representation of the sensory space thus greatly improves exploration efficiency compared to a flat intricate representation of the whole sensory feedback, as it allows to consider the different objects independently to monitor their behavior and select disentangled goals.

4.3.5 CURRICULUM LEARNING

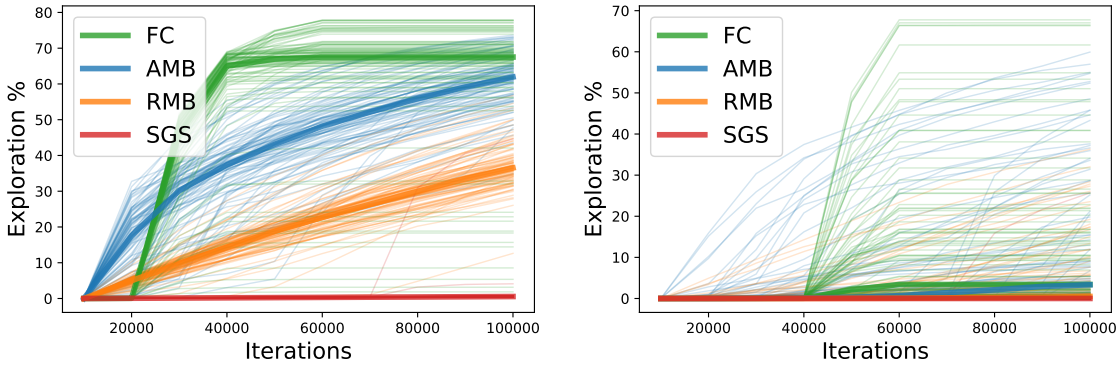
A modular sensory representation based on objects allows AMB agents to self-monitor their learning progress to control each object, and to accordingly explore objects with high learning progress. Here, we compare several variants of agents with a modular sensory representation based on objects, but with a different choice of object to explore. To evaluate the efficiency of the sampling based on learning progress, we define condition Random Model Babbling (RMB), where agents always choose objects to explore at random. The sampling based on learning progress of AMB agents makes agents explore any object that shows learning progress, and ignore objects that do not move, are fully predictable, or move independently of the agent. However if we are only interested in a particular complex skill that we want the agent to learn, such as moving the ball in the robotic environment, then it is not obvious if supervising learning by specifying a curriculum targeted at this skill can accelerate the learning of this skill. We thus define two control conditions with a hand-designed curriculum. In condition Single Goal Space (SGS), agents always choose goals for the same complex target object: the magnet toy in the 2D simulated environment, the cart in Minecraft Mountain Cart,

Figure 10: Examples of intrinsic rewards in the three environments. In the 2D Simulated environment (top), agents are first interested in exploring their hand as this is the only object they manage to move, until they discover one of the sticks. Then they make progress to move the stick, so the intrinsic reward for moving the stick increases and they focus on it, and then on the other objects they discover: the other stick and the two toys. They make no progress to move the distractors so those intrinsic reward are always zero. In the Robotic environment (middle), agents first succeed to move their hand, so they focus on this object at the beginning, until they discover the joysticks. The exploration of the right joystick makes them discover the Ergo toy, which can push the Ball. Some agents also discover how to produce light and sound with the Ball. Agents have a low intrinsic reward for exploring random distractors. In Minecraft Mountain Cart (bottom), agents first focus on exploring the space of their position until they discover the shovel or the pickaxe and start making progress to move them. When they discover how to mine blocks with the pickaxe and to push the cart, they make progress in those goal spaces, get intrinsic rewards and thus focus more on these.



(a) Magnet Tool (b) Magnet Toy

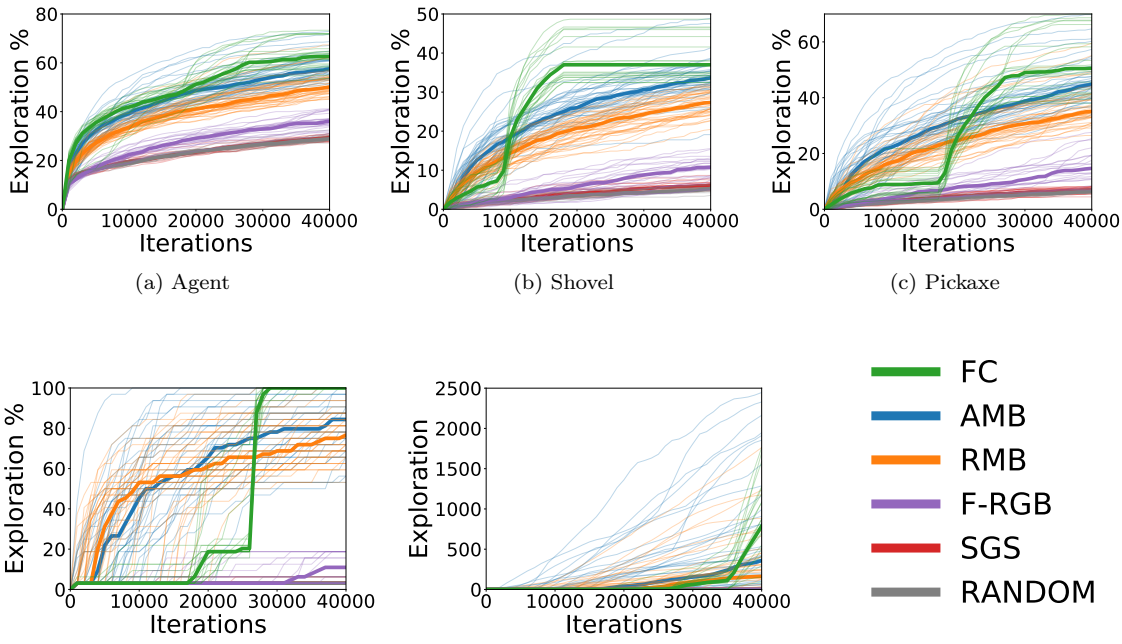
Figure 11: Influence of Goal Modularity on exploration in the 2D simulated environment. The agents using a modular representation (Active Model Babbling) explore much better the tool and toy spaces than agents with a flat representation (Flat Random Goal Babbling). Control agents always choosing completely random actions do not manage to touch a toy with the stick.



(a) Magnet Tool (b) Magnet Toy

Figure 12: Influence of curriculum learning on exploration in the 2D Simulated environment. Agents self-organizing their curriculum (Active Model Babbling) based on their learning progress explore better than agents choosing to explore random objects (Random Model Babbling) or agents choosing always to explore the magnet toy (Single Goal Space). Agents with a hard-coded curriculum learning sequence from the simpler objects to the most complex have similar exploration results than autonomous AMB agents after 100k iterations.

or the ball in the robotic environment. In condition Fixed Curriculum (FC), a particular sequence of exploration is specified, from the easier stepping-stones to the more complex ones. In the 2D simulated environment, the agent samples goals for 20k iterations on each object in this order: hand, magnet tool, magnet toy, Velcro tool, Velcro toy. In the robotic environment, we define the sequence as the following: hand, left joystick, right joystick, ergo, ball, light and sound.



(a) Agent (b) Shovel (c) Pickaxe

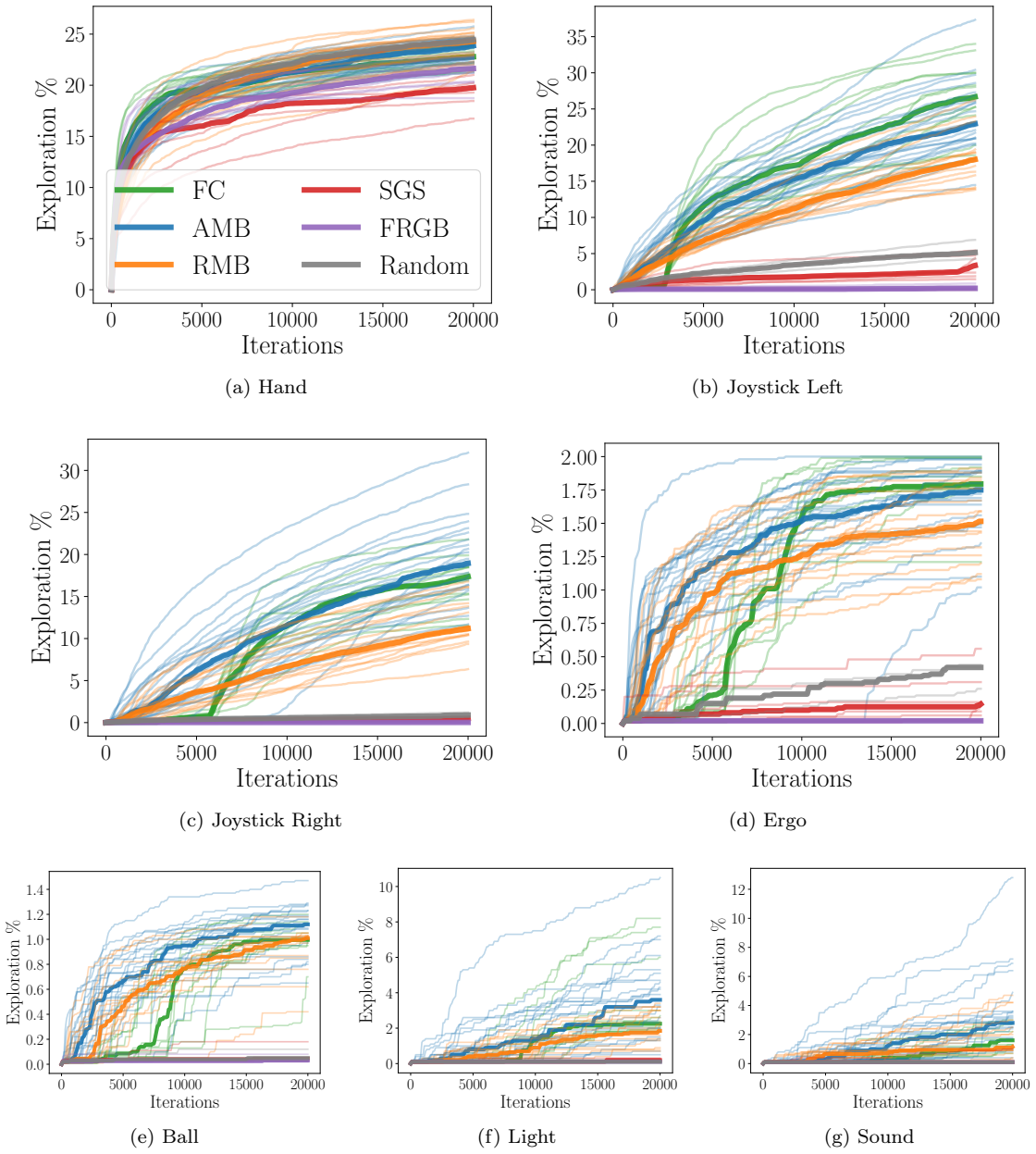
(d) Blocks (e) Cart

Figure 13: Exploration results in Minecraft Mountain Cart. Modular approaches (AMB and RMB) performs significantly better than the flat (F-RGB) approach. Agents actively generating their curriculum (AMB) perform better overall than agents choosing goal spaces randomly (RMB). Focusing on the cart space (SGS) is equivalent to performing random policies (Random). For the agent, pickaxe and shovel spaces, exploration is measured as the cumulative number of reached cells in a discretization of the 2D space. For the block and cart spaces we measure the number of unique outcomes reached.

Fig. 12 shows the exploration evolution in the 2D simulated environment. First, we can see that the sampling based on learning progress (AMB) helps exploration of the tool and the toy compared to the random choice of object to explore (RMB): 62% vs 37% for the tool and 3.3% vs 0.5% for the toy. Agents in the SGS condition did not manage to explore the tool and the toy. Agents with a predefined curriculum succeeded to explore the tool and the toy very well, the tool between 20k and 40k iterations and the toy between 40k and 60k iterations, with a median slightly better than in AMB.

Fig. 13 shows the evolution of exploration in Minecraft Mountain Cart. Agents focusing their goal sampling on the cart space (SGS) have low performances across all goal spaces, especially for the cart and block spaces which are never discovered. Agents using learning progress sampling (AMB) explored significantly more than random sampling agents (RMB) across all goal spaces (Welch's t-tests at 40k iterations, $p < 0.04$). Agents following a hard-coded curriculum (FC) reached higher median performances than AMB agents on every goal spaces.

Fig. 14 shows exploration results in the robotic environment. Agents self-organizing their curriculum (Active Model Babbling) based on their learning progress explore better than agents choosing random objects (Random Model Babbling) in the joysticks, ball, light and sound spaces (Welch's t-tests at 100k iterations, $p < 0.05$), and better than agents with a hard-coded curriculum sequence (FC) in the right joystick and sound spaces. Agents always choosing to explore the ball



(a) Hand

(b) Joystick Left

(c) Joystick Right

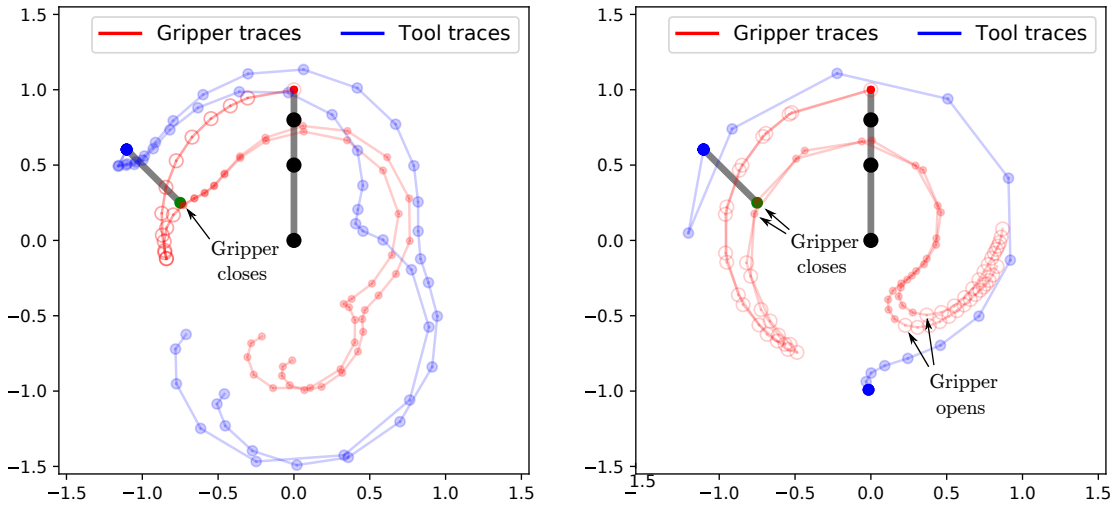
(d) Ergo

(e) Ball

(f) Light

(g) Sound

Figure 14: Exploration results in the Robotic environment. Agents self-organizing their curriculum (Active Model Babbling) based on their learning progress explore better than agents choosing random objects (Random Model Babbling) in the joysticks, ball, light and sound spaces, and better than agents with a hard-coded curriculum sequence (FC) in the right joystick and sound spaces. Agents always choosing to explore the ball (Single Goal Space) and agents without a modular representation of goals (FRGB) have low exploration results in all spaces.



(a) `SSPMutation` (b) `FullMutation`

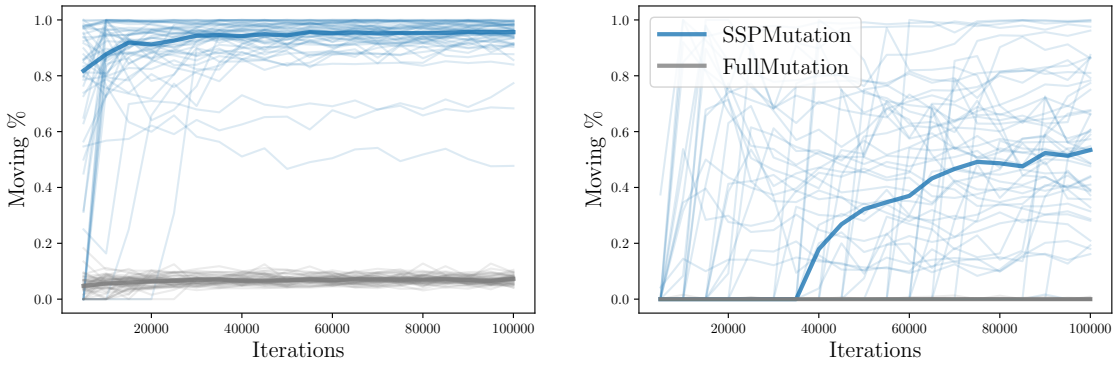
Figure 15: Example of a mutation of a movement reaching the tool, for each mutation operator. With the `SSPMutation` operator, the two gripper trajectories start to diverge only when the stick is grasped such that the mutated movement also grasps the stick, whereas with the `FullMutation`, the mutation starts right from the beginning of the movement, which in this case makes the mutated movement miss the stick.

(SGS) and agents without a modular representation of goals (FRGB) have low exploration results in all spaces.

Overall, the goal sampling based on learning progress (AMB) improves exploration of most objects of each environment compared to a random choice of object (RMB), as those agents focus on objects that are learnable, ignore the distractor objects and reduce the relative interest of objects already explored for some time. Specifying the curriculum by hand results in a very bad exploration if the agent always directly focuses on an object hard to discover, however if we carefully design the learning sequence given our knowledge of the task, then the final exploration results are similar to autonomous AMB agents.

4.3.6 Influence of the modularity of exploration mutations

The efficiency of the Stepping-Stone Preserving Mutation operator (`SSPMutation`, see Sec. 3.3) relies on its ability to preserve the part of the movement that reaches a stepping-stone in the environment, and explore only after the target object started to move in the previous movement being mutated. For instance, the movement would grab the tool without modification, and explore once the controlled toy started to move. To illustrate this mechanism, let us look at actual mutations depending on the mutation operator. Fig. 15 shows one movement that reached the magnet tool together with one mutation of this movement, for each mutation operator. The red trajectories are the traces of the gripper (with a circle when open and a point when closed), and the blue trajectories are the traces of the magnet stick. We also plot the initial position of the arm and the magnet stick. We see that in the case of the `SSPMutation` operator, the two red trajectories start to diverge only when the stick is grasped such that the mutated movement also grasps the stick, whereas with the `FullMutation`, the mutation starts right from the beginning of the movement, which in this case makes the mutated movement miss the stick.



(a) Magnet Tool (b) Magnet Toy

Figure 16: Comparison of the `SSPMutation` and `FullMutation` mutation operators. We show the proportion of iterations that allowed to (a) move the magnet tool while exploring this tool, and (b) move the magnet toy while exploring this toy, with 50 different seeds and median. With the `FullMutation` operator, at the end of the runs agents succeed to move the tool in 7% of iterations versus 95% for the `SSPMutation` operator, and to move the toy in 0.9% of iterations versus 53%.

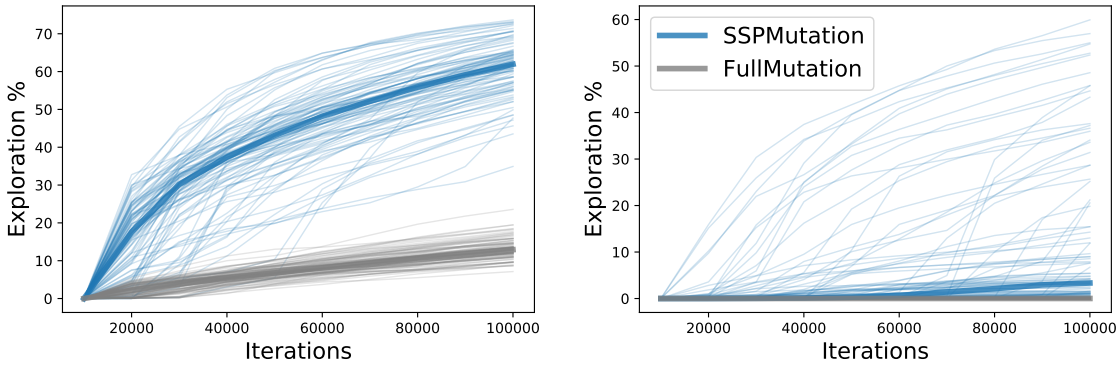
We measure how many times the agents succeed to move a tool when they are exploring it, or to move a toy when they are exploring the toy, depending on the mutation operator. Fig. 16 shows the proportion of the iterations that allowed to (a) move the magnet tool when this tool is the goal object, (b) move the magnet toy when this toy is the target object, with 50 different runs in the 2D simulated environment (individual runs and median). We can see that with the `FullMutation` operator, at the end of the runs agents succeed to move the tool in 7% of iterations targeted at exploring this tool, versus 95% for the `SSPMutation` operator, and to move the toy in 0.9% of iterations targeted at exploring this toy versus 53%.

The ability of `SSPMutation` to explore while still moving the target object with a high probability directly improves exploration. Fig. 17 shows the exploration results of AMB agents with the `SSPMutation` or `FullMutation` operators in the 2D simulated environment in 100 runs with different seeds. The exploration results of the `FullMutation` operator are much lower for the magnet tool (median 13% vs 62%) and magnet toy (median 0% vs 3%, max 0.5% vs 60%).

4.3.7 ENCODING OF GOALS

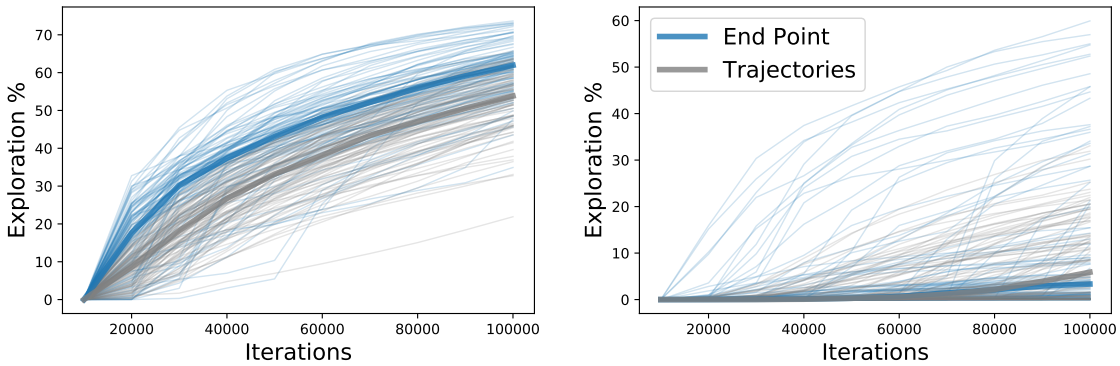
After executing a movement, the agent receives a sensory feedback containing information about the movement of objects in the environment. The agent then uses the sensory space as an encoding for sampling new goals to reach. In the 2D simulated environment, we defined the sensory feedback as the position of each object at the end of the movement. In the robotic environment, as the joysticks may move during the movement but come back by themselves at their rest position, we used a sensory feedback containing information about the whole trajectory of objects as the sequence of their position at 5 time steps during the movement. In this section, we study the influence of the goal encoding on exploration efficiency in the 2D simulated environment.

Fig. 18 shows the exploration evolution of AMB agents depending on the encoding of goals: with object trajectories or end points, in the 2D simulated environment. The exploration is a measure of the proportion of cells reached in a discretization of the space of the last position of each object. The encoding with the end position of each object resulted in a slightly better exploration than with



(a) Magnet Tool (b) Magnet Toy

Figure 17: Exploration efficiency depending on the mutation operator. `FullMutation` results in a much lower exploration for the magnet tool and toy compared to the stepping-stone preserving operator.



(a) Magnet Tool (b) Magnet Toy

Figure 18: Exploration in 2D simulated environment depending on the goal encoding. The encoding with the end position of objects resulted in a slightly better exploration than with trajectories for the magnet tool, and a similar median for the magnet toy but with more variance: standard deviation of 17% vs 8.6% at 100k iterations for the magnet toy.

trajectories for the magnet tool, and a similar median for the magnet toy but with more variance: with standard deviation of 17% vs 8.6% at 100k iterations for the magnet toy. The trajectory encoding represents the whole trajectory of each object instead of the final point only. This is not strictly needed to represent if a tool or a toy has moved in this environment so the end point encoding may be more efficient once the objects are discovered, however the trajectory encoding helps to explore trajectories with more diversity, for the hand or other objects, and thus to discover hard objects in the first hand. With the trajectory encoding, the exploration of objects difficult to move the first time is thus slower once discovered, but they are more often discovered.

Fig. 19 shows examples of interest curves with the goal encoding using trajectories of objects. As the goal spaces are of much larger dimensionality using the object trajectories than with the end

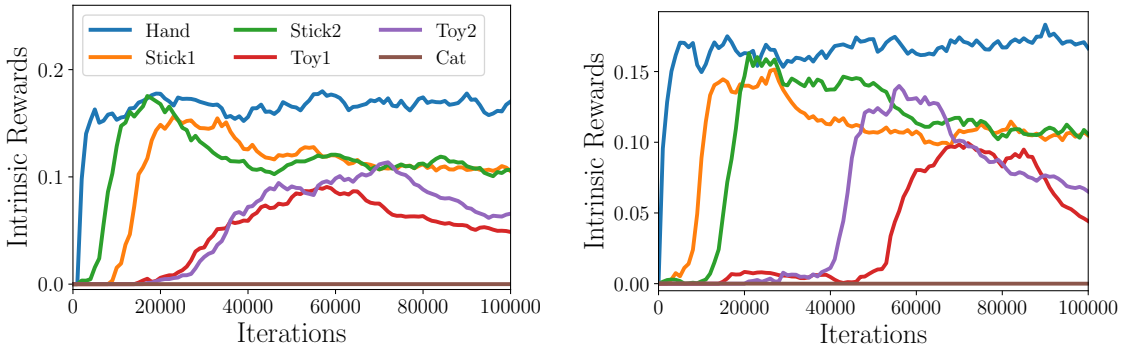
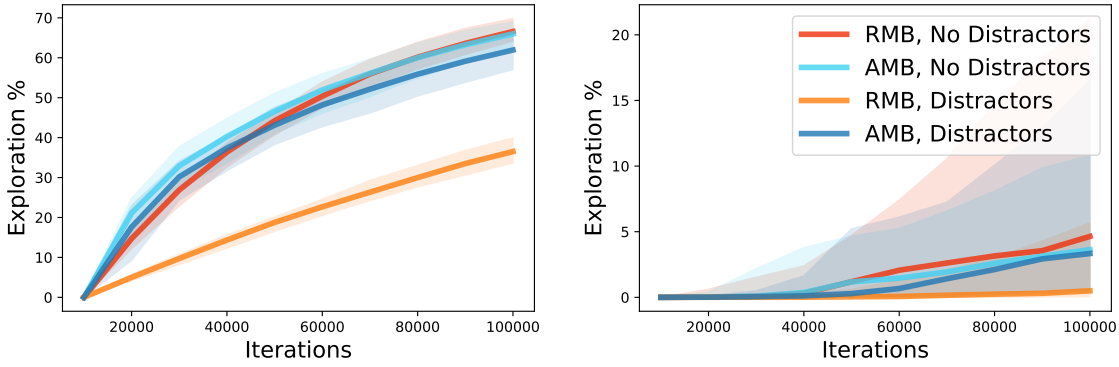


Figure 19: Examples of intrinsic rewards in the 2D simulated environment, with an encoding of goals as object trajectories. The sensory spaces are higher-dimensional and take more iterations to be well covered such that the learning progress decreases.



(a) Magnet Tool

(b) Magnet Toy

Figure 20: Exploration efficiency of Active Model Babbling and Random Model Babbling depending on the presence of distractors in the 2D Simulated environment. The median of 100 runs is plotted together with a shaded area representing the 25-75 percentiles. The efficiency of RMB agents decreases when we add distractors, whereas AMB agents, through their self-estimation of their learning progress to move each object, focus on learnable objects despite having distractors in the environment.

position, it takes a longer time to cover the whole sensory space with reached trajectories such that the self-computed interest to explore the hand is higher than with end positions (comparing with Fig. 10(top)) and the interest in all spaces takes more time to decrease.

The trajectory encoding is more interesting in environments where the full trajectory of a tool is of importance to control an object, such as in our robotic environment where joysticks come back at their rest position by themselves such that their end position is not informative to predict the end position of the controlled object. We thus use this trajectory encoding in the robotic environment, but we use the end point encoding in the Minecraft Mountain Cart environment.

4.3.8 STATIC AND RANDOM DISTRACTORS

In the three tool-use environments, we included distractor objects to harden exploration as those objects can't be controlled by the agent but are however part of their sensory feedback: some of them are static, and some of them move independently of the agent. The Active Model Babbling agents monitor their learning progress to move objects, such that their estimation of their progress to move static object is zero, and to move other uncontrollable objects is low compared to controllable objects. Here we evaluate the exploration efficiency of AMB and RMB agents in the 2D simulated environment in the presence and absence of distractors to evaluate their robustness to distractors.

Fig 20 shows the exploration results depending on the learning condition (RMB vs AMB) and environment condition: with 2 random distractors and 8 static distractors (Distractors) vs without distractors (No Distractors), in the magnet tool and toy spaces (median and 25/75 percentiles of 100 seeds). The RMB agents show a similar exploration without distractors compared to AMB agents. However, we can see that the RMB agents do not cope with distractors while AMB agents do not show a significant decrease in exploration when distractors are added. The learning progress monitoring is thus an efficient mean to discriminate learnable objects from other objects and thus to focus exploration by choosing most goals for learnable objects.

5. RELATED WORK

Early models of intrinsically motivated reinforcement learning (also called curiosity-driven learning) have been used to drive efficient exploration in the context of target tasks with rare or deceptive rewards (Schmidhuber, 1991; Barto, 2013) or in the context of computational modeling of open-ended unsupervised autonomous learning in humans (Kaplan and Oudeyer, 2004; Oudeyer et al., 2007). Reviews of the historical development of these methods and their links with cognitive sciences and neuroscience can be found in Baldassarre and Mirolli (2013); Gottlieb et al. (2013); Oudeyer et al. (2016).

Several lines of results have shown that intrinsically motivated exploration and learning mechanisms are particularly useful in the context of learning to solve reinforcement learning problems with sparse or deceptive rewards. For example, several state-of-the-art performances of Deep Reinforcement Learning algorithms, such as letting a machine learn how to solve complex video games, have been achieved by complementing the extrinsic rewards (number of points won) with an intrinsic reward pushing the learner to explore for improving its predictions of the world dynamics (Bellemare et al., 2016; Houthooft et al., 2016). An even more radical approach for solving problems with rare or deceptive extrinsic rewards has been to completely ignore extrinsic rewards, and let the machine explore the environment for the sole purpose of learning to predict the consequences of its actions (Schmidhuber, 1991; Oudeyer et al., 2007), to achieve self-generated goals (Baranes and Oudeyer, 2013; Oudeyer and Kaplan, 2007), or to generate novel behavioral features (Lehman and Stanley, 2011a). This was shown for example to allow agents to learn to play some video games without ever observing the extrinsic reward (Pathak et al., 2017).

Some approaches to intrinsically motivated exploration have used intrinsic rewards to value visited actions and states through measuring their novelty or the improvement of predictions that they provide, e.g. Sutton (1990); Dayan and Sejnowski (1996); Schmidhuber (1991); Oudeyer et al. (2007) or more recently Bellemare et al. (2016); Houthooft et al. (2016); Pathak et al. (2017). However, organizing intrinsically motivated exploration at the higher level of goals, by sampling goals according to measures such as competence progress (Oudeyer and Kaplan, 2007), has been proposed and shown to be more efficient in contexts with high-dimensional continuous action spaces and strong time constraints for interaction with the environment (Baranes and Oudeyer, 2013).

Several proposed methods are related to IMGEP, including Gregor et al. (2016), Dosovitskiy and Koltun (2016) and Kulkarni et al. (2016), however they have considered notions of goals restricted to the reaching of states or direct sensory measurements, did not consider goal-parameterized rewards

that can be computed for any goal, used different intrinsic rewards, and did not evaluate these algorithms in robotic setups. The notion of auxiliary tasks is also related to IMGEP in the sense that it allows a learner to acquire tasks with rare rewards by adding several other objectives which increase the density of information obtained from the environment (Jaderberg et al., 2016; Riedmiller et al., 2018). Another line of related work (Srivastava et al., 2013) proposed a theoretical framework for automatic generation of problem sequences for machine learners, however it has focused on theoretical considerations and experiments on abstract problems.

Several strands of research in robotics have presented algorithms that instantiate such intrinsically motivated goal exploration processes (Baranes and Oudeyer, 2010a; Rolf et al., 2010), using different terminologies such as contextual policy search (Kupcsik et al., 2017; Queißer et al., 2016), or formulated within an evolutionary computation perspective such as Novelty Search (Lehman and Stanley, 2011a) or Quality Diversity (Cully et al., 2015; Cully and Demiris, 2017) (see next sections). In Forestier and Oudeyer (2016c), the implemented modular population-based version of IMGEP was called Model Babbling. Several variants of Model Babbling were evaluated: Random Model Babbling is a variant where the goal space is chosen randomly and goals are chosen randomly in the goal space and Active Model Babbling one where the goal space is chosen based on the learning progress to control each object. Both implementations are instances of IMGEP as the goal spaces are generated autonomously from the sensory spaces and no "expert knowledge" has been given to the algorithm.

In machine learning, the concept of curriculum learning (Bengio et al., 2009) has most often been used in the context of training neural networks to solve prediction problems. Many approaches have used hand-designed learning curriculum (Sutskever and Zaremba, 2014), but recently it was shown how learning progress could be used to automate intrinsically motivated curriculum learning in LSTMs (Graves et al., 2017). However, these approaches have not considered a curriculum learning of sets of reinforcement learning problems, which is central in the IMGEP framework formulated with goals as fitness functions, and assumed the pre-existence of a database with learning exemplars to sample from. In recent related work, (Matiisen et al., 2017; Portelas et al., 2019) studied how intrinsic rewards based on learning progress could also be used to automatically generate a learning curriculum with discrete and continuous sets of reinforcement learning problems, respectively. In contrast to the present study, both works did not consider high-dimensional modular problem spaces. See Portelas et al. (2020) for a review of other recent curriculum learning methods applied to reinforcement learning scenarios. The concept of "curriculum learning" has also been called "developmental trajectories" in prior work on computational modeling of intrinsically motivated exploration (Oudeyer et al., 2007), and in particular on the topic of intrinsically motivated goal exploration (Baranes and Oudeyer, 2013; Forestier and Oudeyer, 2017).

The concepts of goals and of learning across goals have been introduced in machine learning in Kaelbling (1993) with a finite set of goals. Continuous goals were used in Universal Value Function Approximators (Schaul et al., 2015), where a vector describing the goal is provided as input together with the state to the neural network of the policy and of the value function. However, in these works the goals are not modular, and are considered extrinsic to the agent, with extrinsic rewards that can contain expert knowledge about the tasks being learned. The learning problem is not formulated as an autonomous learning problem where the agent has to explore the most diverse set of states and skills on its own. Another work integrates intrinsic rewards with an extension of Universal Value Function Approximators (Colas et al., 2018a). This is a particular implementation of the IMGEP architecture, that we call GCP-IMGEP, using a unique monolithic (multi-task multi-goal) policy network, that learns from on a replay buffer filled with rollouts on task and goals of high learning progress. Also, using a population-based intrinsically motivated agent within the IMGEP architecture can help bootstrap a deep RL agent (Colas et al., 2018b). Filling the replay buffer of a deep RL agent with exploratory trajectories collected by an IMGEP algorithm kick-starts the RL agent by enhancing its exploratory abilities. It combines the efficient exploration of population-based IMGEP agents with the efficient fine tuning of policies offered by deep RL agents with a function approximator based on gradient descent.

5.1 IMGEP AND NOVELTY SEARCH

In Novelty Search evolutionary algorithms, no objective is given to the optimization process, which is driven by the novelty or diversity of the discovered individuals (Lehman and Stanley, 2011a). In this implementation, an archive of novel individuals is built and used to compute the novelty of the individuals of the current generation of the evolutionary algorithm. If the novelty of a new individual is above a threshold, it is added to the archive. Different measures of novelty can be used, a simple one being the average distance of the individual to its closest neighbors in the archive, the distance being measured in a behavioral space defined by the user. Then, to generate the population of the next generation, the individuals with a high measured novelty are reused, mutated or built upon.

Although designed in an evolutionary framework, the Novelty Search (NS) algorithm can be framed as a population-based IMGEP implementation, assuming that the behavioral space and its distance measure can be self-generated by the algorithm. Indeed, we can define an IMGEP goal space based on the NS behavioral space, with each behavior in that space generating the corresponding goal of reaching that behavior, with a fitness function defined as the negative distance between the target behavior and the reached behavior. In IMGEP, if the goal g (defining the target behavior) is chosen randomly, the algorithm can then reuse the previous reached behaviors that give the highest fitness to reach the current goal g, which are the closest reached points in the behavioral space. The key similarity between our population-based implementations of IMGEP and Novelty Search is that the previous behavior the closest to the current random target behavior is a behavior with high novelty on average. Indeed, a random point in a space is more often closer to a point at the frontier of the explored regions of that space which is thus a high-novelty point. Randomly exploring behaviors or mutating high-novelty behaviors are therefore efficient for the same reasons.

Abandoning the external objectives and focusing on the novelty of the behaviors in Lehman and Stanley (2011a) can be seen in the lens of the IMGEP framework as embracing all self-generated objectives.

5.2 IMGEP AND QUALITY DIVERSITY

The Novelty Search approach stems from the fact that in many complex optimization problems, using a fitness function to define a particular objective and relying only on the optimization of this function do not allow the discovery of the objective as unknown complex successive stepping-stones need to be reached before the final objective can be approached. Relying on novelty allows to reach stepping-stones and build upon them to explore new behaviors even if the objective does not get closer. However, when the behavioral space is high-dimensional, pursuing the final objective is still useful to drive exploration together with the search for novelty (Cuccu and Gomez, 2011). The Quality Diversity approach combines the search for Diversity from Novelty Search approaches and the use of an external objective function to ensure the Quality of the explored individuals (Cully and Demiris, 2017; Lehman and Stanley, 2011b; Mouret and Clune, 2015).

In the MAP-Elites algorithm (Mouret and Clune, 2015), the behavioral space is discretized into a grid of possible behaviors, and a fitness function is provided to assess the quality of individuals according to a global objective. Each new individual is assigned to a behavioral cell in this grid and is given a quality value with the quality function. The population of the next generation of the evolutionary algorithm is mutated, in its simplest version, from a random sample of the set of the best quality individual of all cells. In more sophisticated versions, the parents used for evolving the next generation are selected based on their quality, the novelty of the cells, or a tradeoff between quality and novelty.

In the applications of this algorithm, the fitness function is an extrinsic objective. For instance, in Cully et al. (2015) robot controllers are evolved to find efficient robot walking behaviors. The fitness function given to the algorithm is the speed of the robot, while the descriptors of a behavior can be the orientation, displacement, energy used, deviation from a straight line, joint angles, etc. The algorithm thus tries to find efficient walking behaviors for each behavioral set of constraints.

The concept of Quality Diversity algorithms is thus different from the concept of intrinsically motivated exploration, however Quality Diversity algorithms could be used with a fitness function that is intrinsically generated by the algorithm. In any case, the functioning of the algorithm given this fitness function can also be seen as a population-based implementation of the IMGEP framework. Indeed, each cell of the behavioral grid can generate one different IMGEP goal with a particular fitness function returning the quality of the individual if its behavior falls into that cell and zero otherwise. In MAP-Elites (Mouret and Clune, 2015), the next generation of individuals is mutated from a random sample of elites (the best quality individual of each non-void cell). In an IMGEP settings with those goals, the MAP-Elites sampling is equivalent to selecting a random goal from the set of goals that had a non-zero fitness in the past. When such a goal is selected, the new IMGEP exploration experiment then reuse, in its simplest version, the sample with the best fitness for that goal, which corresponds to the elite.

In the Novelty Search, Quality Diversity and IMGEP implementations, the key mechanisms that makes exploration efficient are 1) a diversity of solutions continue to be explored even if they seem non-optimal, and 2) when exploring solutions to a given region/cell/goal, the algorithm can find solutions to other regions/cells/goals, which are recorded and can be leveraged later.

5.3 IMGEP AND REINFORCEMENT LEARNING

In our setting, the fitness functions f_g have two particularities in comparison with the concept of "reward function" as often used in the RL literature. The first particularity is that these fitness functions are computed based on the trajectory τ resulting from the execution of policy Π, and thus consider the whole interaction of the agent and its environment during the execution of the policy, for instance taking into account the energy used by the agent or the trajectory of an object. Therefore they are not necessarily Markovian if one considers them from the perspective of the level of state transitions s_t.

The second particularity is that since the computation of the fitness $f_g(\tau)$ is internal to the agent, it can be computed any time after the experiment and for any goal $g \in \mathcal{G}$, not only the particular goal that the agent was trying to achieve. Consequently, if the agent stores the observation τ resulting from the exploration of a goal p_1, then when later on it self-generates goals $g_2, g_3, ..., g_i$ it can compute, without further actions in the environment, the associated fitness $f_{g_i}(\tau)$ and use this information to improve over these goals g_i. This property is essential as it enables direct reuse of data collected when trying to achieve a goal for later exploring other goals. It is leveraged for curriculum learning in Intrinsically Motivated Goal Exploration Processes.

6. DISCUSSION

This paper provides a formal framework for an algorithmic architecture called Intrinsically Motivated Goal Exploration Processes (IMGEP). This framework enables a unified description of various related algorithms that share several principles: exploration is driven by self-generated goals, exploring towards a goal gives information that can be reused to improve solutions for other goals, and intrinsic rewards can help the construction and selection of goals.

The IMGEP framework is both compact and general. The goals are defined through fitness functions and therefore can represent any kind of objective that can be computed from the information stored and available to the agent. The policies can be implemented by any algorithm that can learn a function that takes a goal as input and outputs actions to explore this goal, such as a monolithic neural network (Colas et al., 2018a) or a population-based policy (Forestier and Oudeyer, 2016c). We described a particular implementation of IMGEP that formalizes the Model Babbling algorithm described in Forestier and Oudeyer (2016c), through population-based policies with spatial modularity: the agent generates one goal space for each object in the environment, and temporal modularity: the temporal structure of objects' movement is leveraged for a more efficient goal exploration.

The IMGEP framework is well suited to environments where in order to avoid local optima and find advanced behaviors or phenotypes, enough time should be allocated to the continued exploration of non-optimal solutions. Interesting unexpected stepping-stones can be discovered in the exploration process and built upon afterwards. The framework is most useful when the stepping-stones or the targets are unknown to the expert user or too complex such that they can't easily be represented and optimized as a reward function. In that case the use of intrinsic motivations for the exploration of goals can help to discover and build upon a diverse set of stepping-stones. Furthermore, the use of intrinsic rewards, e.g. based on the monitoring of the learning progress in achieving goals, can further improve the efficiency of exploration by focusing on the most interesting problems and avoiding the ones that bring no more information.

We studied IMGEP implementations in different tool-use environments. We designed the first robotic experiment where an intrinsically-motivated humanoid robot discovers a complex continuous high-dimensional environment and succeeds to explore and learn from scratch that some objects can be used as tools to act on other objects. We also created the Minecraft Mountain Cart environment, where the first intrinsically motivated Minecraft agent discovered that a pickaxe can be used to mine blocks.

We evaluated different variants of Intrinsically Motivated Goal Exploration Processes and showed that only the variants where we do not hand-design a curriculum for learning (RMB and AMB conditions) allowed to discover the most complex affordances. Furthermore, when the agent monitors its learning progress with intrinsic rewards (AMB), it autonomously develops a learning sequence, or curriculum, from the easiest to the most complex tasks, and explore more efficiently than without those intrinsic rewards. Also, the comparison between agents only exploring one interesting problem space (SGS) versus all spaces (AMB) shows that if an engineer were to specify the target problems to solve (e.g. move the ball), then it would be more efficient to also explore all other possible intrinsic goals to develop new skills that can serve as stepping-stones to solve the target problem.

Compared to other approaches, our IMGEP implementation is sample-efficient, with a number of iterations of 20k in the real robotic setup, 40k in the Minecraft environment, and 100k in the 2D simulated one. Approaches such as Quality Diversity need 40M iterations for example for the learning of a hexapod's locomotion (Cully et al., 2015), and deep Reinforcement Learning agents also require millions of training steps, e.g. 2M steps in the Atari game Montezuma's Revenge (Kulkarni et al., 2016), or 50M in a Doom-like first-person shooter (Dosovitskiy and Koltun, 2016). The IMGEP implementation in its simplest form with a Nearest Neighbor look-up used as inverse models is also computationally efficient, as we have run the 20k iterations of the real robotic experiment on a raspberry Pi 3.

A current limitation of our setup is that we suppose agents already have a perceptual system allowing them to see and track objects, as well as spaces of representations to encode their transformations. Recent works studied in simulation the learning of a representation of objects from pixels and its use as a goal space for population-based intrinsically motivated agents (Péré et al., 2018; Laversanne-Finot et al., 2018). These first experiments inspired subsequent works on learning goal spaces from pixels for monolithic deep reinforcement learning agents in real-world robotic manipulation scenarios (Nair et al., 2018; Pong et al., 2019).

The IMGEP framework has also been applied to the exploration of very different setups in other scientific domains. In Grizou et al. (2019), an IMGEP implementation allowed to discover a variety of droplet behaviors in a chemical system of self-propelling oil droplets in water, where the exploration parameters were the concentrations of the different components of the oil droplet among others. In yet another domain, Reinke et al. (2019) showed that the IMGEP framework with a goal representation learned online could find self-organized patterns in the complex morphogenetic system Lenia, a continuous game-of-life cellular automaton.

Acknowledgments

We would like to thank Alexandre Péré, Cédric Colas, Adrien Laversanne-Finot and Olivier Sigaud for their valuable comments on an earlier version of this manuscript and Damien Caselli for his technical help.

References

Gianluca Baldassarre and Marco Mirolli. *Intrinsically Motivated Learning in Natural and Artificial Systems*. Springer, 2013.

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

Adrien Baranes and Pierre-Yves Oudeyer. Maturationally-constrained competence-based intrinsically motivated learning. In *2010 IEEE 9th International Conference on Development and Learning*, pages 197–203. IEEE, 2010b.

Adrien Baranes and Pierre-Yves Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1), 2013.

Andrew G Barto. Intrinsic motivation and reinforcement learning. In *Intrinsically motivated learning in natural and artificial systems*, pages 17–47. Springer, 2013.

Marina Bazhydai, Katherine Twomey, and Gert Westermann. Curiosity and exploration. In Janette B. Benson, editor, *Encyclopedia of Infant and Early Childhood Development (Second Edition)*, pages 370 – 378. Elsevier, Oxford, second edition edition, 2020. ISBN 978-0-12-816511-9. doi: https://doi.org/10.1016/B978-0-12-809324-5.05804-1. URL `http://www.sciencedirect.com/science/article/pii/B9780128093245058041`.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

Daniel E Berlyne. Curiosity and exploration. *Science*, 153(3731):25–33, 1966.

Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. Curious: Intrinsically motivated multi-task, multi-goal reinforcement learning. *arXiv preprint arXiv:1810.06284*, 2018a.

Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. GEP-PG: Decoupling exploration and exploitation in deep reinforcement learning algorithms. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1039–1048, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018b. PMLR. URL `http://proceedings.mlr.press/v80/colas18a.html`.

Giuseppe Cuccu and Faustino Gomez. When novelty is not enough. In *European Conference on the Applications of Evolutionary Computation*, pages 234–243. Springer, 2011.

Antoine Cully and Yiannis Demiris. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 22(2):245–259, 2017.

Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

Peter Dayan and Terrence J Sejnowski. Exploration bonuses and dual control. *Machine Learning*, 25 (1):5–22, 1996.

Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*, 2016.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

S Forestier and P-Y Oudeyer. Curiosity-driven development of tool use precursors: a computational model. In *Proceedings of the 38th Annual Meeting of the Cognitive Science Society*, 2016a.

S. Forestier and P.-Y. Oudeyer. Overlapping waves in tool use development: a curiosity-driven computational model. In *Sixth Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, 2016b.

S Forestier and P-Y Oudeyer. A unified model of speech and tool use early development. In *Proceedings of the 39th Annual Meeting of the Cognitive Science Society*, 2017.

Sébastien Forestier and Pierre-Yves Oudeyer. Modular active curiosity-driven discovery of tool use. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 3965–3972. IEEE, 2016c.

Alison Gopnik, Andrew N Meltzoff, and Patricia K Kuhl. *The scientist in the crib: Minds, brains, and how children learn.* William Morrow & Co, 1999.

Jacqueline Gottlieb and Pierre-Yves Oudeyer. Towards a neuroscience of active sampling and curiosity. *Nature Reviews Neuroscience*, 19(12):758–770, 2018.

Jacqueline Gottlieb, Pierre-Yves Oudeyer, Manuel Lopes, and Adrien Baranes. Information-seeking, curiosity, and attention: computational and neural mechanisms. *Trends in Cognitive Sciences*, 17 (11), November 2013. ISSN 13646613.

Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. *arXiv preprint arXiv:1704.03003*, 2017.

Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

Jonathan Grizou, Laurie J Points, Abhishek Sharma, and Leroy Cronin. Exploration of self-propelling droplets using a curiosity driven robotic assistant. *arXiv preprint arXiv:1904.12635*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117, 2016.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, pages 1094–1099. Citeseer, 1993.

Frédéric Kaplan and Pierre-Yves Oudeyer. Maximizing learning progress: an internal reward system for development. In *Embodied artificial intelligence*, pages 259–270. Springer, 2004.

Celeste Kidd and Benjamin Y Hayden. The psychology and neuroscience of curiosity. *Neuron*, 88(3): 449–460, 2015.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 3675–3683, 2016.

Dharshan Kumaran, Demis Hassabis, and James L McClelland. What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in cognitive sciences*, 20(7): 512–534, 2016.

Andras Kupcsik, Marc Peter Deisenroth, Jan Peters, Ai Poh Loh, Prahlad Vadakkepat, and Gerhard Neumann. Model-based contextual policy search for data-efficient generalization of robot skills. *Artificial Intelligence*, 247:415–439, 2017.

Matthieu Lapeyre, Pierre Rouanet, Jonathan Grizou, Steve Nguyen, Fabien Depraetre, Alexandre Le Falher, and Pierre-Yves Oudeyer. Poppy Project: Open-Source Fabrication of 3D Printed Humanoid Robot for Science, Education and Art. In *Digital Intelligence 2014*, Nantes, France, September 2014.

Adrien Laversanne-Finot, Alexandre Pere, and Pierre-Yves Oudeyer. Curiosity driven exploration of learned disentangled goal spaces. In Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto, editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 487–504. PMLR, 29–31 Oct 2018. URL `http://proceedings.mlr.press/v87/laversanne-finot18a.html`.

Joel Lehman and Kenneth O Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011a.

Joel Lehman and Kenneth O Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 211–218. ACM, 2011b.

Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *arXiv preprint arXiv:1707.00183*, 2017.

Clément Moulin-Frier, Sao M Nguyen, and Pierre-Yves Oudeyer. Self-organization of early vocal development in infants and machines: the role of intrinsic motivation. *Frontiers in psychology*, 4, 2013.

Clément Moulin-Frier, Sao M. Nguyen, and Pierre-Yves Oudeyer. Self-organization of early vocal development in infants and machines: the role of intrinsic motivation. *Frontiers in Psychology*, 4, 2014. ISSN 1664-1078.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 9191–9200. Curran Associates, Inc., 2018. URL `http://papers.nips.cc/paper/8132-visual-reinforcement-learning-with-imagined-goals.pdf`.

P-Y. Oudeyer and F. Kaplan. What is intrinsic motivation? A typology of computational approaches. *Frontiers in Neurorobotics*, 1, 2007. ISSN 1662-5218.

P-Y Oudeyer, Jacqueline Gottlieb, and Manuel Lopes. Intrinsic motivation, curiosity, and learning: Theory and applications in educational technologies. *Progress in brain research*, 229:257–284, 2016.

Pierre-Yves Oudeyer and Linda B Smith. How evolution may work through curiosity-driven developmental process. *Topics in Cognitive Science*, 8(2):492–502, 2016.

Pierre-Yves Oudeyer, Frdric Kaplan, and Verena V. Hafner. Intrinsic Motivation Systems for Autonomous Mental Development. *IEEE Transactions on Evolutionary Computation*, 11(2), April 2007. ISSN 1089-778X.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. *arXiv preprint arXiv:1705.05363*, 2017.

Vitchyr H. Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. *CoRR*, abs/1903.03698, 2019. URL `http://arxiv.org/abs/1903.03698`.

Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. *3rd Conference on Robot Learning (CoRL)*, 2019.

Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. Automatic curriculum learning for deep rl: A short survey. 2020.

Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016.

Alexandre Péré, Sébastien Forestier, Olivier Sigaud, and Pierre-Yves Oudeyer. Unsupervised learning of goal spaces for intrinsically motivated goal exploration. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=S1DWPP1A-`.

Jeffrey Frederic Queißer, René Felix Reinhart, and Jochen Jakob Steil. Incremental bootstrapping of parameterized motor skills. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 223–229. IEEE, 2016.

Chris Reinke, Mayalen Etcheverry, and Pierre-Yves Oudeyer. Intrinsically motivated exploration for automated discovery of patterns in morphogenetic systems. *arXiv preprint arXiv:1908.06663*, 2019.

Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing-solving sparse reward tasks from scratch. *arXiv preprint arXiv:1802.10567*, 2018.

M. Rolf, J.J. Steil, and M. Gienger. Goal babbling permits direct learning of inverse kinematics. *IEEE Transactions on Autonomous Mental Development*, 2(3), 2010.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, pages 1312–1320, 2015.

J Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *From animals to animats: Proceedings of the first international conference on simulation of adaptive behavior*, pages 15–21, 1991.

Rupesh Kumar Srivastava, Bas R Steunebrink, and Jürgen Schmidhuber. First experiments with powerplay. *Neural Networks*, 41:130–136, 2013.

I. Sutskever and W. Zaremba. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.

Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference on machine learning*, pages 216–224, 1990.

Esther Thelen and Linda B Smith. *A dynamic systems approach to the development of cognition and action*. MIT press, 1996.