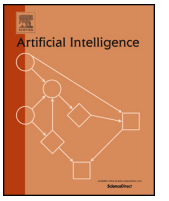
Artificial Intelligence 337 (2024) 104228

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Integration of memory systems supporting non-symbolic representations in an architecture for lifelong development of artificial agents

François Suro *, Fabien Michel, Tiberiu Stratulat

LIRMM - Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, Université de Montpellier, 161 Rue Ada, 34090, Montpellier, France

ARTICLE INFO

Dataset link: https://gricad-gitlab.univ-grenoble-alpes.fr/surof/evoagents2020

Dataset link: https://gite.lirmm.fr/suro/evoagents2020

Keywords:
Lifelong learning
Learning agents
Artificial behaviour
Modular architecture
Memory system
Internal representation

ABSTRACT

Compared to autonomous agent learning, lifelong agent learning tackles the additional challenge of accumulating skills in a way favourable to long term development. What an agent learns at a given moment can be an element for the future creation of behaviours of greater complexity, whose purpose cannot be anticipated.

Beyond its initial low-level sensorimotor development phase, the agent is expected to acquire, in the same manner as skills, values and goals which support the development of complex behaviours beyond the reactive level. To do so, it must have a way to represent and memorize such information.

In this article, we identify the properties suitable for a representation system supporting the lifelong development of agents through a review of a wide range of memory systems and related literature. Following this analysis, our second contribution is the proposition and implementation of such a representation system in MIND, a modular architecture for lifelong development. The new *variable module* acts as a simple memory system which is strongly integrated to the hierarchies of skill modules of MIND, and allows for the progressive structuration of behaviour around persistent non-symbolic representations. *Variable modules* have many applications for the development and structuration of complex behaviours, but also offer designers and operators explicit models of values and goals facilitating human interaction, control and explainability.

We show through experiments two possible uses of *variable modules*. In the first experiment, skills exchange information by using a variable representing the concept of "target", which allows the generalization of navigation behaviours. In the second experiment, we show how a non-symbolic representation can be learned and memorized to develop beyond simple reactive behaviour, and keep track of the steps of a process whose state cannot be inferred by observing the environment.

* Corresponding author.
E-mail addresses: surof@univ-grenoble-alpes.fr (F. Suro), fmichel@lirmm.fr (F. Michel), stratulat@lirmm.fr (T. Stratulat).

https://doi.org/10.1016/j.artint.2024.104228
Received 19 July 2022; Received in revised form 31 July 2023; Accepted 9 September 2024

Available online 12 September 2024

1. Introduction

The field of Machine Learning has provided many solutions applicable to the training of intelligent agents, suited to different learning contexts, levels of supervision and availability of data. Some works have focused on multitask learning, and how previous learning experience can be leveraged when the goals and problems faced by the agents change. Transfer Learning [1,2], for instance, provides solutions to retrain an agent for a new target behaviour, and Protean Learning [3] for a change in its domain and signature. These solutions aim at solving the problem of change and help the agent adapt to new constraints. However, they are not designed to integrate change as part of the learning process, with long term and open-ended development in mind.

In contrast to other fields focusing on learning, Developmental Robotics seeks novel methodologies from studies in the biological and psychological development of natural systems to create flexible artificial systems able to develop skills suited to real world problems and go beyond single-task learning. "The search for flexible autonomous and open-ended multitask learning system is in essence, a particular re-instantiation of the long-standing research for general-purpose AI" [4]. Developmental Robotics supports the claim that embodiment is required for the emergence of intelligence as we know it: biological systems are not passively exposed to sensory input, but instead interact actively with their surrounding environment [4] leading to self-organization of dynamical interactions among brain, body and environment [5].

The benefits of embodiment can be seen directly in the simplification of a number of traditional AI problems, such as image processing which can take advantage of depth by actively changing point of view. It supports development in the formation of the notion of object which emerges from the simultaneous experiences of seeing (spatial bounds) and grasping (impenetrability). From this notion can begin the categorization for practical use (food and non-food) which in turn serves as a basis for the emergence of symbols [4].

Ongoing emergence [6] proposes criteria for a complete developmental approach: the bootstrapping, the continuous acquisition and incorporation of new skills, their stability and reproducibility, and the autonomous development of values and goals. In these criteria, developmental robotics meets the field of lifelong (machine) learning, specifically Lifelong Robot Learning [7] which aims at providing artificial agents with the ability to accumulate skills without compromising or forgetting previous ones (stability), in an incremental manner that will favour the acquisition of new skills of greater complexity.

Along with the ability to progressively learn and structure its behaviour, an agent aiming at lifelong development will require the ability to form internal representations to evolve beyond purely reactive behaviour. Internal representations play a role at many levels of the cognitive process, from *fragile* and *working* memory keeping short-term information required for immediate spatio-temporal problems, to long term memory of experiences and acquired symbols needed in abstract reasoning as we understand it. Memory is required to commit to a task and exhibit a behaviour that is not entirely conditioned by immediate sensory information. To develop its own values and goals, an agent must be able to maintain an internal state, existing apart from the environment and other agents. Such internal states grant agents individuality and enable independent behaviour even in a completely homogeneous population. This ability plays an important part in social behaviour, allowing the representation of roles able to change dynamically [8,9].

In this article we address memory systems and representations in the context of Lifelong Development for agents. We intend to approach memory systems along the same principles of lifelong development of procedural knowledge: complex representations in both their form and meaning are reached by the progressive increase in complexity of previously acquired representations, beginning with the non symbolic. Closely coupled with procedural knowledge and its development mechanism, non-symbolic internal representations are flexible enough to fit the particular needs of agents, follow their development path and increase their complexity along with it. The use of this memory system, the representations stored in it and the interpretation of these representations, will have to be learned according to behaviours. The validity of the representations acquired by the agent depends solely on their ability to support the agent's actions in the world. This emergent process is the first step towards developing "grounded" symbols [10,5], which are later introduced and negotiated in a social context [10].

We propose an implementation of this approach as the *Variable module* on MIND, an architecture designed for lifelong agent development which serves as our testbed. The MIND architecture [11] is based on a flexible modular system able to integrate new skills and progressively structure complex behaviours through a curriculum of training tasks. The new *Variable module* integrates closely to skills and their learning process, making it suitable for the acquisition of emergent representations. We show how it can be used to increase the complexity of behaviour, and develop beyond the reactive level. We explore new options in the structuration of behaviour offered by the ability to store, retrieve and share information, such as in-line or branching structures organized around a variable module representing a particular concept. Furthermore, we discuss how internal representations impact learning strategies and agent development, and offer opportunities to introduce human guidance and control, improve explainability of behaviour and serve as a basis for motivational systems.

After reviewing the domain, motivating and defining our approach to memory systems in Section 3, we present MIND and its new *Variable module* in Section 4, along with its implementations and mechanism to enable human guidance and control, the *Drive module*. Section 5 presents the experimental context and training process, and two experiments evaluating our new system: we first explore an alternative structuration of a previously established skill hierarchy, using a variable to exchange information between learning skills (Section 5.2), and then we provide a variable to a learning skill to memorize a step in a sequence of actions (Section 5.3).

2. Related works

Our work is related to the field of developmental robotics [4,5] where representation of information is studied under different aspects, such as the acquisition of symbols [10,12], their grounding [13], and their obvious use in the decision process, or the part

such information can play in motivational systems [14,15]. We do not deal with any of these particular mechanisms beyond what is needed for our experiments, relying for now on a fixed curriculum to train behaviour, leaving the implementation of autonomous development [16–18] for future works. This article focuses on the structure and elements able to support representation of information in a developmental context.

Our general approach to structuration using MIND is close to the works on evolving virtual creatures [19] for its hierarchical aspect and combination mechanism, differing mostly by our encapsulation of heterogeneous sub-structures. The specific focus of this article on the place of representation of information in such structures bears resemblance to several connectionist approaches, from autoencoders [20] to layered learning [21] and modular neurocontrollers [22], forming what can be considered as emergent sub-symbolic representations into specific neuron layers. The information processed into these representations can be provided as input for different parameterized skills [23]. Unlike these works however, the integration of elements for representation of information is done following a "behaviour shaping" approach [24] at the level of the architecture, with a higher level purpose aiming at establishing building blocks for the lifelong development of agents [7], and their possible use in future works on meta behaviours such as intrinsic motivation [14,25].

The experimental process we use is similar to experiments on embodied developmental models associating abstract representation and motor skills [26,27]. However, our goal is to acquire practical representations of information in a lifelong development context, rather than simulate specific psychological models. Our results show the formation of sub-symbolic *prototypes* [10] grounded in the agent's experience. This process is comparable to the concept quantization [28] used in neuro-symbolic concept learning [12], although in our embodied approach, physical interaction is substituted to language feedback, and the program execution is validated by its actions in the environment and progress towards a goal. Unlike the ambitious works of Konidaris on symbol generation [13], our approach covers the acquisition of skills along with the formation of representations.

3. Memory and representation for lifelong development of artificial agents

The development of an agent as an individual distinct from others involves the persistence of knowledge, outside the restrictions of the environment (boundedness, ephemerality). Skills are a form of persistent knowledge, and many works studied the acquisition and structuration of such procedural knowledge (*'know-how'* or *'savoir-faire'* knowledge). Skills are usually long term knowledge and do not change on a short term scale.

Skills and procedural knowledge are often distinguished from declarative memory and propositional knowledge which represent information on knowing *'that'*, on storing facts and events. From the perspective of cognitive psychology, declarative knowledge refers to long-term symbolic memory, which can be retrieved and spoken about explicitly. However, this definition has a few exceptions such as the memory of faces, which is not symbolic and explicit. The evolutionary perspective views declarative memory, which occurs only in higher animals, as more recent development compared to procedural memory. As such, it might be considered as an outgrowth of procedural memory, which argues for a less clear-cut and more gradual distinction between them [29].

In this section we look at declarative knowledge in artificial systems in a broad sense, that is information about the current context. This will include short term memorization of states to processed information representing concepts which can be structured or exchanged between agents. Section 3.1 introduces low-level connectionist approaches to this issue which are used in many fields because of their general purpose and non-symbolic nature. Section 3.2 presents the top-down designs found in cognitive architectures created specifically to serve the needs of embodied artificial agents. Section 3.3 discusses the acquisition of representations and their relation to meaning. Section 3.4 gives a synthesis on the use of internal states and representation with regard to the lifelong development of agents.

3.1. Neuro-inspired low-level memory

Bio-inspired approaches to Machine Learning and connectionist AI techniques have investigated memory systems and the bottom-up formation of low to intermediate levels of non-symbolic representations for a wide range of applications [30,31]. Early developmental agent architectures such as robot shaping [24] include a memory of the past state of the agent's sensors to deal with time series (following remarks from Whitehead and Lin [32]). The authors note that this kind of memory *need not be regarded as a "representation" of anything*. This improvement upon purely reactive agent behaviour involves memory as understood in common language: recording past states of the environment.

This aspect of short term memory is akin to the idea of *iconic memory* or *fragile memory* in the human brain. Both are short-lived high-capacity memories used in storing a much larger raw perceptual information than can be immediately processed. Applications are obvious in spatio-temporal problems such as extrapolating the future position of a projectile based on previously recorded positions. Recent studies [33] indicate *fragile memory* has some level of processing, although not on the same level as *working memory*. The low-level processing of *fragile memory* could be compared to context units [34] in Recurrent Neural Networks (RNN), or *reservoir computing* [35] methods such as *echo states networks* [36] or *liquid state machines* [37]. In reservoir computing [38] instantaneous input is fed to the reservoir network which accumulates and enriches the state space. Readouts are done by another network feeding from the reservoir and trained by conventional methods (Fig. 1).

In contrast to *iconic memory* and *fragile memory*, *working memory* is a (comparatively) long-lived low capacity memory used to retain processed perceptions that can be consciously addressed. *Working memory* can be understood as storing high-level elements for short term planning, for instance the spatial coordinates of an entity in the context of a navigation problem. An overview of

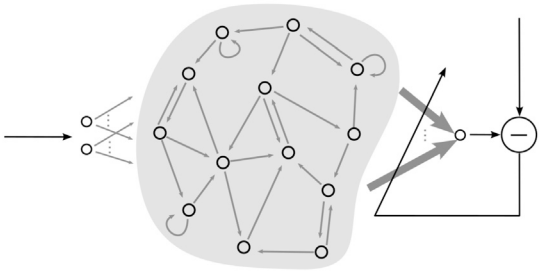


Fig. 1. Reservoir computing: the instantaneous input on the left is fed to the reservoir network (in grey). On the right, the readout is done by another network (from Lukoševičius and Jaeger [38]).

these distinct short term memories in the human brain and an examination of their respective properties is given in Vandenbroucke et al. [33].

The auto-encoder (or auto-associator) [20] is a solution to processing data into compact intermediate level representations. Auto-encoders are neural network structures in which an encoding layer, processing a large number of input neurons, is linked to a decoding layer, mirroring the input neurons, through a smaller hidden layer. The auto-encoder is trained to map the input onto itself, using the error measured between the input and the encoded-decoded output. Since the hidden layer is much smaller than the input, this process creates a compact representation in the hidden layer which only retains the information useful to recreate an acceptable approximation of the input.

The idea of using an intermediate information element containing processed representation as a junction between two processes (here the encoder and decoder) has inspired several approaches to the structuration of procedural knowledge such as layered learning [21] or modular neural network policies [22]. In these works, separate levels or layers of control are linked by a memory element used to share an intermediate representation. In layered learning output neurons of a subnetwork are connected to the input neurons of the following network. In modular neural network policies different sensor-dependent and actuator-dependant networks can be interchanged by training a common middle layer.

Low to intermediate levels of representations stored in persistent memory modules, as we have seen, have many uses for autonomous agents in general, from learning time dependant behaviours to improving structuration, but they also are of particular interest for developmental agents. Such memory elements can represent simulated physiological or psychological internal states used to trigger and regulate autonomous behaviour. Motivational systems inspired by *drive reduction theory* [39] use sub-symbolic internal representations to that effect for autonomous agent development [14].

3.2. Memory in cognitive architectures

Long term memory systems are found in cognitive architectures for both procedural and declarative knowledge, association and past experiences. Current elements of working memory or perception are mapped to the appropriate long term memory structure to gain higher level knowledge or prediction.

The belief hierarchy of the ICARUS architecture [40] is a mechanism for long term memory of percept associations: percepts are combined to form low-level beliefs, which in turn can be combined to form higher level beliefs. In the example of self-driving car given in Fig. 2, the agent uses primitive beliefs (*right-of*) to determine the relative position of 3 lines on the road. From these beliefs

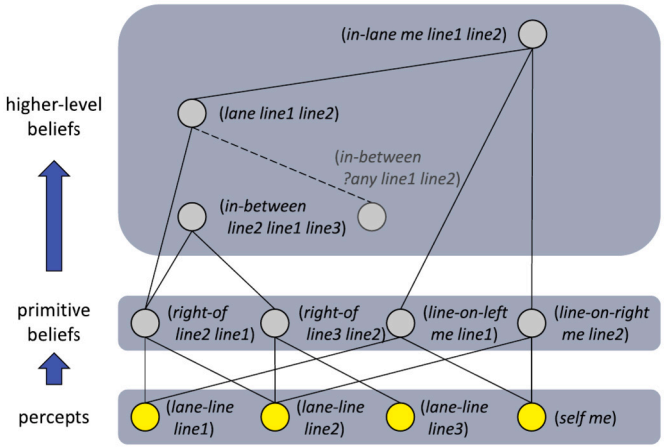


Fig. 2. An example of the ICARUS belief system. The agent determines it is in the lane 1-2 from the perception of the relative position of 3 lines and its own position (from Choi and Langley [40]).

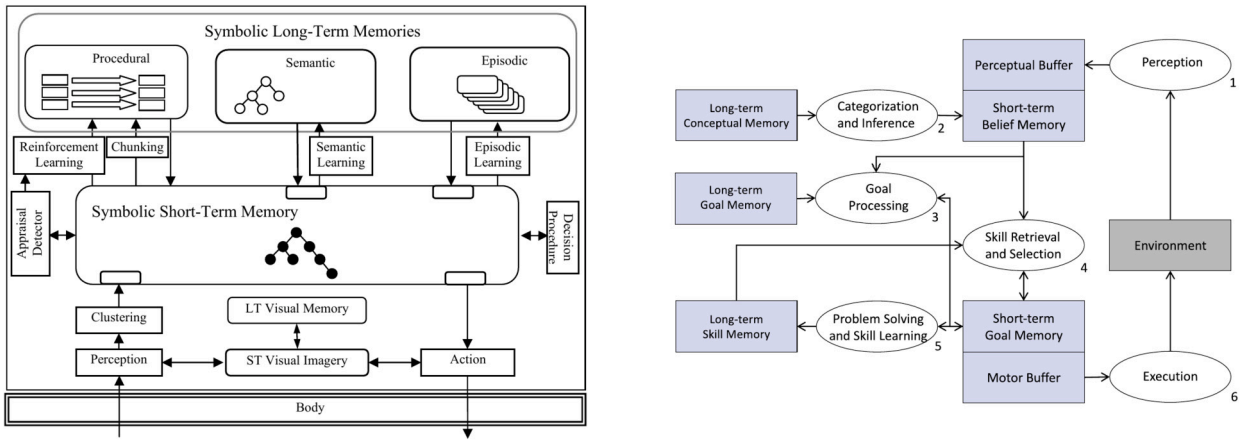


Fig. 3. On the left: the extended SOAR architecture, showing a flat mapping between long term and short term memory (from Laird [43]). On the right: the ICARUS architecture, mapping long term to short term memory of different components at different stages of the deliberative process (from Choi and Langley [40]).

it can reach the higher level knowledge of the existence of a lane between line 1 and 2, and its current position within that lane. The semantic structure associating beliefs is the long term memory element and reflects a knowledge about the world that should hold true. Although explicit, this belief hierarchy shares some similarities with reservoir structures where nodes of any level can be accessed by the control hierarchy. For ICARUS, the control hierarchy is a teleoreactive hierarchy of subgoals (see [41,42]), where each subgoal uses the appropriate beliefs, and level of beliefs, for its decision process. For instance, a high-level goal of overtaking a vehicle might use lane positions and occupation beliefs whereas a simple subgoal of keeping inside a lane will use low-level beliefs about line positions.

In addition to long term semantic memory, the extended SOAR architecture [43] includes other kinds of long term memory systems, such as episodic memory. Episodic memory is designed to store sequences of states, perceptions and working memory elements, that were experienced by the agent. When the agent experiences a succession of states matching a partial sequence in the Episodic memory, the rest of the sequence can be recalled and considered to make a prediction.

Experiments were conducted [44] in which an agent, among other tasks, must collect energy charges as its battery discharges. While performing other tasks, the agent might discover an energy charge which it does not need at the present moment, it will however record the successive steps/state preceding its discovery. When the agent is in need of energy and recognizes several steps of a memorized sequence in its current context, it will enact the following steps of the episode to reach its goal. The results show that this approach is ten times more efficient than a random search. Although the example given is a navigation problem, the encoding of the episodic memory is claimed to be task-independent (as long as there remains a temporal/sequential relationship).

Having a representation of knowledge strongly dependent on the task and type of knowledge is a common practice in many projects with commercial or industrial applications. For instance, self-driving cars may use dedicated sensors such as LiDAR to accurately map their environment. The clouds of 3D points are interpreted as simple volumes, bounding boxes and planes which can be stored in memory. This model of the world is not only convenient for humans to understand, but the agent can also be given algorithms specific to this representation, such as path planning algorithms. It also allows for simulations and projections that can guide the decision process.

The top-down approaches for memory and representation used in cognitive architectures and other autonomous agent control systems are very efficient at solving the problems they were designed for and offer many advantages. Conceptual predicates can be used to generate higher level information or predictions, and specific algorithms compatible with the representation of information can be used to generate optimal solutions. However, they are generally task-dependant on some level, or make use of pre-specified representations and rules, or innate conceptual predicates [45]. Because such elements must be specified *a priori*, their application tends to be limited to problems for which a solution is already known and a context well-defined, they thus can not be solely relied upon for general purpose use, especially in the context of lifelong development.

3.3. Perception, categorisation, meaning, symbols

Previous sections show that the acquisition of representations involves filtering relevant information from perceptions or states. This process is comparable to feature extraction, categorization, encoding or even compression in a way that what is perceived is reduced to what is needed to understand the situation [46]. Increasing the level of representation generalizes what is observed, details are removed and the representation becomes more compact, as with the different levels of short term memories in the human brain [33], or the different levels of the belief hierarchy in the ICARUS architecture [40]. Such representations are said to be perceptually grounded, their meaning came from the interaction between the agent and its environment.

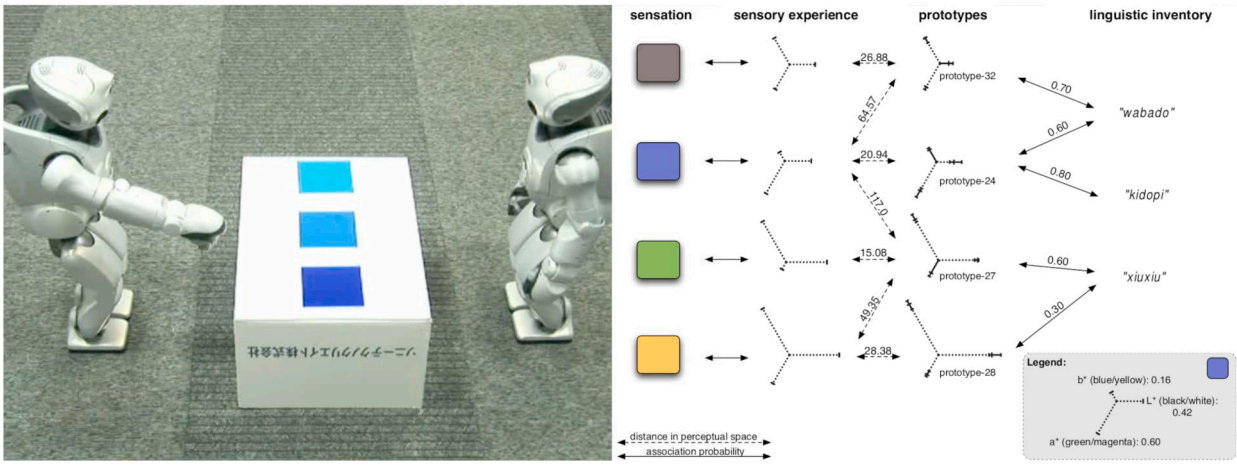


Fig. 4. On the left: Two robots playing a language game of naming and pointing at colours. On the right: portion of a semiotic network for a single agent, linking sensations to sensory experiences, prototypes, and symbols. The generated symbols are negotiated between agents during the language game (from Steels [10]).

Through this process, an agent generates abstract representations which can be used in high-level planning, for instance by matching them with conditions for execution, expected outcomes and measures of success of a plan. Konidaris et al. [13] shows how these representations can be autonomously acquired and linked to conditions and effects of motor skills, and in turn how they can be used in high-level planning.

With increased processing of the information, the selection of a particular domain to consider and the use of classification methods, abstract representations move from non-symbolic to symbolic. While the meaning remains grounded in individual experience, its association to a symbol is negotiated during social interactions. The cost and constraints of communication justify the use of highly abstracted, highly compressed symbolic representations (indeed, attempts at conveying a complex feeling or experience will find one "at a loss for words").

Recent works on neuro-symbolic systems [47,12] attempt to bridge the gap between low-level neural architectures and natural language. Questions given in natural language are converted in a tree structure of combination operators from a given domain specific language (*e.g.* filter, relate, exist). A neural-based vision system is trained to recognize concepts fitting the operators, such as entities and attributes (*e.g.* red, cube, dog) and spatial relations (*e.g.* right of, behind). In this work, the formation of concepts is guided by the language and the set of questions provided by the instructor.

The question of grounded symbols and emergent vocabularies has been largely covered in the works of Steels [10]. Through language games, agents develop a common language (semiotic landscape) enabling them to refer to elements of the environment such as colours [48] or relative spatial coordinates [49]. Fig. 4 shows a language game where agents take turns naming a colour[1], while the other agent must guess and point at the colour referred to. Each agent has developed its own representations (semiotic network) from sensations to prototypes, and generated its own linguistic inventory. Through the game, they negotiate the association between symbols and meaning, success in shared communication causes words to spread in the population [10]. In this work, the social context for learning is a peer relation between agents sharing the same environmental constraints. Unlike trainer-trainee interactions of other works (*e.g.* [12]), this approach is less susceptible to introduce the trainer's bias and favours the emergence of concepts with a level of precision suited to their common goals. The extensive sea ice terminology found in dialects of the Inupiaq language [50] (Alaska) illustrates the importance of this socio-environmental relation in the emergence of a culture and the formation of its relevant concepts.

In the example shown in Fig. 4, it is interesting to see the increased "compression" and the loss of detail that occurs when processing sensations into symbols. Categorization already loses the specific hue of the colour when converting sensations into prototypes (it is "blue"), but this also occurs when associating representations to symbols: here the word "xiuxiu" designates both the green and yellow prototypes. In this instance, one symbol was deemed sufficient to express the meaning (which is most likely "the other colours") needed to succeed in the guessing game.

Perceptually grounded approaches to the acquisition of representations are particularly suited to the development of memory systems and internal states for agents: each successive level of complexity serves a corresponding level of procedural knowledge, meaning that both can develop simultaneously and support each other. With such a mechanism supporting development, social behaviour and communication (at least with simple symbols) can be apprehended as an increment in complexity rather than a radical change. Abstract ideas and symbols which cannot be grounded are left for further steps in this development process.

[1] For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.

3.4. Internal states for lifelong development of agents

It is well known that intelligent behaviour can emerge without internal representations [51]. However, agents aiming at lifelong development require a way to form internal representations, if they are to reach levels of complexity approaching higher animals. In the previous sections, we reviewed a number of approaches to representation of declarative knowledge, from low-level to complex, non-symbolic to symbolic, from the formation of concepts to the emergence of language. We now provide an analysis of these approaches from the perspective of lifelong agent development. The requirements for the general purpose representation system for lifelong development of artificial agents we derive from this analysis is presented in Section 4.2.

On the integration of the representation system Should the skills of the agent use structures such as Recurrent Neural Networks to represent procedural knowledge, some form of low-level memory system will already be included in the form of the memory layer. However, these memory elements are tied to a single skill and cannot be shared with others, which precludes the representations acquired during training from supporting the development of subsequent skills. To address these limitations, an intermediate level memory system should also be included at the level of the control structure. This would allow commitment to a task as a memory layer, and also offer alternatives in skill structuration by, for instance, generalizing some behaviour around representations of concepts. Representations existing independently of the procedural knowledge manipulating them enable their use in the developmental process, where they can serve as base elements in the subsequent creation of skills whose purpose was not anticipated.

On purpose-built representation systems A developmental agent could benefit from using purpose-built high-level representation mechanisms. For instance, using a dedicated spatial memory for mapping and path planning makes practical sense, as navigation in a spatial environment is a problem often related to embodiment. However, the main drawback of these representation systems is their dependency on the type of knowledge they represent. Building agents relying on a given set of specialized elements, whether it is memory systems (*e.g.* Fig. 3) or motor primitives for behaviours, is already making the assumption that we have anticipated and provided all that will be needed for their development. Even if such systems can be very efficient for autonomous agents, they cannot completely fill the role of a more general purpose mechanism for lifelong development.

On representation systems and development We have seen mechanisms for classifying and encoding high dimensional states or intermediate elements of the working memory into a highly compressed representation, and how these representations play a role in emergent symbol grounding. In the long run, the emergence of high-level symbols will open the way to general purpose artificial intelligence and high-level reasoning as understood by humans. We regard this bottom-up approach as the most suited to developmental agents, each level of complexity of behaviour forming its useful and grounded representation as a basis for the next until abstract level operations can be reached to process these representations.

The representation system for lifelong development we propose borrows properties from the different approaches previously discussed, and is designed to be integrated into a developmental agent architecture. The following sections present our contribution, the *Variable module*, and its implementation as an extension of MIND, a reactive architecture for developmental agents.

4. An architecture integrating memory systems for lifelong development

Our approach to memory systems follows the same principles as the lifelong development of procedural knowledge: complex representations are reached by the progressive increase in complexity of previously acquired representations, and are closely coupled with procedural knowledge, providing mutual support in accomplishing new tasks. Therefore, this memory system has to be strongly integrated with the structure supporting procedural knowledge, and such architecture must fit the properties required by the memory system.

We propose an implementation of this approach as the *Variable module* on MIND, an architecture designed for lifelong agent development which is based on a flexible modular system able to integrate new skills and progressively structure complex behaviours through a curriculum of training tasks. As a regular component of the architecture, the *Variable module* is able to integrate closely to skills and their learning process, making it suitable for the progressive acquisition of emergent representations.

Section 4.1 presents MIND and its specific coordination mechanism, the *Influence*. Section 4.2 presents the *Variable module*, our general purpose memory system compliant with the *Influence* mechanism. Section 4.3 details several of its possible implementations. Section 4.4 presents the *drive module*, an implementation solution providing an entry point into MIND hierarchies to enable human guidance and control.

4.1. MIND, an architecture for lifelong development

The MIND architecture (Modular Influence Network Development) [11] is an artificial agent control architecture suited to open-ended and cumulative learning. MIND encapsulates heterogeneous structures representing procedural knowledge (such as neural networks or programmed procedures) into skill modules dedicated to performing different sub behaviours. Skill modules are combined into hierarchies, from the low-level *base skills* performing simple sensorimotor behaviour, to *complex* and *master skills* performing the combination of concurrent lower level skills.

Compared to similar research, the main original aspect of MIND lies in its multi-layered hierarchy [52,19] using a generic control signal, the *Influence*, allowing the use of vector composition [53–55] to obtain an efficient global behaviour. It offers significant

improvements over methods such as the Robot Shaping techniques [24] and other hierarchical approaches [25] by allowing high-level controllers to interpolate between the subspaces of low-level policies. The *Influence* mechanism proves to be suited to direct neurocontrol [19,56,57,22] as well as motor primitives or schemas (policies) [53]. Unlike cognitive architectures [43,40,25] using multiple sub-systems, the network of modules constructed by MIND should be viewed as development of reactive architectures towards complex behaviours. The modular design and the encapsulation of the skill internal function makes the choice of the controller independent of the implementation of MIND, and allows the use of different controllers for each skill, working together within the same hierarchy, which is an advantage over comparable systems [58,19].

4.1.1. Base skill, complex skill, and Influence

We consider sensory information and motor commands represented as vectors of real numbers, normalized between 0 and 1. It is possible to create a module encapsulating a vector-valued function f that maps the input vector $V_I = [I_1, I_2, ..., I_n]$ to the output vector $V_O = [O_1, O_2, ..., O_m]$ (Eq. 1). The function f can be implemented as a programming procedure, or it can be a function approximator such as a neural network, or any other kind of function that associates two vectors of real numbers. Such a module is called a *skill*, and the module whose output vector is used directly as motor commands is a *base skill*.

Several concurrent *base skills* can be trained to perform the subtasks of a complex task. Each *base skill* only associates the inputs and outputs necessary to accomplish its assigned task. To perform the complex task, a *complex skill* is created which will coordinate several *skills* (its *subskills*) by means of a signal called *Influence* which determines how much "weight" a *subskill* has on the overall behaviour. This approach can be understood as delegating to one or a combination of *subskills* the resolution of the current task in the same fashion the *Boid brain* coordinates its sub behaviours to perform the flocking behaviour [59].

A *Complex skill*, as any other skill, encapsulates a function as defined in Equation 1). Its output vector is directed at its subskills and called the *Influence* vector $V_{Infl} = [Infl_1, Infl_2, ..., Infl_m]$. A complex skill can be the subskill of a higher level complex skill, thus creating hierarchies of skills. The top level complex skill, the *master skill*, receive a constant *Influence* of 1.0, an impulse setting the whole decision process in motion.

Fig. 5 shows a Hierarchy of skills. The *Influence* flows along a vertical axis from the master skill down to the base skills to determine *who* (and with which magnitude) is in charge of the overall behaviour. The information from the sensors reaches all the skills of the hierarchy, and the motor commands are output from the base skills to the actuators forming a horizontal information flow. Its purpose is to determine *how* the behaviour is executed. Unlike other approaches (*e.g.* [24,60]), sensory inputs are available to every skill, including complex skills. This enables a complex skill to perform subtle coordination based on information that may not be needed by subskills.

4.1.2. Using Influence to determine motor commands

Starting from the master skill, each complex skill computes its output vector V_O and multiplies each element by the sum of the *Influences* it received, forming the *Influence* vector V_{Infl}. The skill then sends each element $Infl$ of the *Influence* vector to the corresponding subskill (Fig. 6).

The base skill computes its output vector and multiplies each element by the sum of the *Influences* it received, similarly to equation 2, forming the motor command vector $V_{Com} = [Com_1, Com_2, ..., Com_m]$. The base skill then sends each element Com_x of the motor command vector to the corresponding motor module along with the sum of the *Influences* ($\Sigma Infl$) the base skill received. Each motor

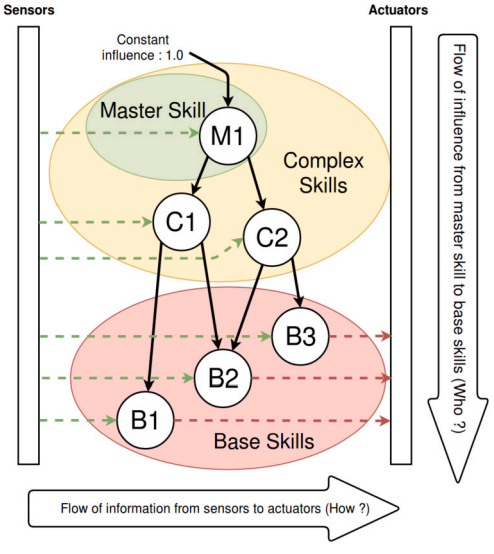


Fig. 5. A skill hierarchy, a *master skill* influences *complex skills* which in turn influence the *base skills*.

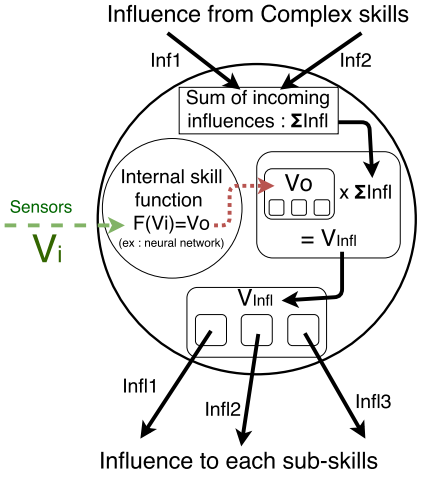


Fig. 6. Internal architecture of a skill.

$$V_O = f(V_I) \tag{1}$$

Eq. 1: The input vector V_I is supplied to the internal function f of the skill to produce the output vector V_O.

$$V_{Infl} = V_O * \sum_{c=1}^{Cs\mapsto s} Infl_c \tag{2}$$

Eq. 2: With V_{Infl} the *Influence* vector to the subskills, $V_O = f(V_I)$ the output vector of the internal function of the skill, and $\sum_{c=1}^{Cs\mapsto s} Infl_c$ the sum of all *Influences* the skill received (also noted $\Sigma Infl$).

$$Mo = \frac{\sum_{b=1}^{Bs\mapsto Mo} Com_b}{\sum_{b=1}^{Bs\mapsto Mo} \Sigma Infl_b} \tag{3}$$

Eq. 3: With Mo the resulting final motor command, b the index of the base skill that is sending a motor command, Com_b the weighted motor command for this motor module from the base skill b, $\Sigma Infl_b$ the sum of *Influences* from the base skill b.

$$Mo = \frac{\begin{array}{c}\sum_{b=1}^{Bs\mapsto Mo}(F_b(Vi_b)\mapsto_{Mo} * \\ \sum_{c=1}^{Cs\mapsto b}(F_c(Vi_c)\mapsto_b * \\ F_{Ms}(Vi_{Ms})\mapsto_c * 1.0))\end{array}}{\begin{array}{c}\sum_{c=1}^{Cs\mapsto b}(F_c(Vi_c)\mapsto_b * \\ F_{Ms}(Vi_{Ms})\mapsto_c * 1.0)\end{array}} \tag{4}$$

Eq. 4: With Mo the resulting final motor command, $Bs\mapsto_{Mo}$ the *Base* skills connected to the motor module, $Cs\mapsto_b$ the *Complex* skills connected to the Base skill b, F_{Ms} the internal function of the *Master* skill, $F_y(Vi_y)\mapsto_x$ the element directed to x of the output vector of the internal function F of skill y processing its input vector Vi. The 1.0 factor represents the constant *Influence* given to the *Master* skill, which would be replaced by the *Influence* of a higher level skill when extending the hierarchy.

module then computes the corresponding motor command for its actuator as a normalized weighted sum according to Equation 3. An example of the complete computation of a motor command from the master skill to the actuator in a three level hierarchy is shown in Equation 4.

4.1.3. Training MIND and experimental results

The MIND architecture was evaluated through a series of experiments on the acquisition of navigation and collection behaviour by a simulated robot. MIND hierarchies are trained following a curriculum, beginning with low-level *base skills*, which increases in complexity and requires combining previously acquired skills. The experimental setup involved a curriculum of six tasks to form a behaviour hierarchy of six skills (Fig. 7) enabling an agent to collect objects in an environment while avoiding obstacles. This controlled "shaping" [24] approach to development greatly reduces the complexity of new tasks through the categorizability of context [61]. The experiments also introduced a retraining and learning optimization strategy for skill hierarchies, along with strategies for the establishment of curriculums inspired by Robot Shaping [24] and coevolution [62].

To demonstrate the suitability of MIND for lifelong development, the collect behaviour was then extended to include energy management constraints, along with new inputs and physical elements indicating the battery level of the agent and the position of a recharge zone. In MIND, each new skill has independent access to sensor data [52] and performs local coordination of the added elements (software, hardware), including elements not present or needed during the formation of the previous skills. This experiment shows that new behaviours and physical elements can be integrated without any impact on the previously acquired behaviours, which remain available for future combination and will not have to be learned again. Finally, a proof-of-concept was deployed on a real world robot using the skills trained in simulation.

Videos are available for each experiment [63–65] and the preliminary robotic application [66].

4.1.4. Benefits and limitations

MIND is by design an architecture supporting the lifelong development of artificial agent, its modularity reflects the modular and hierarchical nature of complex tasks and is the key to the progressive learning and accumulation of skills. Encapsulation into modules provides stable and identifiable skills [6]: it defines an area of responsibility for the skills within the global behaviour and makes them available for combination. This offers a great flexibility in building hierarchies: other levels are not concerned with how a module accomplishes its function, and modules performing similar functions can be substituted. For instance a skill relying on a *reach target* subskill is not concerned if the target is reached by flying or swimming. Stability means subsequent training makes use of previously acquired skills without altering their purpose, thus avoiding conceptual drift and the risk of catastrophic forgetting which would impair all other skills relying on them. Existing hierarchies can be built upon, sub-hierarchies can be retrained or replaced, and new modules can be added to interface additional physical elements. At each step, the target behaviour for which an agent is

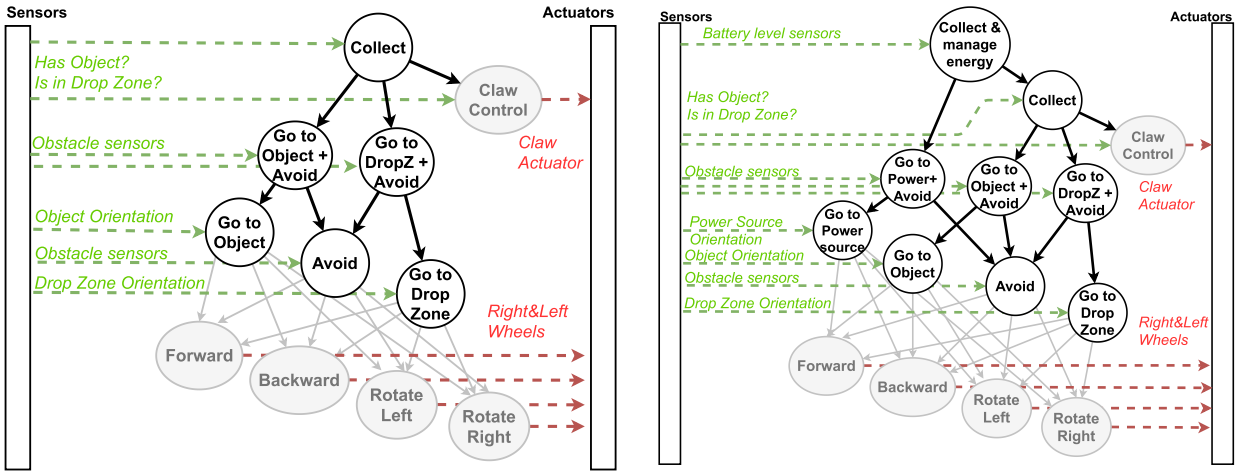


Fig. 7. An overview of the division into skills. On the left: the original Collect task, on the right : the *Collect* hierarchy with the addition of energy management.

trained can be an element for the future creation of one or even several behaviours of greater complexity, whose purpose cannot be anticipated.

Compared to underlying architectures of other works focusing on autonomous development, such as GRAIL [60], the use of different decision techniques within the same system is extended to higher levels. Variants of GRAIL [67,68] such as C-GRAIL, M-GRAIL can be implemented in MIND as different internal functions of complex skills. Since it does not introduce an explicit two-level separation, such as "goal selectors" and "experts", the subdivision of areas of responsibilities of skills can be finer and gradually distributed over multiple levels. There are no restrictions on the decision technique used at each level, and neural based approaches can be used to control lower level skills. Finally, multi-level hierarchies enable MIND to overcome the limitation of M-GRAIL [67] on the reuse of previously learned "sequences", as illustrated by the extension of the collect hierarchy with energy management (Fig. 7) and object counting (Section 5.3).

Further research involving MIND aims at the integration of mechanisms for intrinsic motivation, autonomous structuration of hierarchies, human-robot interfaces and take further steps toward a complete architecture for developmental robotics, leaning towards the emergentist paradigm of cognitive architectures. Application to robotics and drones, as well as research on multiagent problems are being investigated. However, a more fundamental concern is the lack of any form of memory systems or ways to represent information inside hierarchies.

Fairly complex behaviours can be performed using a reactive control mechanism [51], and other works have shown that MIND reactive hierarchies are able to support coordinated multiagent behaviours [69]. However, some behaviours require the ability to memorize or represent information. It can be as simple as retaining information about the environment which is no longer observable, or much more complex mechanisms involving the evolution of an internal state over time, for purposes such as social specialization in a homogeneous population [70]. As discussed in Section 3.4, memory elements could be added to the skill internal function, using for instance RNNs, but those memory elements could not be accessed or shared between skills due to encapsulation. Each skill would have to learn and manage its own representations, and those representations could not be used for the creation of future behaviours.

To address this issue we provide a memory and representation system at the level of the hierarchy, were its elements can be reused by multiple skills in a developmental manner. This memory system conforms to the principles of MIND and interacts with skills through the *Influence* mechanism. It enables agents to commit to behaviours, beyond a purely reactive level, and supports lifelong development of internal representation. Each new concept is introduced along with related skills, the representations acquired are grounded in the behaviour, and remain available for future combination, supporting development in the same way skill modules do.

On a structural level, internal representations enable skills to exchange information through the *Influence* mechanism. This is immediately beneficial for basic navigation which can be generalized to a single skill using the concept of "target" assigned by higher level skills, instead of being bound to a specific sensory input (object, drop zone, power supply).

The following section shows how our memory and representation system is implemented as a new type of module for MIND: the *Variable* module.

4.2. The memory system supporting non-symbolic representation: the variable module

To meet the requirements of lifelong development and comply with architectures handling the development of procedural knowledge, we follow the guidelines highlighted in Section 3.4 and provide a **(1) general-purpose** non-symbolic memory system **(2) integrated** with procedural knowledge. Memory elements are **(3) identifiable, stable and reusable** and support the **(4) structuration** of behaviour. The development of values is grounded by the same learning process used by the skills, leading to **(5) emergent** representations and could support the design of **(6) explainable** systems. This is achieved by conforming to the modular principle

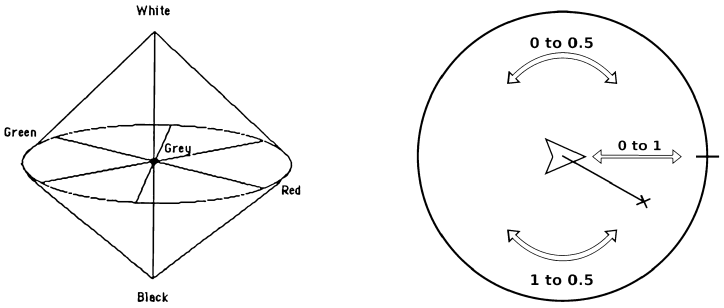


Fig. 8. On the left: the HSV representation of colour as an example of a conceptual space defined by the 3 dimensions of hue, saturation and value (from [28]). On the right: Polar coordinates relative to an agent. The 2 dimensions can be represented by 2 MIND variables.

of MIND and its signal approach for representing information, enabling the close integration with skills to form representations that are emergent and persistent.

Modularity: To support the development process of the agent, the information, internal states or concepts for which representation are formed are implemented as *variable modules*. They are accessible by skills at hierarchy level, and subjected to the same mechanisms for regulation, access and use as other information carrying modules (*i.e. sensor* and *motor modules*). This ensures that *variable modules* are regular components preserving the unicity and scalability of the underlying approach. *Variable modules* can be used by skills as input to receive shared information, or as output to share information and synthesize several concurrent commands according to the *Influence* mechanism. *Variable modules* exist independently of skill modules, and skill modules depending on them are, to some extent, able to function independently of *variable modules*. Indeed, skills organized to exchange information through *variable modules* will not fail because of missing input when a skill is removed or rendered inactive, as they would if a direct exchange between skills was allowed. The *variable module* acts as a buffer for information exchanged between skills which allows alternative structuration of hierarchies, such as in-line or branching information processing structures, while keeping the benefits of the *Influence* mechanism. As a module, it is identifiable and addressable, and its shared use in multiple behaviour ensures the stability of its dedicated purpose in the same way skill modules provide identifiable and stable behaviours. These internal states are observable, and could support the design of explainable systems.

Representation: Variable modules are intended as a general purpose representation system and must handle any kind of information, such as the orientation of a target, time, an alert level, a role, etc. An individual *variable module* can represent a simple concept, a *conceptual space* limited to a single *quality dimension* [28]. To form complex *conceptual spaces*, several variables can be provided together to a skill. Fig. 8 illustrates the notion of *conceptual space*, with the psychological representation of colour, and the position of a target in polar coordinates. Conforming to the signal approach of MIND, the concept is represented using a real number bound between 0 and 1 which allows for a rich non-symbolic representation. Applied to motor control, this approach is able to represent a binary choice (above 0.5, for activation, under 0.5 for deactivation) as well as the fine control of an actuator (from 0 to 1, from full speed forward to full reverse). Applied to declarative knowledge, it is independent of type of knowledge it represents, and its continuous range can be divided into as many classes as needed: 4 seasons, 10 roles, 24 hours, 26 letters or 360 degrees, the limit is placed on the precision of the function interpreting the information.

Emergence: The integration of a medium for non-symbolic representations and its instantiation alongside new skills during the agent's development results in strong interactions between behaviour and internal representations in every new learning situation. Inferences made on the memory elements, their meaning and interpretation, depend on the skills manipulating them, and whatever particular form these may take, they are perceptually or behaviourally grounded by the same learning process used by the skills, leading to emergent representations. When learning sensorimotor behaviour, skills attempt to improve their efficiency by finding the appropriate commands to send to the actuators. These output commands are translated by a driver layer into the specific command codes of the actuator and, as this driver layer is given, skills have to adapt to its expected values. In the case of the *variable module*, skills will both provide and interpret the information of the variable. This means that skills writing into a variable and skills reading that same variable will have to converge on the meaning of its values through an emergent process, by subdividing this variable into classes whose bounds are grounded in experience. For instance, if a skill would set a variable to represent one of 3 possible roles an agent can play, it would pick 3 different values to represent these 3 roles. A skill which reads this variable to act according to its role will have to learn to classify this value into 3 different classes. The process is comparable to languages games between agents [10], happening here between skills within a single agent. The agreed representation is valid as long as it improves the behaviour of the agent in the world.

Persistence: The information contained in a *variable module* persists until overridden, and can be used as memory. Its application parallels *iconic* and *fragile memory*, up to *working memory* in humans, as a way to store past observations and information in an intermediate stage of processing. Beyond commitment to tasks and low-level cognitive behaviours using working memory, persistent information shared explicitly within the architecture offers the possibility of representing internal states subject to gradual evolution and reinforcement. It enables the design of agents emulating physiological or psychological states, which can be instantiated as

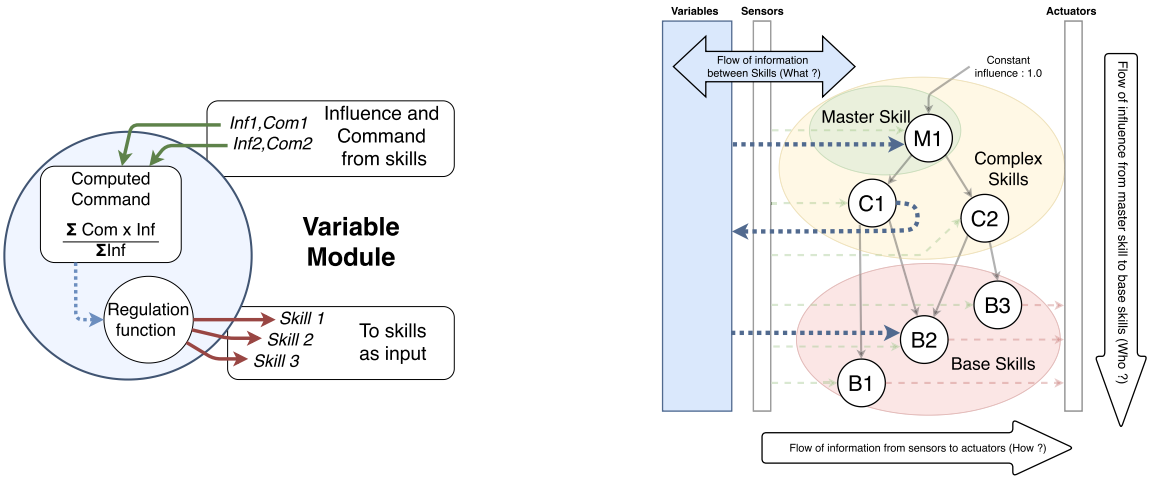


Fig. 9. Left : Internal architecture of a *variable module*. Right : Variable integration in a MIND hierarchy.

identifiable elements. Such internal states can regulate behaviour and vary over time by aggregating information, such as feedback from various mental processes. This opens many possibilities for the future integration of various motivational systems such as those based on *drive reduction theory* [39].

Implementation: Variable modules provide information to skills in the same way *sensor modules* do. *Variable modules* can also receive commands from skills in the same way *motor modules* do. The transmission of a command to a *variable module* conforms to the *Influence* mechanism, the value of the command is determined in the same manner as for motor commands. Fig. 9 shows the diagram of a *variable module* and how it integrates into a skill hierarchy.

Once the input commands coming from skills have been processed, a regulation function is applied before making the result available to other skills. In its simplest form, it is a linear transfer function and outputs the normalized input commands. In the following section, we present possible implementations of the regulation function which allow *variable modules* to serve varied purposes.

4.3. Variable implementation

The regulation process offers many possible implementations for variables. As skills can accommodate many kinds of internal functions, *variable modules* support any regulation process that conforms to the input/output rules of the *Influence* mechanism. Thus, *variable modules* may serve many purposes such as memory management, counters, signal generators, internal clocks, etc. The choice of function is left to the designer, as is the possibility of adding new types of functions.

Here follows an overview of 3 implementations of the *variable module* developed for our experiments: the (simple) variable, the counter and the sinusoid wave generator.

4.3.1. Simple variable

The *simple variable* uses a linear transfer function as its regulation function. Its output value is the direct result of arbitration between the commands of the input skills, computed following the equation 3 for motor commands, Section 4.1.

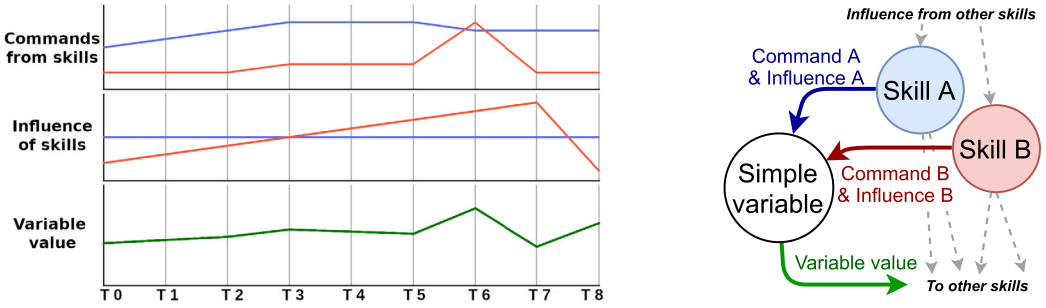


Fig. 10. *Simple variable*. On the right: a diagram of two skills writing to a *simple variable*. On the left: the evolution over time of the commands, *Influences* and resulting value of the variable. The top graph shows the commands sent by the two concurrent skills, the middle graph shows their respective *Influence* and the bottom graph shows the evolution of the output value of the variable.

Other variable types extend the behaviour of the *simple variable* by applying their regulation function to this processed input. Fig. 10 shows the evolution of the value of a *simple variable* linked to two input skills. This example could represent the priority level for a mining/collector robot to return to its base, one skill writing its assessment according to energy consumption, the other according to its cargo load. As the relative *Influence* of the red skill increases, its effect on the value of the variable becomes noticeable, at T6 the effect of the red skill's command is clearly visible.

The *simple variable* is the most general purpose variable implementation: it stores raw commands coming from skills and provides them as they are. As such, *simple variables* are very convenient to share rich representations between skills.

The *simple variable* can persist a value over time, however, it is continuously overridden by any input command of non-null *Influence* value, no matter how weak it is. As a result, the use of *simple variables* for memorization requires carefully designed hierarchies and very accurate training of the associated skills.

This is related to the sensitivity issue of the signal based approach, discussed in [11]. Instead of punctual commands or messages between components of the hierarchy, the control mechanism simulates continuous signal updates, and the weakest *Influence* becomes dominant in the absence of a greater *Influence* counteracting it. Although this has not proven to be a problem so far, weak *Influences* of insufficiently trained skills might accumulate in larger hierarchies and disrupt behaviour. Should training skills to the required accuracy be too costly, it was proposed that mechanisms such as activation thresholds for *Influence* could be added to address the issue.

The application of variables to memorization and persistence revives this reflection on the sensitivity problem and the signal approach of MIND. In addressing this issue, many simple solutions inspired by signal processing, biology or electronics can be thought of such as accumulators, capacitors, thresholds, etc. One such solution is implemented as the *Counter* variable described in the following section.

4.3.2. Counter

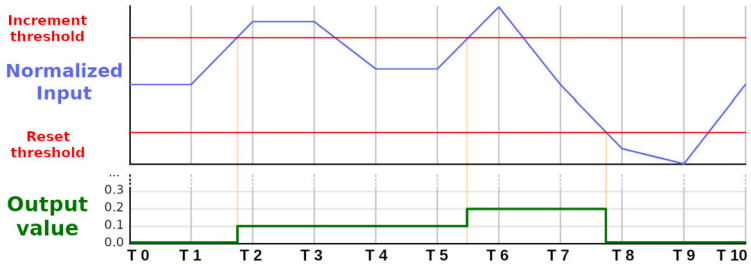


Fig. 11. A *Counter* variable: the rising edges of the input increment the counter, the falling edges reset it.

The *Counter* variable is a simple solution for memorization which records "events" from its input signal and provides a "discrete" count of these events as an output. The events recorded are rising and falling edges of the signal around given thresholds.

The *counter* outputs the current recorded count as a fraction of its maximum possible count. In our default implementation the maximum is set to 10, therefore the possible values for the counter are [0.0 0.1... 0.9 1.0].[2] The counter is incremented on the rising edge of the input value above the *increment threshold*. The counter resets to zero on a falling edge of the input value below the *reset threshold*.[3] In case of overflow, the counter resets to zero.

Fig. 11 shows the evolution of the value of a counter over time. Between T1 and T2, the first rising edge of the input value over the increment threshold brings the value of the counter to 0.1. Between T5 and T6, the second rising edge brings the value to 0.2. Between T7 and T8, the falling edge of the value below the reset threshold resets the value to 0.0.

Compared to a *simple variable* the value is more stable over time: moderate fluctuations of the input which do not cross thresholds do not affect the output value. This greatly diminished the accuracy requirements of the skills manipulating these variables. However, the representation of the information is much poorer (partly due to this "discrete" form) and subjected to some higher level processing (counting "events") which might not be suitable for all applications.

4.3.3. Wave generator

When learning reactive motor skills, providing a random value as input helps the agent unstuck itself from behaviour loops. In these cases, slight variations of the behaviour are acceptable, and taking into account a random value allows the agent to produce different outputs for the same set of sensory inputs.

[2] In the EvoAgents implementation of MIND, the maximum value N of a counter can be provided via the agent configuration file, and is set at instantiation. Its value increments in $1/N$ steps, the only limit for N is the limit of double precision.

[3] Default *increment threshold*: 0.8. Default *reset threshold*: 0.2.

$$Sin\left(\frac{\tau}{\varphi - \varphi \times I + c}\right) \Big/ 2.0 + 0.5 \tag{5}$$

Eq. 5: An implementation of the wave generator using integer tick value as a measure of time. With τ the current tick, I the normalized input, φ the range of frequency values, c a constant. The result is shifted to the $[0, 1]$ interval.

This idea of virtual input fits with the variable system as *generators*: the output is a signal generated as a function of time (or ticks), the input can remain unused or can control a parameter of the generator. In the case of the *wave generator*, the output is a sinusoidal function of a measure of time, the input value sets the frequency.

Fig. 12 shows the evolution of the value of a wave generator over time according to the normalized input value: the higher input value between T3 and T5 causes a faster oscillation of the output value, the lower input value between T6 and T8, a slower one.

Wave generators can have multiple uses, such as measuring time or providing a fluctuating value. Different types of oscillating functions are used in works on evolving virtual creatures [71,19] to synchronize movements between limbs for locomotion for instance. In such cases, one can see the benefit of varying the frequency to accelerate or decelerate synchronized walk cycles.

In our actual implementation, the measure of time used is the update tick of the system, a value which is incremented after each evaluation of the MIND hierarchy. This is a simple method with low computing costs which suits most uses. Such measure is relative to the execution of the system, which means it starts at the same point (tick 0) for each execution, share the same update rate, and is paused and resumed along with the execution of the agent's behaviour. This relative measure is better suited for the purpose of synchronization of behaviours in the hierarchy, for other purposes dependent on a measure of real time (24 hour cycles, powered down periods, etc.) a call to the real time clock system can be substituted to update ticks.

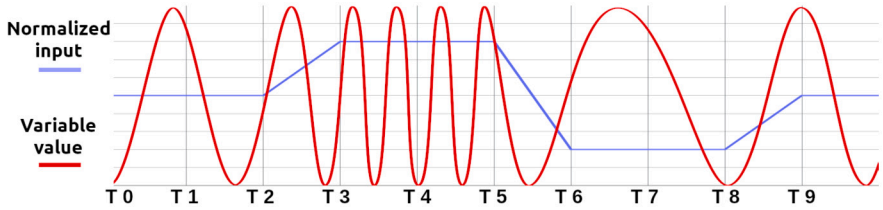


Fig. 12. A wave generator: the input value controls the wavelength.

4.4. An entry point for human guidance and control: the drive module

Exploring the new possibilities of representing and manipulating information within the hierarchy required a convenient way to understand and interact with *variable modules*. When shaping hierarchies, we needed to provide temporary structures and inputs, set up arbitrary goals needed for such training, all of which required tricks and ad-hoc solutions, as shown in Section 5.2.

Ultimately, the *master skill* of a fully developed MIND agent should be able to define its own goals, fulfil its needs, and continue its development *for itself*, following *ongoing emergence* [6]. Another perspective on the purpose of such work is that artificial agents are build to help us achieve our goals and fulfil our needs, which requires a way to control what the agent is trying to learn or

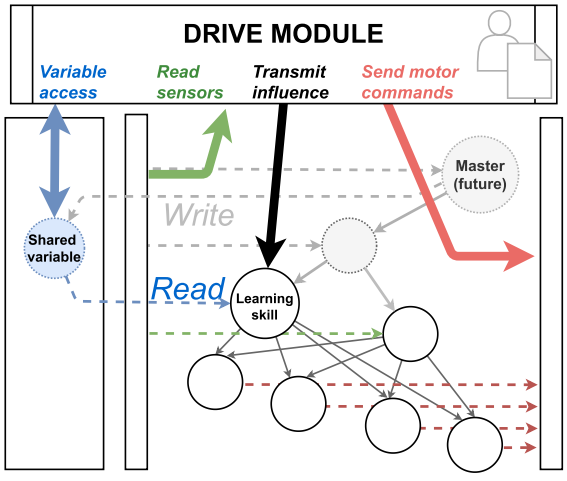


Fig. 13. The *drive module* and its relation to a MIND hierarchy. In this example the *drive module* guides learning by reading from the sensors and writing into the shared variable, taking the place of the *master skill* which has not been learned yet (shown in grey).

accomplish. Most control architectures offer a formal way to input the operator's commands: "mission modules", "supervisors" [72], "mission layer" [73]. The name chosen for such system often reflects the design philosophy of the architecture.

In MIND, this role is played by the *drive module* (a term borrowed from *drive reduction theory* [39]). The *drive module* (Fig. 13) is the entry point in the hierarchy for setting mission parameters and objectives, task configuration, constraints, and can provide adjustable autonomy and temporary teleoperation [74] or human guidance in planning (so called "human-on-the-loop"). It enables the designer to define a programmed procedure called at each evaluation of the hierarchy. This procedure can load and access all modules available to the agent and can be used to set the value of a variable, add a concurrent command to an actuator or send *Influence* to skills, in the same manner as a skill controls subordinate components (actuator, variable, subskill). Appendix C discusses applications and a proof of concept for adjustable autonomy, a demonstration video of this behaviour is available at [75].

In this article, the *drive module* is applied to the simplification of the behaviour shaping process in the following experiments.

5. Experiments with the variable system

To demonstrate the applications of the variable system of MIND, we present two scenarios showing different uses of *variable modules*. These experiments are conducted using the latest version of *EvoAgents*, a framework implementing MIND, along with the learning algorithms and simulation environments required to train agents. Section 5.1 briefly presents *EvoAgents*, the experimental context and training methods used. In keeping with the spirit of lifelong development, the new skills learned in these experiments will reuse as much as possible the skills and hierarchies acquired during the previous series of experiments described in Section 4.1.

The following sections present experiments on the use of variables to store, retrieve and share information. In Section 5.2, we investigate the use of variables as a means to exchange information between skills and organize in-line structures: the output of one skill is the input of another. We reorganize the different navigation skills of the Collect scenario around the "target" concept, represented by a variable whose value is set by another skill.

Section 5.3 investigates the use of variables as memory. In this experiment we learn to count steps in a process which shows no indications of its current state in the environment. The challenge is to learn a useful internal representation through interaction with the environment.

5.1. Experimental context

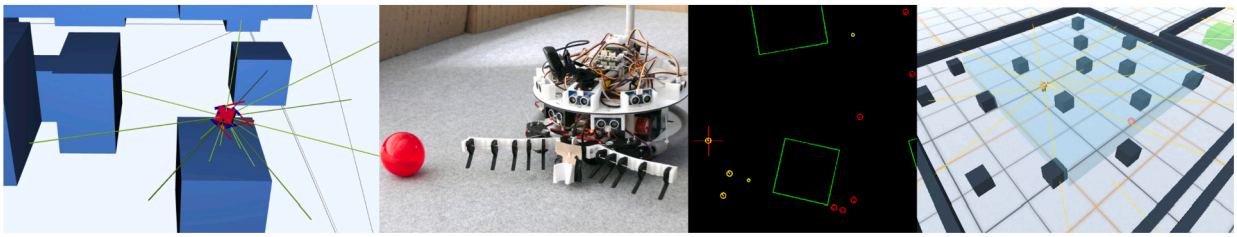


Fig. 14. Interfaces and simulation environments of EvoAgents. From left to right: 3D physics environment, our remote robot, multiagent environment, a remote environment in unity game engine.

EvoAgents is a custom framework written in Java designed to conduct experiments with agents using the MIND hierarchy as their control system (Fig. 14). It includes the implementation of MIND and its different modules, a set of learning algorithms for neural networks and genetic programs based on the Encog library [76] to train the skills internal functions, and a number of interfaces (remote control and simulations) and physics based simulation environments (2D [77], 3D [78], multi-agent). EvoAgents is designed to run parallel learning algorithms in headless configuration and has been used on HPC clusters. EvoAgents is open source software distributed under GPL 3 licence, and available to the community for review or implementation of MIND hierarchies [79].

For the purpose of this article we only consider the 2D physics environment. All dynamic objects have a simulated mass and friction properties, and behave according to laws of motion.

The simulated robot is composed of two drive wheels and a front claw to grab objects. In the physics environment, the effect of each wheel is represented by a force impulse vector on the robot's body at a position and with a direction corresponding to the relative position of the wheel (if only one of the wheels is set to forward motion, the robot rotates in place). The robot is equipped with 18 sensors providing information such as obstacle distance or target orientation. *Sensor modules* are in charge of interfacing sensors with the MIND architecture and enriching the sensory information. They provide a history of past values and derivative to be used as input by skills.

Motor commands which are issued by a *motor module* to its corresponding actuator as real numbers in the [0,1] range, are interpreted by a driver layer as a percentage of the actuator's capability (either power, speed, angular position...). For the wheels, 1 corresponds to full speed forward, 0 to full speed backwards and 0.5 to stopped. For the claw, the value corresponds to its state: above 0.5 is closed, under 0.5 is opened.

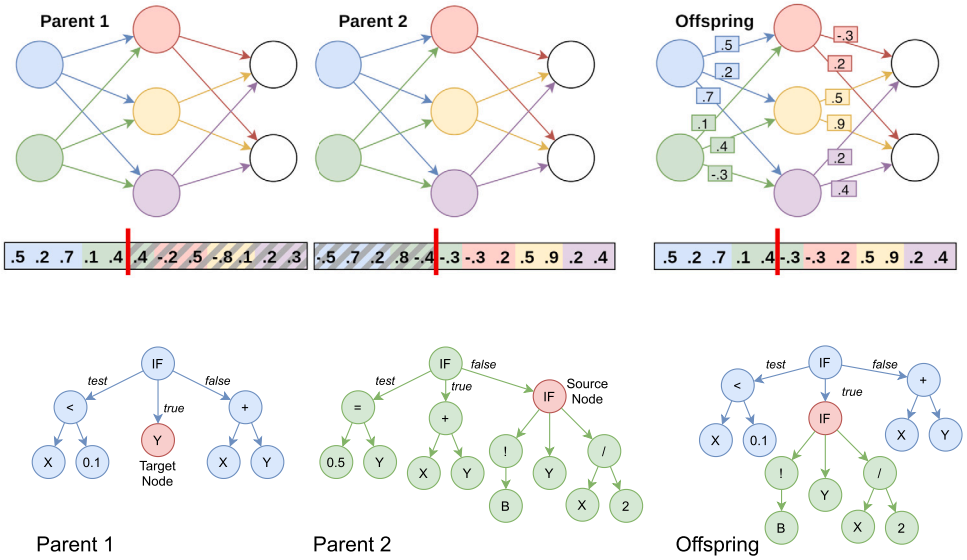


Fig. 15. Examples of crossover operations for neural networks and evolutionary programs. In bright red: the crossover point or its equivalent.

The learning skill modules are controlled by small multilayer perceptrons (MLP) which are trained by a simple genetic algorithm [80] using tournament selection and speciation. We chose this solution for its simplicity, exploratory properties and good performance with delayed rewards, however, other learning algorithms could be used. EvoAgents also include Evolutionary Programming [81] which uses a genetic algorithm to evolve programs using a tree representation comparable to S-Expressions. These Genetic Programs (GP) can only provide one output, for skills requiring multiple outputs we altered the genetic algorithm to consider multiple GPs as a single individual, which we refer to as a Genetic Program Forest (GPF). Fig. 15 illustrates of how MLPs and GPs are altered by the crossover operation of the genetic algorithm. Algorithm 1 details the genome evaluation process for an individual skill.

Algorithm 1: Genome evaluation process for an individual skill.

```
foreach genome in population do
    score = 0;
    hierachy = recursiveInstantiation(masterSkill);
    hierachy_LearningSkill ← new Function(genome_Vector);
    environment = new Environment(taskSettings);
    environment_Robot ← hierarchy;
    for step = 1 to maxStep do
        environment.runStep();
        foreach rewardFunction in environment_RewardFunctions do
            score += rewardFunction.computeReward();
        end
    end
    Genome.setScore(score);
end
```

Hierarchies are trained using a curriculum learning technique, which suits the developmental context and the hierarchical nature of MIND: new skills of increasing complexity are added through new "lessons" aiming at more complex goals.

Curriculum learning [82] are progressive learning methods intended to simplify complex learning problems through decomposition into simpler problems. These methods are used in a number of domains, from function approximation using neural networks to robotics and video games [83]. Curriculum learning subdivides a learning task into different but complementary subtasks (or *source tasks*) to be learned in a given order, generally increasing in complexity.

For our purpose, a curriculum will consist in a series of tasks, each corresponding to a skill of the hierarchy to train, ordered from *base skills* to *master skill*. The curriculum is handcrafted, each task uses a simple environment and set of reward functions, which both simplifies the process of designing learning environments and reduces the cost in supervision during training. The curriculum guides the development of the agent through the key skills to acquire and, as with the separation into different skill modules, each new task focuses on the additional complexity of the new environment and corresponding goals.

Fig. 16 shows how a hierarchy is initially trained using a curriculum: low-level skills are trained as separate tasks, each skill has the role of the *master skill* during the training episode and must cover all the expected behaviour of the task. Higher level skills are

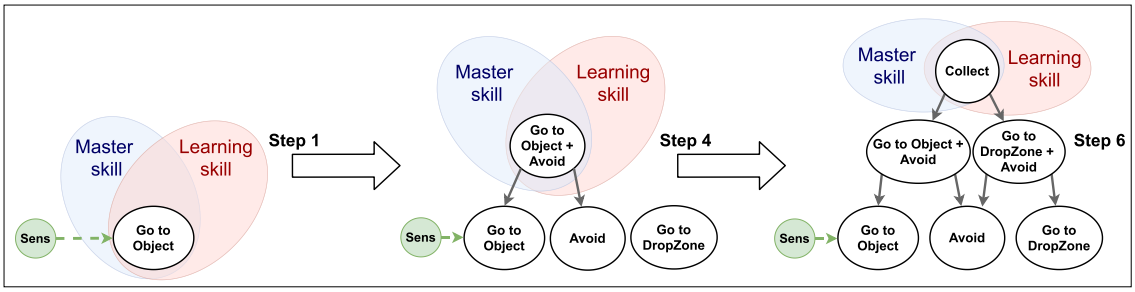


Fig. 16. Using a curriculum to build a MIND hierarchy composed of 6 skills. At each step one skill is trained using one task of the curriculum, the figure shows from left to right steps 1, 4 and 6. Lower level skills are trained first, at each step the learning skill has the role of *master skill*.

then trained to use low-level skills on a corresponding task of higher complexity. Learning from low-level to high-level skills is a general rule, the order of the tasks really depends on having the subskills needed. For instance, in the example given in Fig. 16, the *Go To Object + Avoid* skill could be trained before the *Go To DropZone* skill, as both its subskills are already trained.

True to the spirit of lifelong development of artificial agents, the following experiments will make use of the skills and skill hierarchies acquired during the previous series of experiments [11]. Fig. 7 in section 4.1 shows the skills already available which will not need to be trained again.

5.2. Scenario 1: Target variable

This scenario demonstrates the use of variables for the centralization and exchange of information between skills. This example of application intends to show how the result of a computation made by one (or several) skill can be stored and made available as input for others, offering the possibility of forming in-line or branching information processing structures. The information exchanged can be the intermediate analysis of perceptions or the result of a more complex decision process. It is identified as a concept around which skill structures can be organized.

In this experiment we generalize navigation behaviours by reorganizing the different navigation skills of the Collect scenario around the concept of target. The master skill selects the relevant target and sets a variable with its orientation information. In turn this variable is provided as input to a dedicated navigation skill.

5.2.1. Protocol

The goal is to learn the collect behaviour of previous experiments, described in Section 4.1. The collect task consists in picking up an object and bringing it back to a drop zone while avoiding collisions.

Hierarchy: The hierarchy is a modified version of the original *Collect* hierarchy shown in Fig. 7: the *GoToObject* and *GoToDropZone* skills are replaced by *GoToTarget*.

Our lowest level *complex skills* are:

1. *Avoid*: Move while avoiding obstacles. Its inputs are the 10 proximity sensors. This skill is reused from previous experiments and did not require training.
2. *GoToTarget*: Similar to *GoToObject* and *GoToDropZone*, the orientation to the target is given by a variable (*Target* in Fig. 17) instead of the object or drop zone sensor.

GoToTarget + Avoid combines *Avoid* and *GoToTarget* using the proximity sensors to navigate towards the target while avoiding collision in an environment with obstacles. It substitutes itself to the *GoToObject + Avoid* and *GoToDropZone + Avoid* skills of the original *Collect* hierarchy (Fig. 7).

Finally, the *CollectVariable* skill combines *GoToTarget + Avoid* and the *base skill ClawControl*. It also outputs to the *Target* variable. Its inputs are the object and the drop zone presence sensors, to decide what must be done, but also both orientation sensors for the object and the drop zone to set the appropriate value for the *Target* variable.

The original *Collect* skill simply needs to delegate to the appropriate subskill, setting "*who*" is in charge. Each of its subskills has access to the appropriate information via sensors to determine *how* to accomplish its subtask, the object sensor for one skill and the drop zone sensor for the other. The *CollectVariable* skill must relay through the variable the information needed to determine *how* to accomplish the task, *i.e.* the orientation of either the object or the drop zone, by identifying *what* is the appropriate target (object or drop zone).

Training environment: The initial training environments are similar to the original collect experiments. For instance, the environment for the *master skill* contains an object to collect, a zone in which to deposit the object and obstacles to avoid. The agent is given a limited time during which it can augment its score. A small reward is given for picking up the object, and a large reward for depositing the object in the zone. A collision ends the evaluation.

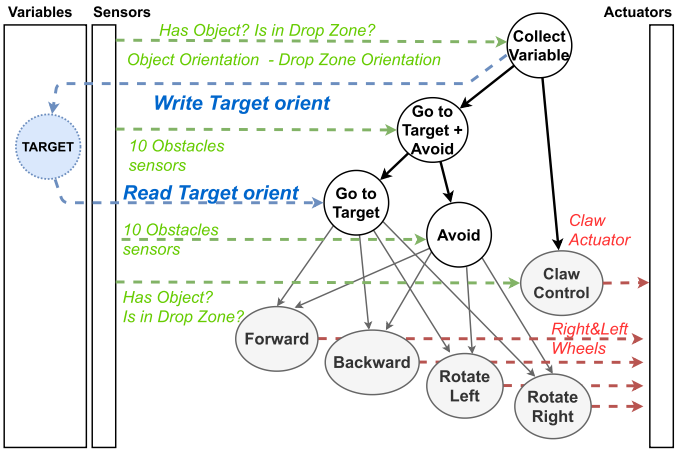


Fig. 17. *Collect* hierarchy using a variable for the target.

The original setup places a single object in the environment which is always tracked by the agent's sensor. When the agent succeeds in collecting the object, it is placed at random in the environment so that the task can be repeated. We found out that this configuration can be exploited by the *Influence* mechanism to simplify the learning task. When an agent brings an object to the drop zone, the object tracked by the orientation sensor is held in the frontal claw. Under these conditions the orientation sensor always reports the object being in front and thus the output given by *GoToObject* is to move straightforward. Using vector composition, the *GoToObject* behaviour could therefore remain active while the agent brings the object back to the drop zone without any adverse effect on the trajectory, simply adding a constant forward movement.

To make sure the agent learns a strict exclusion between the different subtasks of the collect behaviour, it must not receive an input pointing always forward from its object orientation sensor when carrying an object, so that it is forced to ignore this information in favour of the orientation of the drop zone. The experimental setup was thus altered in two ways:

- Multiple target objects are present in the environment. The object orientation sensor will give the orientation of the closest one.
- An object carried in the claw does not register on orientation sensors.

Skills internal functions: The default internal function for skills remains the Multilayer Perceptron. However, in this scenario one of the skills uses Genetic Programming to evolve its internal function. We will see in the result section why this type of controller is better suited for a particular task.

Shaping variable dependent skills and the need for a drive module:

Following the curriculum approach, lower level skills are trained first, higher levels are progressively built on top of the previous ones as illustrated in Fig. 16 Section 5.1. In the *CollectVariable* hierarchy (Fig. 17) the lower level skill *GoToTarget* depends on a variable whose value is set by the higher level skill *CollectVariable* (Fig. 18). To shape *GoToTarget* we provide a value to the variable, the orientation of a target object, and train it with the *GoToObject* environments and its set of rewards.

Our initial solution inspired by context retraining methods developed in [11] uses a *master skill* programmed to copy the value of the object orientation sensor to the *Target* variable and activates *GoToTarget* by sending it a constant maximum *Influence*. However, creating such "scaffolding" skills quickly proved to be cumbersome and highlighted the need for direct control of higher level functions, either for learning or setting a goal for exploitation, motivating the development of the *drive module* described in Section 4.4.

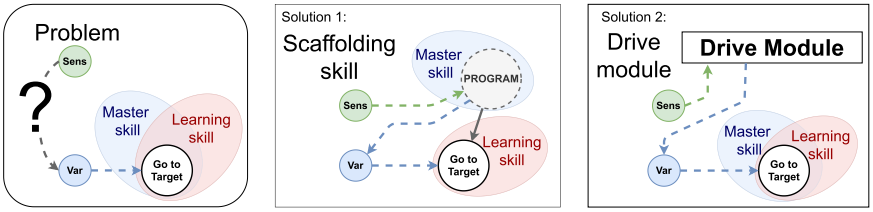


Fig. 18. From left to right: The problem of the *Target* variable, the scaffolding solution using a programmed *master skill* and the *drive module* solution accessing the *Target* variable.

5.2.2. Results and analysis

Videos of the results are available [84,85], for quantitative analysis of the training process, 10 replications with random initial seeds were done for each skill. Fig. B.27 in Appendix B shows the evolution of the average score during training.

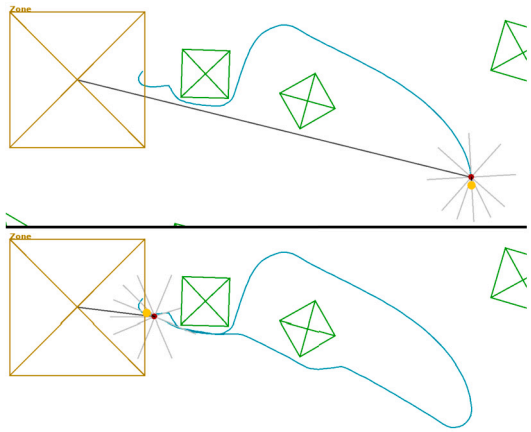


Fig. 19. Trajectory of the *Collect Variable* skill collecting a single object.

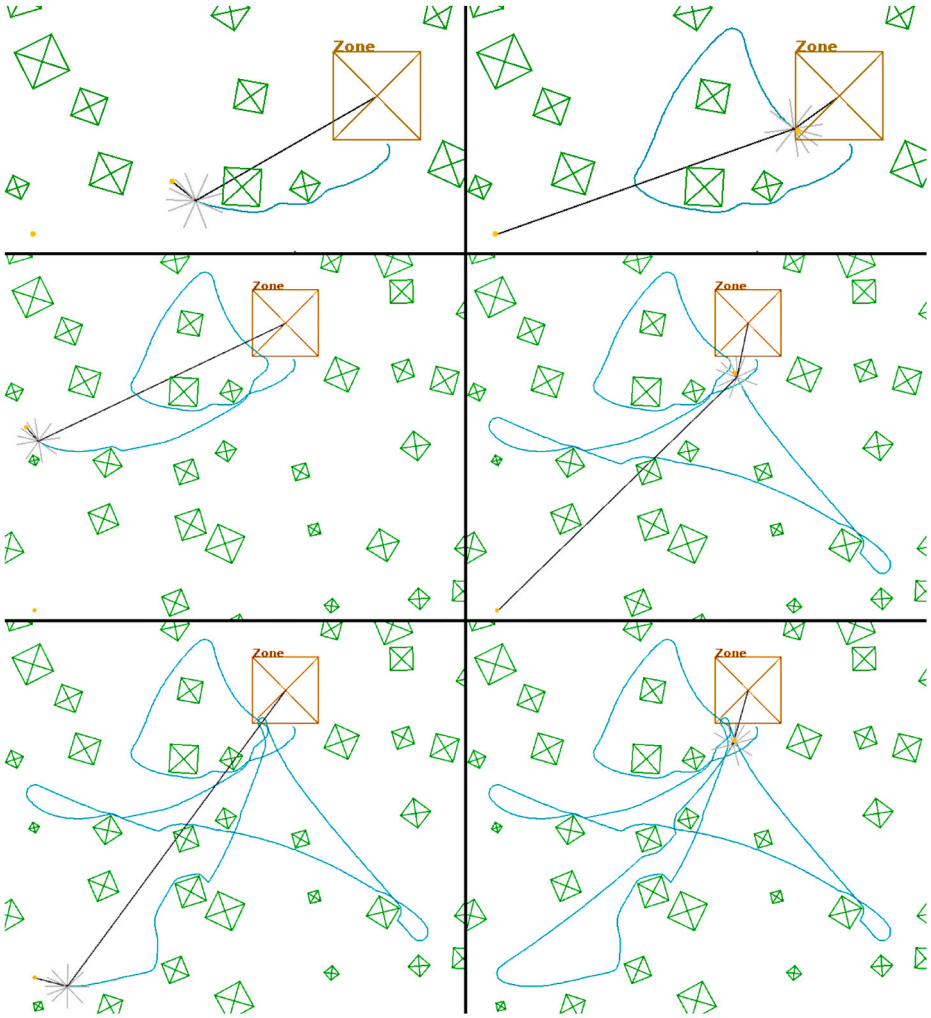


Fig. 20. Trajectory of the *Collect Variable* skill collecting 3 objects (neural network variant).

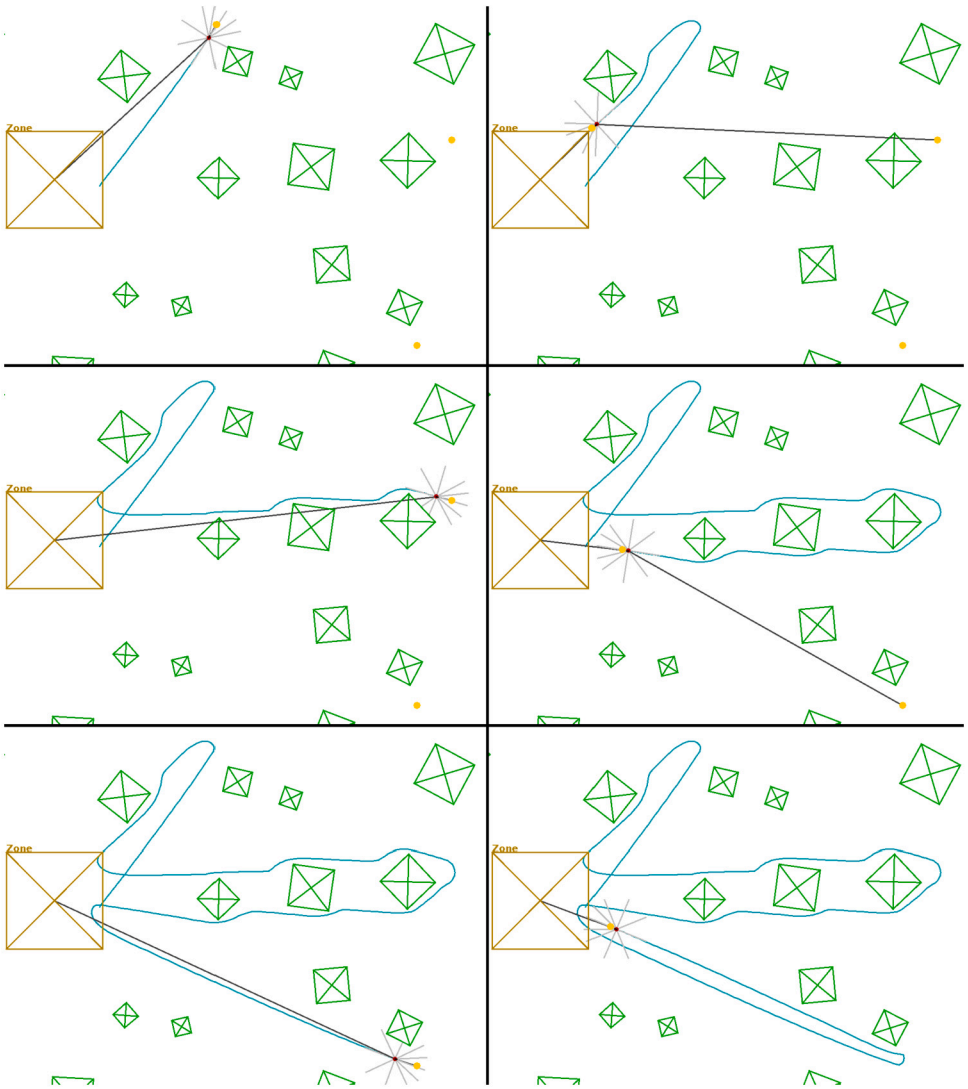


Fig. 21. The optimal trajectory of the *Collect Variable* skill using a function evolved via genetic programming.

Using neural networks: In a single object configuration of the environment the behaviour is similar to the original hierarchy (Fig. 19). Fig. 20 shows the behaviour in the alternate setup with multiple objects: each row shows the collect of an object, on the left the agent is reaching the object and on the right, the drop zone.

When the agent brings back the second object, we can see a loop on the right side of the screen. This is due to the sensory information given by the third object interfering with the orientation to the drop zone. Even if the agent does succeed in collecting several objects, it is clear from observation that this behaviour is far from efficient.

We suspect that this is a limitation of the combined use of neural networks and genetic algorithms on this specific task. The skill must perform a strict exclusion between setting the *Target* variable either to the orientation of the object or drop zone. In addition, this exclusive decision has to let one of the input signals (orientation of the object or orientation of the drop zone) through the network without distortion, which is difficult for a neural network. Fig. A.26 in Appendix A shows that an optimal configuration exists, however it is very specific and its neighbouring solutions have very poor performance which is not suited to our learning algorithm. The configurations we were able to learn are part of local optimums, which are far from the global optimal solution in the configuration space.

We experimented with various topologies and training methods, which slightly improved performance, but specific combinations of inputs still cause erratic behaviour. In all instances we were unable to consistently train a network over several replications.

Using genetic programs: The encapsulation of controllers into skill modules enables MIND agents to combine heterogeneous sets of controllers within the same hierarchy. It is thus possible to select a control function more suited to this task, without losing any of the neural network based skills accumulated so far.

The problem of strict exclusion when selecting the appropriate sensor can be solved by a simple program, such as:

```
if SENSOBJ > 0.5 then                                   /* if the object sensor detects an object carried */
    return RADDZ                                /* set the variable to the value of the drop zone orientation */
else
    return RADOBJ                                 /* set the variable to the value of the object orientation */
end
```

Evolutionary Programming can produce such program, using the same training environment and reward functions. When evolving a single Genetic Programs (GP) to set the value of the variable, optimal programs can be found in less than a few dozen generations. Here are two examples of results, both exhibit optimal behaviour[4] as shown in Fig. 21:

```
IF SENSOBJ ≤ 0.7 THEN RADOBJ ELSE RADDZ
IF SENSOBJ > 0.4 THEN RADDZ ELSE RADOBJ
```

Simple GPs can only provide a single output. To generalize its application to skill modules we modified the GP algorithm to evolve a GP for each output, referred to as a *Genetic Program Forest* (GPF). The evolutionary algorithm treats this collection of GPs as a single individual.

The optimal behaviour for *CollectVariable* is a combination of the previously described program for the variable output and any non-null *Influence* values for the subskills *GoToTarget+Avoid* and *Claw Control*. Solving this task with a GPF only adds little complexity to the search space.

5.3. Scenario 2: Counter variable

This scenario presents the use of variables as a memory and internal representation system to handle counting and memorizing events. Perceiving numbers and quantities is one of the most basic perceptual skill of humans and animals, and constitutes a suitable case study in the gradual development of abstract reasoning skills [86]. Embodied developmental systems have been used to implement psychological models of association between abstract representation and motor skills, for number comparison tasks and incremental count of objects [26,27], the latter task involving Recurrent Neural Networks.

Although we do not intend to implement any specific psychological model, our approach shares a similar experimental context: the goal for the agent is to learn a representation through physical interaction, which enables it to perform an incremental count using our variable mechanism.

In this experiment, the final desired behaviour of the agent is to perform a sequence of actions, without any indication in the environment of which step of the sequence it has already accomplished. By memorizing the current step of the sequence, we depart from purely reactive behaviour and begin to investigate emergent memory representations and the process of evolving low-level cognitive functions from the ground up.

To study the emergent aspect of representations, we use a protocol that does not allow the teaching entity direct access to the memory modules. The learning process remains a genetic algorithm dependent on a reward function determined solely by observing the behaviour of the agent in the simulation, and not the behaviour of its internal mechanism and states. Respecting the barrier of the interiority of the agent's mind allows the agent to form representations based on its individual experience.

5.3.1. Protocol

The agent must collect an object twice, then activate a validation trigger by entering a specific zone. This expresses, through a motor skill, that the agent has managed to count and memorize a given number of events. This sequence is repeated indefinitely (Collect-Collect-Validate-Collect-Collect-Validate...). A small reward is given for each object collected and a large reward is given for a full sequence. An incorrect sequence (3 objects collected) ends the simulation.

Hierarchy: The hierarchy (Fig. 22) extends the original *Collect* hierarchy.[5] The master skill *Count Objects* combines the *Collect* skill with *Go To Valid + Avoid*, which is a duplicate of the *Go To DropZone + Avoid* skill using a different orientation sensor to drive the agent to the validation zone while avoiding obstacles.

[4] Values for binary sensors such as SENSOBJ are 0.2 for false and 0.8 for true.

[5] The *CollectVariable* hierarchy was still under development at the time of this experiment, however they are interchangeable.

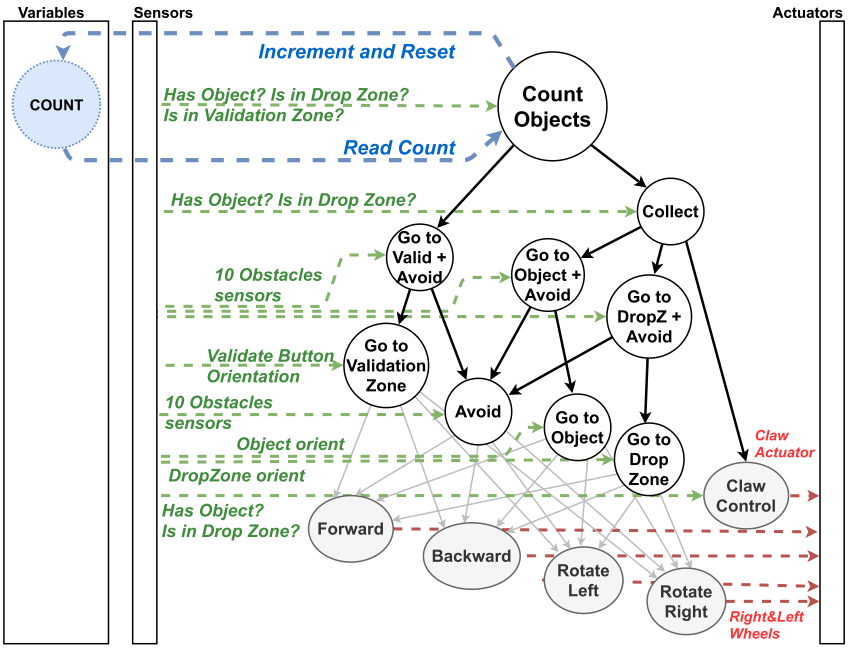


Fig. 22. The *Count Objects* hierarchy.

Count Objects uses the object, the drop zone and the validation zone presence sensors. The *Count* variable is used as input, to choose between the collect or the validation task, and used as an output to increment and reset the sequence count.

Training environment and skills internal functions: All skills use simple multilayer perceptrons. The environment contains the drop zone, the validation zone, obstacles and a single target object.

5.3.2. Results and analysis

Videos of the results are available at [87]. Figs. 23 and 24 show the resulting behaviour and the state of the MIND hierarchy over 5 steps of the process.

1. The agent brings the first object to the drop zone, VAR_COUNT is at 0.
2. The agent just dropped the first object, VAR_COUNT is incremented by 0.1.
3. The agent brings the second object to the drop zone.
4. The agent just dropped the second object, VAR_COUNT is incremented by 0.1, bringing it to 0.2. The agent is now headed to the validation zone.
5. The agent just reached the validation zone, VAR_COUNT is reset to 0. The agent is headed for the object.

The agent was able to learn to count through exposure to a (simulated) real world problem. The learning supervision did not have access to the internal state of the agent, nor did it teach a predefined symbol or memory representation. Using MIND, our agent is able to go beyond simple reactive behaviour and base its decision process on an internal state independent of the environment, which persists over time. In turn, the decision process is able to affect the internal state.

The relation between internal states and decisions, and the effect of decisions on internal states are learned by the agent. Hence, the meaningful values of the internal states, the non-symbolic representations or "prototypes" (see Section 3.3), are formed in an emergent process and grounded in experience.

In the results shown in Figs. 23 and 24, the agent learns to count from 0, successively increments until reaching 0.2, and resets the value to 0 when starting the sequence over. In other attempts at learning the same behaviour, the agent starts at a value of 0.1, increments to 0.3 and resets the counter to the value of 0.1. This can be parallelled to the formation of different semiotic networks within a population of agents that are nevertheless able to function together [10], both representations are able to support the same behaviour. It is interesting to note the emergent aspect these representations, the formation of one or the other is due to the genetic process and the bias induced by the random initialization of the networks.

The goal of this experiment was to collect a fixed number of objects. This constant (*viz.* 2) was given through the training process and reward functions, as part of the expected behaviour. Further experiments could investigate the collect of a variable number of objects, with the desired count provided as an input from the environment through a sensor, or as a request from an operator through the *drive module*.

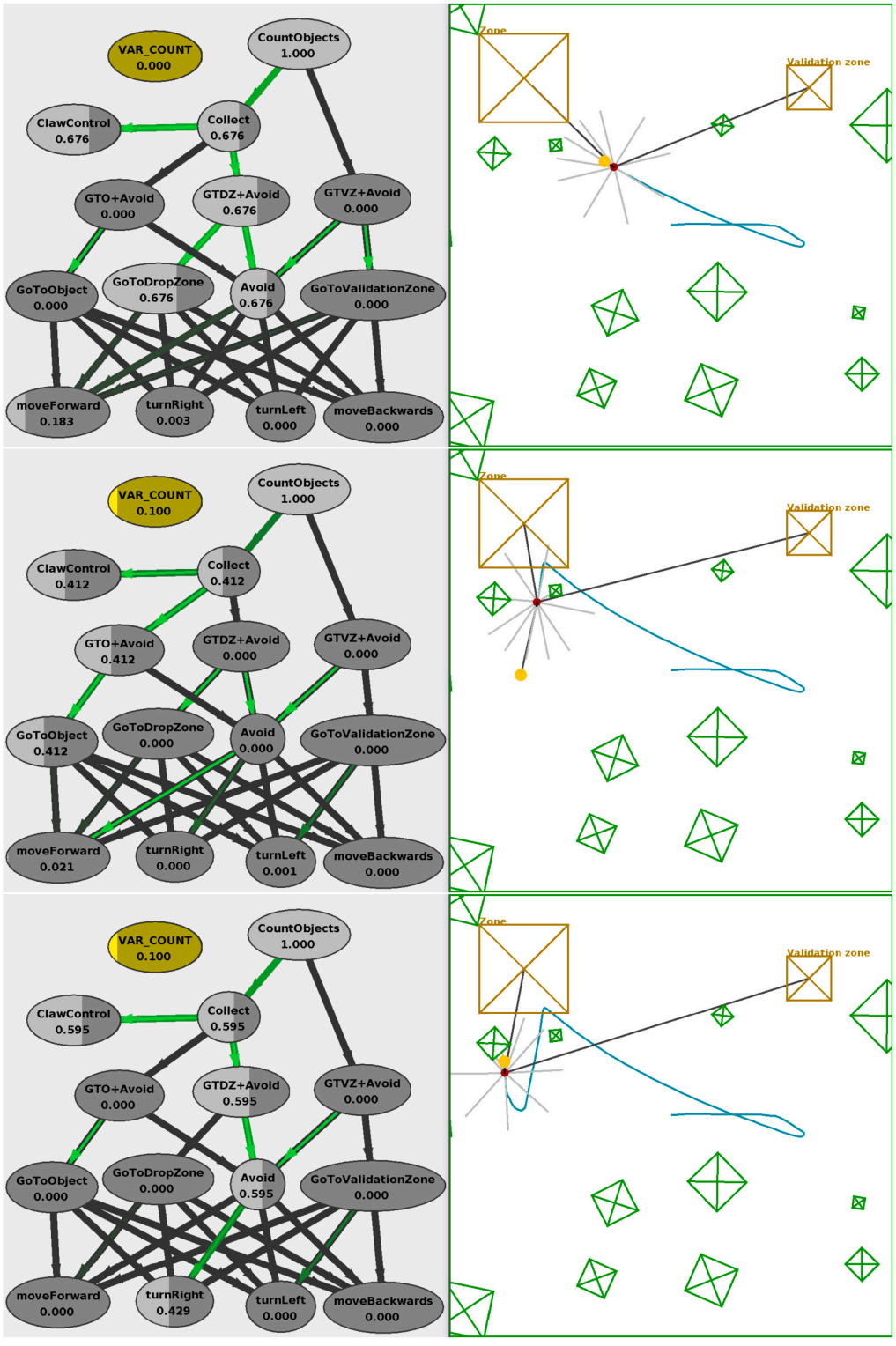


Fig. 23. Step 1-3 of the *Count Objects* behaviour. On the right the simulation, on the left the state of the MIND hierarchy.

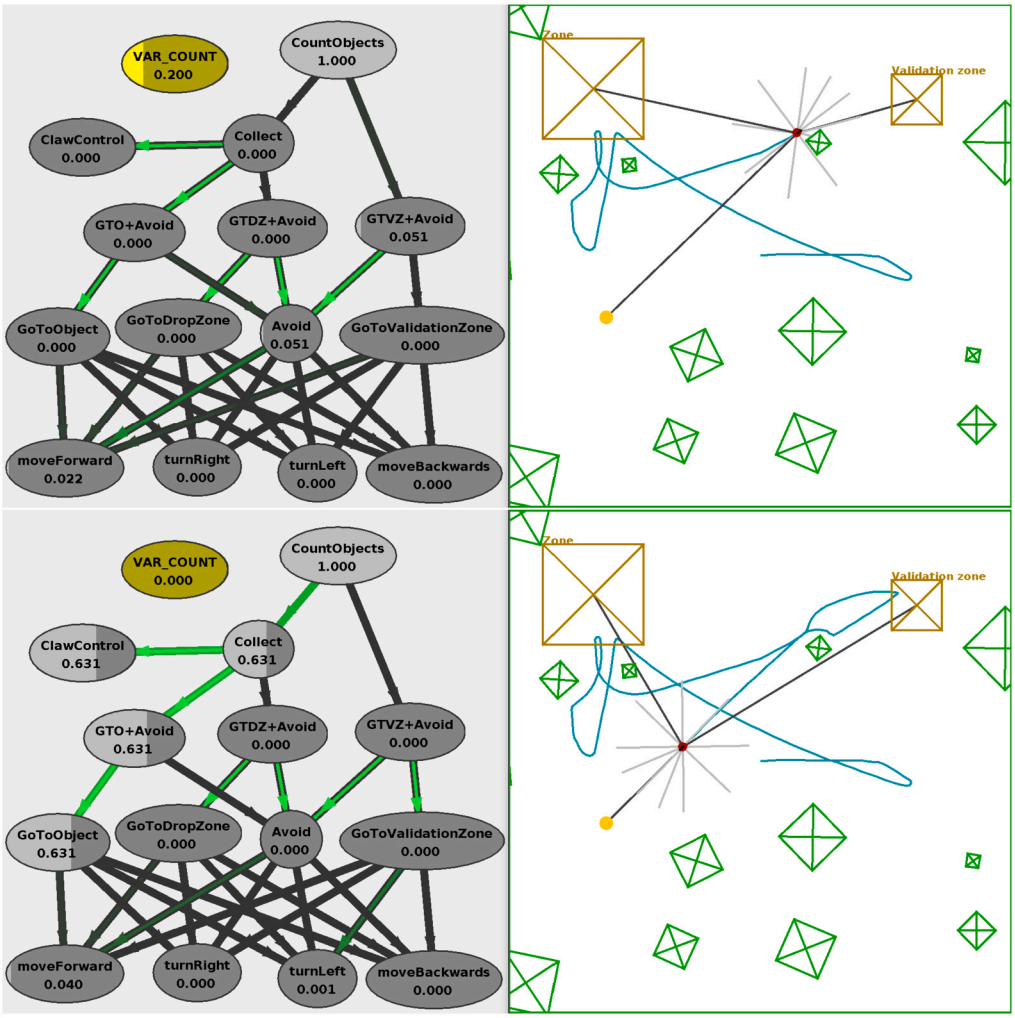


Fig. 24. Step 4-5 of the *Count Objects* behaviour. On the right the simulation, on the left the state of the MIND hierarchy.

The training would involve creating and interpreting representations that can match the different number of requested objects and deliver them (in effect, a task similar to the number comparison performed in the ICUB experiment [26]). Moreover, we could investigate complex uses of *counter variables* such as the combination of several variables to form a multidimensional concept space. For instance, with *counter variables* limited to 11 distinct values (in the interval $[0, 10]$), could it be possible to learn to count beyond 10 by combining two *counter variables*? Could we train a behaviour to count to 20 using a representation of 2 times 10? or even to a hundred by using the two dimensions to represent the units and tenths digits?

5.4. Assessment of variable module use in MIND

The *CountObject* behaviour did not require any alteration of the training methods used in previous experiments. The only real challenge in training hierarchies using *variable modules* was in scaffolding the behaviour for the *CollectTarget* task, and much of this process was streamlined by the introduction of the *drive module*. *CollectTarget* is also the first case where a simple multilayer perceptron was not suited as skill internal function: it could not be trained to provide the specific values, or representations of information, required as output. Because of our choice of scaffolding method, the representation expected in the *Target* variable by *GoToTarget* is a direct copy of the value of a sensor. Other representations might exist, which could be learned by a multilayer perceptron. In any case, MIND offers the possibility of combining different types of skill functions which completely by-passed this issue.

The final behaviour of each experiment could be learned without using variables: *CollectTarget* is functionally identical its reactive counterpart, and since *CountObjects* is the only skill using the *Count* variable, it could probably be trained using an internal function containing its own memory system within the skill module (such as a RNN [27]). Nonetheless, the success of these experiments

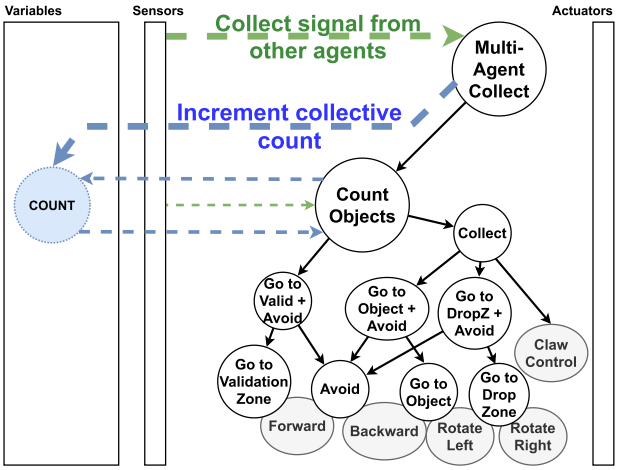


Fig. 25. A possible extension of the *Count Objects* hierarchy for multiagent coordination.

ensures that the MIND can use variables to learn such behaviours just as well. The actual benefit of the *variable module* resides in its potential in a lifelong development context, and the further acquisition of skills based on previously acquired representations.

In the *GoToTarget* experiment, a navigation skill which fits most cases is learned once and for all, independently of what could be considered a target later on. Indeed, any new navigation request will simply provide the required information through the *Target* variable using the appropriate representation, regardless of the kind of new objects or positions to consider, and the corresponding new sensors or sources of information. Furthermore, a "target" no longer has to be a direct perception of an element of the environment, and can be a position chosen by calculations and decisions based on multiple sources of information. For instance, the position of both flags forming a gate in a slalom race can be combined to give an orientation guiding the agent through the gate, while in a different context the position of goal posts and defending opponents could be combined. Learning generalized behaviour favours re-use of skills, which suits the philosophy of agent development, and from a designer's perspective, helps in establishing a robust library of skills.

In the case of *CountObject*, the information about the current step of the process can be observed and accessed by other skills (or human observers). When extending the agent's behaviour, this information could be used by other skills in planning and optimizing decisions, for instance in answering questions such as *"Am I almost done with a rewarding sequence? Should I complete my task or abandon it to recharge batteries?"*. It may be the case that further development of this agent would call for multiagent cooperation to accomplish the sequence task. In this case, agents would signal others when an object is collected so that each of them can increase their own count variable. Fig. 25 shows how the *CountObject* hierarchy could be extended for this task.

6. Discussion

6.1. Analysis

The purpose of the *variable module* for MIND is to allow the circulation of information in hierarchies and act as a general purpose memory system. Its integration is achieved by following key aspects of the architecture: the modularity providing flexibility, identifiability and stability; the signal approach of the *Influence* mechanism providing a general purpose non-symbolic representation of information; and its integration with skills and the learning of procedural knowledge to support the developmental process. The experiments with *variable modules* demonstrate different aspects of this contribution: how an independent module carrying information can be used and its effects on the structuration of hierarchies, and how representations can be learned and memorized to develop beyond reactive behaviour.

We have seen how *variable modules* can be used by skills to store, retrieve and share information, and how it affects the internal organization and structuration of hierarchies. We have shown how a higher level skill can share information with a lower level skill, which can now function as parameterized skills [23]. It is also possible for a lower level skill to send information, such as feedback on the result of its actions, to a higher level skill and affect its decision process, thus allowing the formation of hierarchies similar to Robot Shaping [24]. Used as a buffer for information, *variable modules* allow centralizing and redistributing intermediate analysis of the environment or the results of decision processes identified as concepts, which can be addressed by the variable's name. Compared to other connectionist approaches previously discussed, such as layered learning [21] or modular neural network policies [22], the use of variables as interface between controllers follows the same principle of lifelong development which guides the acquisition of skills: the concepts represented can become elements for the future creation of behaviours of greater complexity, whose purpose cannot be anticipated.

The generalization of behaviours is one of the benefits of sharing information between skills, as illustrated with navigation in Section 5.2. The hierarchy used in the example of the *Target* variable is a very simple one, but *variable modules* can be used as any sensor or actuator, which means several skills in the hierarchy can use the same variable as input and send concurrent commands to the same variable as output. For instance, skills such as *GoToTarget* and *FleeFromTarget* would use the same target variable for different purposes. Since the *Target* variable represents a heading, one could imagine a version of *Avoid* setting the heading of the agent to avoid collisions being concurrent with a *GoToObject* skill on the *Target* variable. Generalization also reduces the number of skills, which is beneficial form a designer standpoint to organize and maintain libraries of skills, as well as reduce the memory requirements for applications in embedded systems.

Representing information in identifiable modules favours the explainability of global behaviour and supports the creation of systems for human control and interactions. We have seen in Section 5.2 how these modules can be manipulated through the *drive module* for training purposes, and Appendix C shows how it can be used for adjustable and semi-autonomous operation, where a human operator can set alternative goals for the agent through the manipulation of the *Target* variable. The idea behind this approach is to enable control of the agent through the manipulation of *Influence* signals and variable values. This is a subtle difference between asking the agent to "collect objects" and giving the agent a need for objects, the latter does not constrain the agent regarding how the objects are obtained. This approach is integrated and fits our future plans for the autonomous acquisition of a large collection of skills: several ways of accomplishing the task may be considered and, given an intrinsic motivational system [14], the agent may even evaluate the need for the acquisition of new skills to satisfy its drives.

Sensor modules already provided a low-level memory system, a history of past samplings, but with the *variable module*, any form of processed information can be memorized offering the agent the ability to commit to higher level behaviours. Since this information is shared by skills with the ability to learn on both sides of the process (storing and retrieving), the meaning of a variable's values has to be determined through an emergent process. The example given in Section 5.3 shows the emergence of the meaning of "enough" by subdividing the values of a variable into classes whose bounds are grounded in experience, and that different representations can emerge for the same meaning.

However, the scenarios we presented only constitute limited experimentations in learning grounded representations, as we bypassed the difficulty in converging on an acceptable representation shared by two separate skills. In the first experiment, the representation used for the *Target* variable shared by the two different skills cannot be considered as autonomously acquired. The receiving skill *GoToTarget* was first shaped using a direct copy of the orientation sensor as a representation, the *CollectVariable* skill then adapted to provide the representation expected by *GoToTarget*. The second experiment was intended, and did succeed in investigating emergent representations along with memorization, however the count variable is used by a single skill which has the opportunity to decide both how to represent information and how to interpret this representation.

Linking new skills to an already established shared representation should not cause any particular issue, the new skill will conform to the given representation as was the case in the first experiment. This is comparable to symbol grounding in a population of agents where new agents introduced in an existing population will adapt to the pre-existing vocabulary [48], as following the path of the least resistance. Acquiring the original shared representation is a more delicate matter: if we are to compare the problem to symbol grounding again, the language game between agents [10] is a back and forth process between two learning entities. However, both sides of this interaction are identical in function and do not have assigned roles or area of responsibility as skills do. Some clues may be found in coevolution [62] where multiple skills composing a global behaviour are learned together, sharing the same task and training episode, although in this work the responsibility of each skill (assigning what sub-behaviour each skill is in charge of) is given by the designer in the form of a decision tree. We pointed out [11] that assigning each skill its area of responsibility is part of the function of curriculum learning, creating a bias in the distribution which can be exploited by a coevolution mechanism.

A simple solution would be to initialize representations using a single skill, as with the *Count* hierarchy, and link other skills after a useful representation has started to form. Tasks dedicated to the initialization of a representation system can be included in curriculums, comparable to children's learning through play. From a more technical standpoint, this approach is comparable to auto-encoders forming their representation through a process having no practical purpose, the decoding part being then discarded to train a useful behaviour on the basis of this representation, an idea that inspired modular neural network policies [22].

6.2. Limits

Our platform implementing MIND is limited by the few control functions available at this time, and no attempt was made so far to implement available state-of-the-art learning techniques for training skills. Further iterations of the EvoAgents platform will integrate such components, as needed or per request, without the need for any change to the process described in this article. Indeed, over the years multiple 'off-the-shelf' software solutions and algorithms have been integrated, from simulation environments to machine learning libraries, while the process of building behaviour hierarchies using a curriculum remains the same.

The major concern remaining for the implementation is in bridging the gap between the skill learning techniques used and the curriculum established by the designer. While the curriculum approach is acceptable for high-level human guidance in development, a number of meta learning algorithms are needed to increase the autonomy and streamline the process of building hierarchies. These algorithms would perform such tasks as reviewing existing skills to determine if the given task requires a completely new skill or if it can be learned by duplicating an existing skill and optimizing it. They would also evaluate and select, through sensorimotor babbling for instance, the relevant input and output to use in creating a new skill. These two examples of meta learning algorithms face new challenges with the addition of the *variable module*: which existing variables should be part of the inputs and outputs of a new skill, and should an established concept be reused or a new variable created?

Finally, there remain concerns over the ability of MIND and the *Influence* mechanism to handle deep hierarchies, and the accuracy requirements for skills to avoid noisy motor output. As *variable modules* are also recipients for skill commands, this effect might manifest itself there also. The *Target* Variable scenario (Section 5.2) is an example of application requiring precise values (even though these requirements were induced by our training method, as discussed before). This issue is closely related to the effect of *Influence* on a *simple variable* discussed in Section 4.3.1, should it become necessary to address it, we would follow up on the proposed solutions, using approaches inspired by signal processing and robust training methods involving the *drive module*.

7. Conclusions and future works

In this article, we identified the desirable properties of a system supporting internal representations for the lifelong development of agents, out of a wide range of memory systems and representation mechanisms, from low-level memory to symbols. Adhering to the developmental approach of reaching complex representation by building upon simpler ones, we proposed a mechanism for general purpose non-symbolic representation, discussed its place within the lifelong development process, and formulated a proposition for its integration within an architecture. We then explained the principle of MIND, the architecture for lifelong development used as testbed, summed up the previous experimental results on reactive behaviours, before introducing the *variable module* with several of its possible implementations and applications. Through experiments, we have demonstrated the use of *variable modules*. With skills able to share information, we have shown alternative ways to structure hierarchies, which generalizes behaviour through shared concepts (Section 5.2), and that the close coupling with the skills and learning process favours the emergence of non-symbolic representations (Section 5.3). By providing a mechanism to manage internal representations we addressed one of the limitations of the architecture and allow the development of agents to continue beyond simple reactive behaviour (Section 5.3).

Beyond the validation of our proposition, the positive experimental results are an encouraging first step in the investigation of emergent memory representations. We are confident it makes an excellent testbed for new research on evolving low-level cognitive functions from the ground up, and its connectionist and developmental approach gives a promising alternative to symbolic Artificial Intelligence. Specifically, the mechanism of encoding high dimensional states or intermediate elements of the working memory into a highly compressed representation which in turn can be processed into higher level representations offers a grounded and emergent way to reach symbolic representation, suitable to general purpose Artificial Intelligence, and high-level reasoning.

In the formation of such general purpose long term memory elements, social interaction will certainly play an important role. The individual experience and its synthetization process will lead to the generation of many symbols. Through exchange of such symbols in a community bound to encounter similar experiences, their validity can be cross-checked, the symbols can be refined or rejected, merged or simplified, and their expression standardized. This process constitutes a vastly distributed experimental machine where the concept can go through much more testing and refinement, with a much wider range of possible conditions than would be possible in the lifetime of a single agent. Symbols surviving this process are adopted by the majority as a culture and form the basis of communication.

Keeping in mind the relation between emergent representations and social interactions, we want to focus our efforts on the development of multiagent behaviours. Our next immediate objective is the study of developmental agents relying on internal states to learn cooperative tasks in a society of agents. Based on our work with models of social specialization, using reinforcement to form an internal representation [88,70], we will investigate the possibility of learning the reinforcement behaviour itself: In addition to learning how to find the appropriate roles and their distribution within a group on a species level, each agent will use this mechanism to learn its own role within a society during its lifetime.

Additional experiments need to be conducted on the use of variables, such as increasing the number of variables involved in a behaviour, the complexity of the hierarchies and tasks to perform. The process of converting perceptual information into representations leads to perceptually grounded symbols as explained by Steels [10], but with the possibilities of structuration offered by variables, what about the conversion of deliberative and behavioural processes into representations? To use once more the image of psychological “drives”, skills could output information on their work process simulating “satisfaction” or “frustration” to inform motivational systems and meta behaviours. Using a dual internal function associating the controller with a predictor, skills could output their “surprise” [16] at unexpected results. Such meta information has many potential applications, from autonomous learning through intrinsic motivation [14,16] to multiagent coordination [54].

Scaffolding behaviour through the drive module is a step towards the integration of the training process within the control architecture, which so far was limited to the manipulation of skill hierarchies from without. The same approach could be used to increase the autonomy of development, by integrating processes such as IMAGINE [17] to generate new skills through intrinsically motivated goal discovery. We are particularly interested in the natural language (NL) interactions and descriptive social partner (SP) approach of IMAGINE, especially since it seems that direct human guidance can be realistically used in the role of the SP. By manipulating elements of the hierarchies, such approach would help in identification, naming procedures (skills), perceptions (sensors) and declarative knowledge (variable), and exploit permutations of elements in the language to explore new combinations of procedures and objects, considered as specific concepts though variables. In such context, the specificity of MIND would introduce the idea of achieving simultaneous tasks in the natural language descriptions. For instance, a description of the goal reached by *GoToObject+Avoid* could be: “you fetched the object while avoiding collisions at the same time”.

In addition to the autonomous and emergent aspects of agent development, there is a real need to study the possibilities of building control systems designed around variables for human control and interactions in an adjustable and semi-autonomous operation context. The use of the *drive module* to control variables, modify *Influence* or even alter hierarchies and dynamically load libraries of skill modules would offer a very flexible control system for drones, UAVs or human-assistance robotic systems. In this context,

lifelong development architectures would ease the addition of new skills and goals, of new components and external agents, which would help in supporting the lifetime evolution of such complex projects.

Tangentially related to our work, the failure of neural networks to learn the function associated to the task in Section 5.2 led to questions about the generalization of the *Influence* mechanism to neural networks. Following the No Free Lunch Theorem, neural networks were considerably improved by making assumption on the nature of the data they process. For instance, convolutional neural networks processing images use a topology which exploits the spatial proximity of pixels. The requirements for the function in Section 5.2 are very similar to the kind of problems our architecture was designed for: one higher level decision inhibits or lets through lower level signals without altering them. Since the inputs, outputs and the *Influence* are all signals by nature, this structure could be generalized as a neural network topology with the addition of a simple operator dynamically adjusting the weights of connections. Naturally the proposition of an Influence Neural Network would have to be accompanied by corresponding alterations to popular learning algorithms.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

As mentioned in the article, all research data and code is available in the following public repositories: https://gricad-gitlab.univ-grenoble-alpes.fr/surof/evoagents2020 or https://gite.lirmm.fr/suro/evoagents2020

Acknowledgements

This work has been realized with the support of the High Performance Computing Platform HPC@LR, financed by the Occitanie / Pyrénées-Méditerranée Region, Montpellier Mediterranean Metropole and the University of Montpellier, France.

Appendix A. Handcrafted network for collectVariable

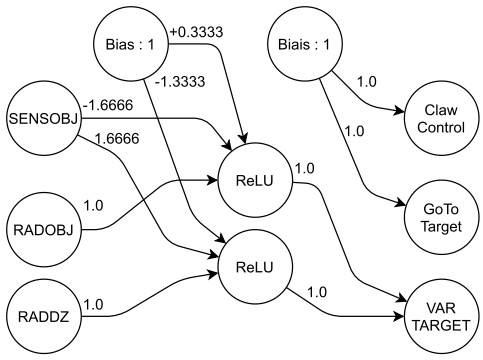


Fig. A.26. The optimal solution for the *CollectVariable* skill using a multilayer perceptron. Connections not shown have their weights set to 0. SENSOBJ gives the value of 0.8 if an object is carried by the agent and 0.2 if not. This configuration is able to perform exclusion by shifting the excluded value of the orientation sensors to negative and using a ReLU to bring the excluded value to 0, having thus no effect on the final sum. It is however very specific and its neighbouring solutions have very poor performance which is not suited to our learning algorithm.

Appendix B. Quantitative results for *Target* variable

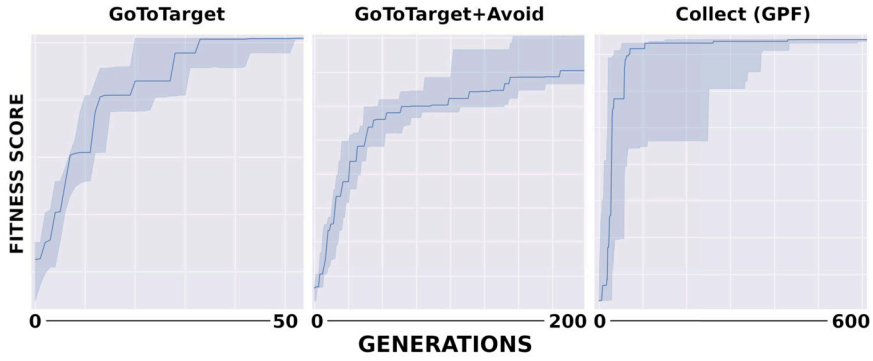


Fig. B.27. Evolution of the fitness score during training for the skills of the *Target* variable experiments (Section 5.2) for 10 replications using random initial seeds. The *Collect* skill using *Genetic Program Forests* reaches its optimal score around 100 generations. The greater variation in scores for *GoToTarget+Avoid* is due to the greater importance of random elements in the training scenario (obstacle generation and random target placement).

Appendix C. Applications for the drive module

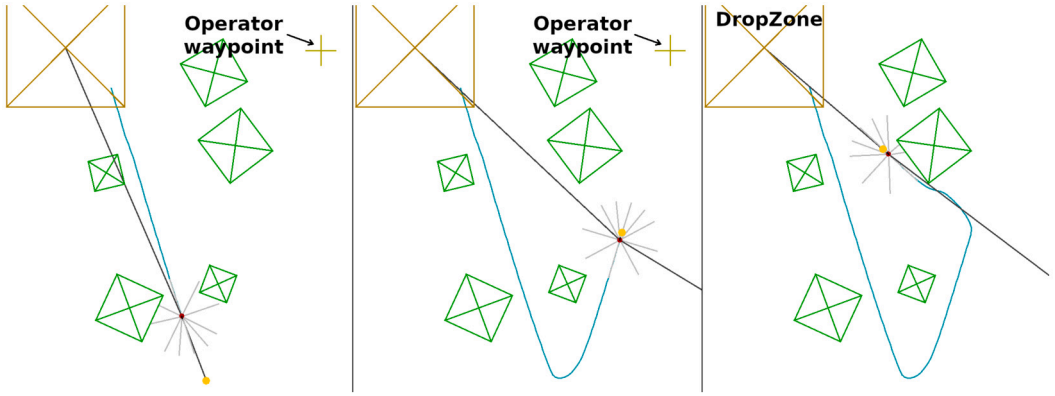


Fig. C.28. An example of use for exploitation purposes: here the *drive module* takes operator input in real time and overrides variables to set alternate targets for delivering an object. The agent will only take operator input into account if it is carrying an object to deliver. If the waypoint is removed by the operator, the agent resumes its fully autonomous behaviour, setting its own values for variables through the hierarchy.

The *drive module* enables control of the agent through the manipulation of *Influence* signals and variable values, to set both exploitation goal and learning motivation. In its simplest form, it encapsulates a programming procedure which has access to all the elements of a MIND hierarchy.

The *drive module* can be used in the context of learning, to simplify the process of shaping or scaffolding behaviour. From a development standpoint, this can be understood as creating drives for play. The arbitrary object of a game has no meaning to the agent on its own, but the skills learned from the pursuit of this goal can be transferred to other situations, when the agent learns to set its own goals. In future iterations of the architecture, the *drive module* could be used to autonomously guide the agent's development, placing it in a learning mode, hinting corrections to existing behaviour, and generating artificial goals to pursue in order to acquire new skills, through mechanisms simulating intrinsic motivation [14].

Another example of application to learning is in improving the robustness of skill training. The *drive module* can be used to send parasitic commands and noise to actuators during learning, to deal with the sensitivity issues of the *Influence* mechanism, which answers the open question on the limits of MIND hierarchies discussed previously in Suro et al. [11].

For exploitation purposes, it can be used to set mission objectives and parameters, altering behaviour through variables representing its drives and selecting the appropriate *master skill* by manipulating the *Influence* signal. It can also provide interactivity and offer an adjustable level of control, from fully autonomous to programmed or teleoperated, in a way that integrates with the principles of MIND. An example of application to adjustable autonomy is shown in Fig. C.28, where an agent performing a collect task can be given alternate drop zone coordinates in real time by a human operator. Human input is interpreted in a semi-autonomous fashion: the orientation of the alternate goal is given to the agent, but the avoidance of obstacles is still done autonomously. If the alternate goal is removed by the operator, the agent resumes fully autonomous operation. A demonstration video of this behaviour is available

```
1  public class DM_OPERATOR_dzOverride extends DriveModule{
2    VariableModule targetVar;
3    SensorModule objSens;
4    SkillModule claw;
5    SkillModule gtt;
6    SimEnv2DViewer view;
7    BotBody2D bot;
8    /... skip initialisation code .../
9    public void doStep() {
10     if(objSens.getValue() > 0.5 && view.getMarker()!= null){
11       targetVar.setReadOnly(true);
12       double relOrient = getRelativeOrient(bot, view.getMarker());
13       targetVar.overrideValue(relOrient);
14     }
15     else
16       targetVar.setReadOnly(false);
17     claw.transmitInfluenceCommand(1.0,1.0);
18     gtt.transmitInfluenceCommand(1.0,1.0);
19   }
20   /... skip management code .../
21 }
```

Listing 1: Abridged program for the drive module shown in Fig. C.28. L.12 computes the relative orientation of the operator marker given by the simulation viewer GUI. This value overrides the value of the target at l.13. Unlike the override procedure, L. 17&18 show examples of sending a regular Influence command which will be integrated with other commands exchanged in the hierarchy. Both skill receive a command of value 1.0, with a cumulative Influence of 1.0.

[75]. Listing 1 shows the corresponding implementation of this drive module. Here, the operator input is given through a GUI element (l.6&12), other means could be used such as NetworkSockets or various I/O libraries for peripherals.

References

[1] R. Caruana, Multitask learning, Mach. Learn. 28 (1997) 41–75.
[2] M.E. Taylor, P. Stone, Transfer learning for reinforcement learning domains: a survey, J. Mach. Learn. Res. 10 (2009).
[3] I. Bonnici, A. Gouaïch, F. Michel, Input addition and deletion in reinforcement: towards protean learning, Auton. Agents Multi-Agent Syst. 36 (2022) 1–34.
[4] M. Lungarella, G. Metta, R. Pfeifer, G. Sandini, Developmental robotics: a survey, Connect. Sci. 15 (2003) 151–190.
[5] P.-Y. Oudeyer, Developmental robotics, in: Encyclopedia of the Sciences of Learning, Springer, 2012, pp. 969–972.
[6] C. Prince, N. Helder, G. Hollich, Ongoing emergence: a core concept in epigenetic robotics, in: Proceedings of the Fifth International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems, 2005.
[7] E. Oztop, E. Ugur, Lifelong robot learning, in: Encyclopedia of Robotics, Springer Berlin Heidelberg, 2020, pp. 1–12.
[8] J. Ferber, O. Gutknecht, F. Michel, From agents to organizations: an organizational view of multi-agent systems, in: P. Giorgini, J.P. Müller, J. Odell (Eds.), Agent-Oriented Software Engineering IV, in: Lecture Notes in Computer Science, vol. 2935, Springer, Berlin Heidelberg, 2004, pp. 214–230.
[9] T. Stratulat, J. Ferber, J. Tranier, MASQ: towards an integral approach to interaction, in: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2, AAMAS '09, International Foundation for Autonomous Agents and Multiagent Systems, 2009, pp. 813–820.
[10] L. Steels, The symbol grounding problem has been solved, so what's next, in: Symbols and Embodiment: Debates on Meaning and Cognition, 2008, pp. 223–244.
[11] F. Suro, J. Ferber, T. Stratulat, F. Michel, A hierarchical representation of behaviour supporting open ended development and progressive learning for artificial agents, Auton. Robots (2021).
[12] J. Mao, C. Gan, P. Kohli, J.B. Tenenbaum, J. Wu, The neuro-symbolic concept learner: interpreting scenes, words, and sentences from natural supervision, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, 2019.
[13] G. Konidaris, L.P. Kaelbling, T. Lozano-Perez, From skills to symbols: learning symbolic representations for abstract high-level planning, J. Artif. Intell. Res. 61 (2018) 215–289.
[14] P.-Y. Oudeyer, F. Kaplan, What is intrinsic motivation? A typology of computational approaches, Front. Neurorobot. 1 (2007).
[15] T. Hester, P. Stone, Intrinsically motivated model learning for developing curious robots, Artif. Intell. 247 (2017) 170–186.
[16] S. Blaes, M. Vlastelica Pogančić, J. Zhu, G. Martius, Control what you can: intrinsically motivated task-planning agent, Adv. Neural Inf. Process. Syst. 32 (2019).
[17] C. Colas, T. Karch, N. Lair, J.-M. Dussoux, C. Moulin-Frier, P. Dominey, P.-Y. Oudeyer, Language as a cognitive tool to imagine goals in curiosity driven exploration, Adv. Neural Inf. Process. Syst. 33 (2020) 3761–3774.
[18] S. Forestier, R. Portelas, Y. Mollard, P.-Y. Oudeyer, Intrinsically motivated goal exploration processes with automatic curriculum learning, J. Mach. Learn. Res. 23 (2022) 6818–6858.
[19] D. Lessin, D. Fussell, R. Miikkulainen, Open-ended behavioral complexity for evolved virtual creatures, in: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, 2013, pp. 335–342.
[20] G.E. Hinton, R.S. Zemel, Autoencoders, minimum description length, and Helmholtz free energy, Adv. Neural Inf. Process. Syst. 6 (1994) 3–10.
[21] P. Stone, M. Veloso, Layered learning, in: European Conference on Machine Learning, Springer, 2000, pp. 369–381.
[22] C. Devin, A. Gupta, T. Darrell, P. Abbeel, S. Levine, Learning modular neural network policies for multi-task and multi-robot transfer, in: IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2017, pp. 2169–2176.
[23] B.C. Da Silva, G. Konidaris, A.G. Barto, Learning parameterized skills, in: Proceedings of the 29th International Conference on International Conference on Machine Learning, ICML'12, Omnipress, Madison, WI, USA, 2012, pp. 1443–1450.
[24] M. Dorigo, M. Colombetti, Robot shaping: developing autonomous agents through learning, Artif. Intell. 71 (1994) 321–370.
[25] J.A. Becerra, A. Romero, F. Bellas, R.J. Duro, Motivational engine and long-term memory coupling within a cognitive architecture for lifelong open-ended learning, Neurocomputing 452 (2021) 341–354.
[26] M. Rucinski, A. Cangelosi, T. Belpaeme, An embodied developmental robotic model of interactions between numbers and space, in: Proceedings of the Annual Meeting of the Cognitive Science Society, vol. 33, 2011, pp. 237–242.

[27] M. Ruciński, A. Cangelosi, T. Belpaeme, Robotic model of the contribution of gesture to learning to count, in: IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL), IEEE, 2012, pp. 1–6.
[28] P. Gärdenfors, Induction, conceptual spaces and AI, Philos. Sci. 57 (1990) 78–95.
[29] T. Ten Berge, R. Van Hezewijk, Procedural and declarative knowledge: an evolutionary perspective, Theory Psychol. 9 (1999) 605–624.
[30] J.T. Connor, R.D. Martin, L.E. Atlas, Recurrent neural networks and robust time series prediction, IEEE Trans. Neural Netw. 5 (1994) 240–254.
[31] T. Ziemke, et al., Remembering how to behave: recurrent neural networks for adaptive robot behavior, Recurrent neural networks, in: Recurrent Neural Networks: Design and Applications, 1999, pp. 355–389.
[32] S.D. Whitehead, L.-J. Lin, Reinforcement learning of non-Markov decision processes, Artif. Intell. 73 (1995) 271–306.
[33] A.R. Vandenbroucke, I.G. Sligte, A.B. Barrett, A.K. Seth, J.J. Fahrenfort, V.A. Lamme, Accurate metacognition for visual sensory memory representations, Psychol. Sci. 25 (2014) 861–873.
[34] J.L. Elman, Finding structure in time, Cogn. Sci. 14 (1990) 179–211.
[35] M. Lukoševičius, H. Jaeger, B. Schrauwen, Reservoir computing trends, Künstl. Intell. 26 (2012) 365–371.
[36] A. Rodan, P. Tino, Minimum complexity echo state network, IEEE Trans. Neural Netw. 22 (2010) 131–144.
[37] W. Maass, Liquid state machines: motivation, theory, and applications, in: Computability in Context: Computation and Logic in the Real World, 2011, pp. 275–296.
[38] M. Lukoševičius, H. Jaeger, Reservoir computing approaches to recurrent neural network training, Comput. Sci. Rev. 3 (2009) 127–149.
[39] C.L. Hull, Principles of Behavior: An Introduction to Behavior Theory, Appleton-Century (Still Paywalled), 1943.
[40] D. Choi, P. Langley, Evolution of the ICARUS cognitive architecture, Cogn. Syst. Res. 48 (2018) 25–38.
[41] N. Nilsson, Teleo-reactive programs for agent control, J. Artif. Intell. Res. 1 (1993) 139–158.
[42] J.L. Morales, P. Sánchez, D. Alonso, A systematic literature review of the teleo-reactive paradigm, Artif. Intell. Rev. 42 (2014) 945–964.
[43] J.E. Laird, Extending the Soar cognitive architecture, Front. Artif. Intell. Appl. 171 (2008) 224.
[44] A. Nuxoll, J.E. Laird, Extending cognitive architecture with episodic memory, in: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada, AAAI Press, 2007, pp. 1560–1564.
[45] A.E. Martin, L.A. Doumas, Predicate learning in neural systems: using oscillations to discover latent structure, Curr. Opin. Behav. Sci. 29 (2019) 77–83.
[46] P.H. Chang, K.-T. Chen, Y.-H. Chien, E. Kao, V.-W. Soo, From reality to mind: a cognitive middle layer of environment concepts for believable agents, in: D. Weyns, H.V.D. Parunak, F. Michel (Eds.), Environments for Multi-Agent Systems, First International Workshop, E4MAS 2004, Revised Selected Papers, in: LNAI, vol. 3374, Springer, 2005, pp. 57–73.
[47] K. Yi, J. Wu, C. Gan, A. Torralba, P. Kohli, J. Tenenbaum, Neural-symbolic vqa: disentangling reasoning from vision and language understanding, Adv. Neural Inf. Process. Syst. 31 (2018).
[48] L. Steels, T. Belpaeme, et al., Coordinating perceptually grounded categories through language: a case study for colour, Behav. Brain Sci. 28 (2005) 469–488.
[49] L. Steels, A self-organizing spatial vocabulary, Artif. Life 2 (1995) 319–332.
[50] I. Krupnik, W. Weyapuk, Qanuq ilitaavut: “How we learned what we know”, in: Wales Inupiaq Sea Ice Dictionary, 2010, pp. 321–354.
[51] R.A. Brooks, A robust layered control system for a mobile robot, IEEE J. Robot. Autom. 2 (1986) 14–23.
[52] T. Larsen, S.T. Hansen, Evolving composite robot behaviour-a modular architecture, in: Proceedings of the Fifth International Workshop on Robot Motion and Control, RoMoCo’05, IEEE, 2005, pp. 271–276.
[53] R.C. Arkin, T. Balch, AuRA: principles and practice in review, J. Exp. Theor. Artif. Intell. 9 (1997) 175–189.
[54] O. Simonin, J. Ferber, Modeling self satisfaction and altruism to handle action selection and reactive cooperation, in: Proceedings of the 6th International Conference on the Simulation of Adaptive Behavior, vol. 2, 2000, pp. 314–323.
[55] N. Heess, G. Wayne, Y. Tassa, T.P. Lillicrap, M.A. Riedmiller, D. Silver, Learning and Transfer of Modulated Locomotor Controllers, arXiv, 2016.
[56] S. Levine, P. Abbeel, Learning neural network policies with guided policy search under unknown dynamics, in: Advances in Neural Information Processing Systems, 2014, pp. 1071–1079.
[57] I.F. Pérez, A. Boumaza, F. Charpillet, Learning collaborative foraging in a swarm of robots using embodied evolution, in: Artificial Life Conference Proceedings 14, MIT Press, 2017, pp. 162–169.
[58] K.O. Stanley, R. Miikkulainen, Competitive coevolution through evolutionary complexification, J. Artif. Intell. Res. 21 (2004) 63–100.
[59] C.W. Reynolds, Flocks, Herds and Schools: A Distributed Behavioral Model, ACM SIGGRAPH Computer Graphics, vol. 21, ACM, 1987, pp. 25–34.
[60] V.G. Santucci, G. Baldassarre, M. Mirolli, Grail: a goal-discovering robotic architecture for intrinsically-motivated learning, IEEE Trans. Cogn. Dev. Syst. 8 (2016) 214–231.
[61] J.M. Porta, E. Celaya, Reinforcement learning for agents with many sensors and actuators acting in categorizable environments, J. Artif. Intell. Res. 23 (2005) 79–122.
[62] S. Whiteson, N. Kohl, R. Miikkulainen, P. Stone, Evolving soccer keepaway players through task decomposition, Mach. Learn. 59 (2005) 5–30.
[63] F. Suro, J. Ferber, T. Stratulat, F. Michel, MIND: base skills demo, https://hal.archives-ouvertes.fr/hal-02572019v1, 2017.
[64] F. Suro, J. Ferber, T. Stratulat, F. Michel, MIND: complex skills demo, https://hal.archives-ouvertes.fr/hal-02572031v1, 2017.
[65] F. Suro, J. Ferber, T. Stratulat, F. Michel, MIND: power management demo, https://hal.archives-ouvertes.fr/hal-02572023v1, 2017.
[66] F. Suro, J. Ferber, T. Stratulat, F. Michel, MIND: robotic application, https://hal.archives-ouvertes.fr/hal-02594407v1, 2020.
[67] V.G. Santucci, G. Baldassarre, E. Cartoni, Autonomous reinforcement learning of multiple interrelated tasks, in: 2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob), IEEE, 2019, pp. 221–227.
[68] A. Romero, G. Baldassarre, R.J. Duro, V.G. Santucci, Analysing autonomous open-ended learning of skills with different interdependent subgoals in robots, in: 2021 20th International Conference on Advanced Robotics (ICAR), 2021.
[69] F. Suro, J. Ferber, T. Stratulat, F. Michel, Émergence de comportements collectifs basée sur l’apprentissage progressif individuel, in: N. Sabouret (Ed.), Architectures multi-agents pour la simulation de systeèmes complexes - Vingt-huitième journées francophones sur les systèmes multi-agents, JFSMA, Cépaduès, 2020.
[70] F. Suro, J. Ferber, T. Stratulat, CogLogo: une implémentation de MetaCiv pour NetLogo (démonstration), in: O. Simonin, S. Combettes (Eds.), Systèmes Multi-Agents et simulation - Vingt-septièmes journées francophones sur les systèmes multi-agents, JFSMA, Cépaduès, 2019, pp. 183–184.
[71] K. Sims, Evolving virtual creatures, in: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, 1994, pp. 15–22.
[72] M. Mouad, L. Adouane, P. Schmitt, D. Khadraoui, P. Martinet, Architecture controlling multi-robot system using multi-agent based coordination approach, in: 8th International Conference on Informatics in Control, Automation and Robotics, ICINCO, 2011.
[73] R. Alami, R. Chatila, S. Fleury, M. Ghallab, F. Ingrand, An architecture for autonomy, Int. J. Robot. Res. 17 (1998) 315–337.
[74] N. Côté, A. Canu, M. Bouzid, A.-I. Mouaddib, Humans-robots sliding collaboration control in complex environments with adjustable autonomy, in: IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, vol. 2, IEEE, 2012, pp. 146–153.
[75] F. Suro, J. Ferber, T. Stratulat, F. Michel, MIND: drive module demo, https://hal.archives-ouvertes.fr/hal-03311387, 2021.
[76] J. Heaton, Encog, https://www.heatonresearch.com/encog/, 2015.
[77] D. Murphy, JBox2D, http://www.jbox2d.org/, 2014.
[78] M. Dvorak, JBullet, http://jbullet.advel.cz/, 2008.
[79] F. Suro, EvoAgents public repository, https://gite.lirmm.fr/suro/evoagents2020, 2020–2022.
[80] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed., Prentice Hall Press, 2009.

[81] R. Poli, W. Langdon, N. Mcphee, A Field Guide to Genetic Programming, Creative Commons, 2008.
[82] Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, in: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, 2009, pp. 41–48.
[83] S. Narvekar, J. Sinapov, M. Leonetti, P. Stone, Source task creation for curriculum learning, in: International Conference on Autonomous Agents & Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems, 2016, pp. 566–574.
[84] F. Suro, J. Ferber, T. Stratulat, F. Michel, MIND: taget variable demo, https://hal.archives-ouvertes.fr/hal-02924783, 2020.
[85] F. Suro, J. Ferber, T. Stratulat, F. Michel, MIND: taget variable demo - genetic program version, https://hal.archives-ouvertes.fr/hal-03637119, 2021.
[86] A. Cangelosi, M. Schlesinger, Growing babies and robots, in: Developmental Robotics: From Babies to Robots, The MIT Press, 2015, pp. 1–18.
[87] F. Suro, J. Ferber, T. Stratulat, F. Michel, MIND: count variable demo, https://hal.archives-ouvertes.fr/hal-02950608, 2020.
[88] J. Ferber, J. Nigon, G. Maillé, T. Stratulat, MetaCiv: un framework multi-agent basé sur MASQ pour modéliser des sociétés humaines, in: R. Courdier, J.-P. Jamont (Eds.), Principe de Parcimonie - Vingt-deuxièmes Journées Francophones sur les Systèmes Multi-Agents, JFSMA, Cepadues Editions, 2014, pp. 87–96.