A hierarchical representation of behaviour supporting open ended development and progressive learning for artificial agents

François Suro, Jacques Ferber, Tiberiu Stratulat, Fabien Michel

Noname manuscript No.
(will be inserted by the editor)

A hierarchical representation of behaviour supporting open ended development and progressive learning for artificial agents

François Suro · Jacques Ferber · Tiberiu Stratulat · Fabien Michel

the date of receipt and acceptance should be inserted later

Keywords Developmental robotics · Learning agents · Artificial behaviour · Modular architecture · Hierarchical architecture · Curriculum learning

Abstract One of the challenging aspects of open ended or lifelong agent development is that the final behaviour for which an agent is trained at a given moment can be an element for the future creation of one, or even several, behaviours of greater complexity, whose purpose cannot be anticipated.

In this paper, we present MIND (Modular Influence Network Design), an artificial agent control architecture suited to open ended and cumulative learning. The MIND architecture encapsulates sub behaviours into modules and combines them into a hierarchy reflecting the modular and hierarchical nature of complex tasks. Compared to similar research, the main original aspect of MIND is the multi layered hierarchy using a generic control signal, the influence, to obtain an efficient global behaviour.

This article shows the ability of MIND to learn a curriculum of independent didactic tasks of increasing complexity covering different aspects of a desired behaviour. In so doing we demonstrate the contributions of MIND to open-ended development: encapsulation into modules allows for the preservation and re-usability of all the skills acquired during the curriculum and their focused retraining, the modular structure serves the evolving topology by easing the coordination of new sensors, actuators and heterogeneous learning structures.

LIRMM - Laboratoire d'Informatique, Robotique et Microélectronique de Montpellier - Université de Montpellier - CNRS
161 Rue Ada, 34090 Montpellier, France
E-mail: suro@lirmm.fr, ferber@lirmm.fr, stratulat@lirmm.fr, fmichel@lirmm.fr

1 Introduction

Jean Piaget's (Piaget, 1954; Piaget and Duckworth, 1970) theory of cognitive development as a dynamical process of coordination schemes through multiple stages (from sensory-motor schemas to abstract level operations) is a starting point for every research on epigenetic, or developmental, robotics (Lungarella et al., 2003). Piaget's main idea is that learning is done progressively through interaction between the child and his environment, more complex tasks being learned on top of simpler tasks. This idea influenced the field of developmental robotics, where it is fundamental that the complexity of knowledge and skills acquired by an artificial agent increases progressively (Oudeyer, 2012).

According to Piaget, the complexity of learning can evolve along three axes:

1. Environment can progressively be more and more complex, from sand box to real world situations.
2. Motivations can be more and more complex, from grasping an object to building a house.
3. Skills and their structuring into behaviour should develop to handle the increase in complexity of both environment and motivations.

From early Renaissance pioneers identifying areas of the human brain linked to specific functions to more recent works on coordination between these areas, research shows that biological systems use a modular approach (Felleman and Van Essen, 1991). Different areas of the brain are dedicated to specific processes, and organized in modules to accomplish tasks. Works on the primate visual cortex aimed at creating a bridge between biology and computer vision have pointed out the hierarchical structure of the brain and the need for

hierarchical design in computer vision (Kruger et al., 2013).

If computer vision systems can benefit from hierarchical structures to identify objects in a world that is "spatially laid out and structured hierarchically.[1]", and since we are inclined to describe processes and tasks in hierarchies of procedures, whether we call it a recipe or an algorithm, could not a learning system benefit as well from having each of its subskills represented in its own module that coordinates with other modules hierarchically to accomplish complex tasks?

In this article we aim to provide a control architecture for artificial agents that is particularly suited to learning from a curriculum carefully designed by an instructor. A curriculum is defined as a set of independent tasks to perform, each task having an environment, a goal, and rewards or motivation mechanisms. The various tasks of a curriculum are ordered by increasing complexity, latter tasks may depend on the knowledge acquired from previous tasks. This set of tasks will cover various aspects of a desired behaviour, and their mastery will lead to the mastery of the target behaviour. In the perspective of open ended or lifelong development, a crucial importance is placed on the fact that the current desired final behaviour will be an element for the future creation of one, or even several, behaviours of greater complexity, whose purpose cannot be anticipated. Unlike many works using a monolithic, or synthetical black-box approach such as those presented in section 2.2, we build upon modular design principles and propose the Modular Influence Network Design architecture, an architecture which is suited to progressive and curriculum learning. This architecture reflects the modular properties of the curriculum by creating corresponding modules that can encapsulate many forms of learning structures (neural networks, function approximator, etc.). These modules will be provided with a way to coordinate with others to accomplish complex tasks. The goal of this architecture is to give an agent the ability to accumulate new skills continuously and to coordinate previously acquired skills to quickly master new tasks of increasing complexity.

This article is organized as follows: an overview of related works is given in section 2, section 3 describes the MIND architecture and section 4 describes the experimental setup of the three experiments we conducted to evaluate the MIND architecture: learning a complex behaviour from scratch (section 5), improving a behaviour (section 6), learning a new complex behaviour making use of a previous behaviour (section 7). Before concluding, we discuss preliminary experiments on a real world robot (section 8) and potential limitations (section 9).

[1] Objects can be naturally split into parts and sub-parts, complex features and simple features (Kruger et al., 2013)

2 Background

In this section we give a definition of a few terms to help the comparison between works in the field of curriculum learning and skill-based architectures. We will then take a detailed look at the Robot Shaping techniques (Dorigo and Colombetti, 1994, 1998) that encompasses all the desired aspects of developmental agents, bringing together learning techniques and structure. We will compare it to other techniques and recent works that also use curriculum learning to shape a hierarchy of skills.

2.1 Terminology

A *task* is an objective to be accomplished for an agent situated in an environment. If the task is intended to train an agent, it must include a motivation system, a feedback signal or a fitness function.

A *Curriculum* is a series of independent and increasingly complex training tasks, or lessons, covering the different aspects of a desired behaviour.

A *behaviour* is the expression of a skill, a capability, whose purpose is to perform an action related to the task.

A *skill*, or ability, is a memorized element, acquired through experience or training, that is expressed in the form of a behaviour.

A *Base skill* or sensor-motor skill is a skill that directly associates sensors with actuators, and represents the lowest level of decision, reflexes. When several base skills control the same actuator simultaneously, competition occurs.

A *Complex skill* coordinates other skills to express a more complex behaviour than its subskills. The complex skill delegates to its subskills the realization of the actions by combining, prioritizing, and arbitrating according to circumstances.

2.2 Curriculum learning

Curriculum learning (Bengio et al., 2009) is a progressive learning method investigated to this day in the field of machine learning as a way to accelerate learning and simplify learning problems that are otherwise very complex.

Learning requires a feedback, either from the environment directly or through a teaching entity, but when the learning task and environment are too complex, providing a meaningful feedback signal from all the constraints and objectives becomes very difficult. For instance, consider the case of accumulating different reward sources for conflicting behaviours, the different values of the rewards interfere with each other and it is not possible to tell which action should be rewarded without the teaching entity's analysis of the context.

Many recent works are aimed at improving the teaching entity(Lopes and Oudeyer, 2012), using for instance the metaphor of motivation (Oudeyer and Kaplan, 2007). *External motivation* describes simple feedback from the environment, *internal motivation* generates the feedback from the environment through the agent's perspective, finally *intrinsic motivation* is able to plan a learning strategy, selecting the appropriate feedback source and deciding what to learn (Santucci et al., 2016, 2019; Forestier et al., 2017). Within intrinsic motivation, several concepts are investigated such as curiosity (Blaes et al., 2019) or novelty (Barto et al., 2004; Hester and Stone, 2017).

Another approach to the problem is to learn each behaviour with its own feedback as a curriculum, and then learn how to combine these well established behaviours. Instead of using a complex teaching entity, a curriculum is handcrafted using simple feedback (*External motivation*). Creating separate tasks greatly simplifies the process of designing learning environments, reduces the cost in supervision during training and also helps in the exploratory aspect of learning, focusing on the additional complexity of the new environment associated with the task.

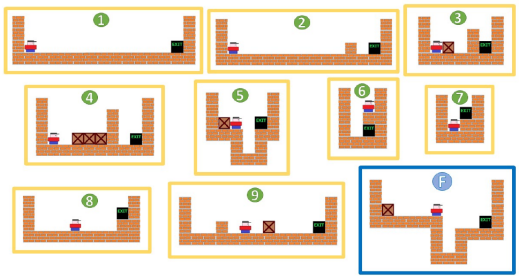

Fig. 1: The final skill F (framed in blue) is learned by transferring all previously learned skills, such as reaching the exit (1), jumping on a block (2), pushing a block (3) ... (from Foglino et al. (2019)).

Curriculum learning subdivides a learning task into different but complementary subtasks (or *source tasks*) to be learned separately, and has been applied successfully to robotics and video games problems (Narvekar et al., 2016). In this work, the various source tasks are memorized by a single function approximator through the use of transfer learning, a set of machine learning techniques designed to re-purpose a model trained on one task to a second related task.

Curriculum learning has also been applied in a more abstract context to teach neural networks to approximate functions (Gülçehre et al., 2016). In this work, the idea is to train the network to match a simplified version of the function and progressively change this target function to match the actual function we want to approximate. This method uses elements of curriculum learning and continuation, smoothing techniques, in which more complexity is added to the same learning task instead of being a collection of complementary learning tasks. In this work, the knowledge is also represented as a single module, the neural network.

Although these methods allow learning by progressive means, they are focused on a specific task designed beforehand. They are able to improve and specialize to reach their goal, however, due to the use of monolithic structures they have limited abilities to diversify their behaviour. They do not cover the lifelong, open ended, cumulative learning of a variety of tasks, a challenge at the heart of developmental robotics that is characterized by the acquisition of skills and knowledge progressively (Oudeyer, 2012).

While curriculum learning facilitates the training of monolithic structures on a complex task, its application to lifelong and open-ended development calls for a specifically designed architecture. To suit the cumulative process of development, the acquisition of a variety of skills serving the emergent needs of the agent, each new task should be assigned to an individual skill module. Such a modular principle will require a way to coordinate the potentially contradictory outputs of the different skill modules. The following section gives an overview of some existing skill-based architectures.

2.3 Skill-based architectures

Previous works in the field of control systems for robots such as the Subsumption architecture (Brooks, 1986) offer solutions for alternating the use of different skills depending on the context. Skills are arranged in a hierarchy: the lower levels perform the most basic functions, such as avoiding an immediate collision and the higher level skill perform functions of an increasing complexity, such as exploration. The higher level skill can either perform its own function or let its

subordinate perform its function. For instance, the skill *explore* starts heading to a distant place, if the agent meets an obstacle, *explore* lets *avoid*, its subordinate skill, take control. As soon as the obstacle is out of the way, *explore* takes the control back and continues on its way to its objective. In the Subsumption architectures skills are mutually exclusive, and thus have complete motor control.

When conflicting or concurrent basic behaviours each exhibit parts of a desired behaviour at a given moment, arbitrating between them seems an interesting solution. For instance, in works that study the flocking and heard behaviour using virtual birds (Boids, Reynolds (1987)), a controller solves the problem of "arbitrating independent behaviours" by combining, prioritizing, and arbitrating between potentially conflicting behaviour.

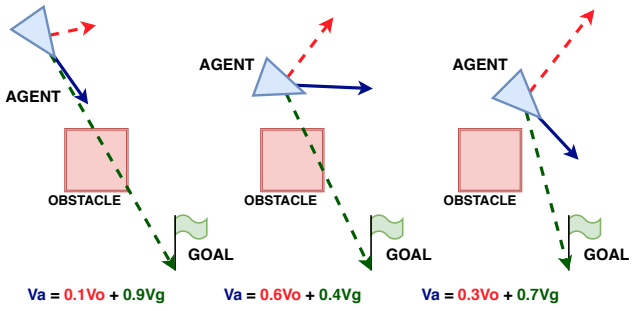

Fig. 2: Vector composition: the direction of the agent (Va) is determined from an obstacle avoiding (Vo) and goal reaching (Vg) behaviours, depending on the context.

The mechanism of output vector composition shown in figure 2 uses a weighted average of the output of each skill. The weights attributed to each skill varies according to the situation. This mechanism is able to choose between alternating or combining the actions of separate skills, giving a weight of 0 to all skills but one would result in the exclusive use of that skill.

Vector composition has seen a limited use in works such as AuRA (Arkin and Balch, 1997) or Sat-Alt (Simonin and Ferber, 2000), but the scale of these architectures remains limited and were not intended for deep hierarchies.

The idea of the coordination of concurrent behaviours (Reynolds, 1987) was also used in complex locomotion problems (Heess et al., 2016). Low level behaviour elements, qualified as "spinal", are in charge of learning sensory-motor primitives, these behaviours are then coordinated by high level "cortical" elements which drive behaviour by modulating the inputs to the spinal network. This distinction emphasizes the time scale difference, allocating more resources for fast sensor acquisition of the "spinal" level. This approach covers a single agent sensory-motor development, to a contemporary standard of complexity (up to a 54-dimensional humanoid), but still does not go beyond a two level hierarchy.

2.4 Learning hierarchies

The idea of building a hierarchy whose elements are trained separately by a different task of the curriculum, and given responsibility for different functions of a complex task has long been investigated. In Layered Learning (Stone and Veloso, 2000) a single layer percepton is trained on a low level task, the output of the trained layer is used as input for a new layer which is trained on a higher level task. This method could be viewed as training a multi-layered perceptron one layer at a time. This is not unlike the structure of Convolutional Neural Networks(CNN)(Krizhevsky et al., 2012) where low level kernels are in charge of simple shape recognition and fed to the next convolutional layer. In Devin et al. (2017) a CNN is trained on a sensorimotor task in such a way that it can be cut in the middle, the input side is referred to as the "task" module and the output side as the "robot" module (body). This method makes different modules interchangeable, allowing one robot module to perform different tasks or one task module to use different robot bodies.

Such methods operate by successively refining the input for a higher level decision, and are not skill based in the sense of combining concurrent behaviours.

A work that encompasses all the desired aspects of developmental agents is the Robot Shaping techniques developed by Dorigo and Colombetti (Dorigo and Colombetti, 1994, 1998). In Robot Shaping, behaviours are learned as a curriculum and represented as individual skills, these skills are then combined to achieve higher level behaviours. The articles cover both the architectural and didactic aspect: Multiple architectures are discussed, from monolithic to multi-level hierarchies, learning methods and reward methods tailored towards artificial agents are proposed that constitute an excellent starting point for our work.

When examining detailed points of the Robot Shaping Hierarchies themselves (noted hereafter RSH) we find that they could be supplemented and improved by recent technical works, and additional concepts.

In RSH the lower level skills are the only ones with access to sensor data and send requests for action to

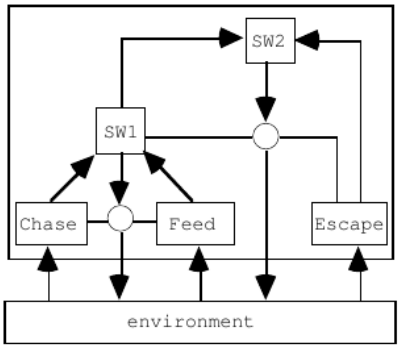

Fig. 3: *An example of three-level switch architecture for the Chase/Feed/Escape behavior. Besides the three basic behaviours can be seen the two switches, SW1 and SW2.* From Dorigo and Colombetti (1994)

the higher level skills (coordinators). Based solely on these requests, the higher level skill chooses which and how subskills should be coordinated. This was improved in other works (Larsen and Hansen, 2005; Niël and Wiering, 2018) by giving the higher level skills direct access to sensor data, including data from sensors not involved with the subskills.

This is a crucial point because dealing with how to solve a problem involves synthesizing, distorting and discarding a part of the information or signal. The low level skill *escape* (of the *chase/feed/escape* hierarchy presented in figure 3) which solves the problem of escaping a predator does not need to know if the predator is an immediate threat or keep track of other priorities to perform its task. Information as to *why*, *when* and *if* is discarded, leading to the same response when subtle details in the context would call for an entirely different approach to succeed. For instance, the information about the current distance of the predator and the importance of hunger, which are not used by *escape*, could influence the choice between continuing to feed a while longer or fleeing immediately.

Another limitation of RSH is its use of binary strings as outputs for its skills. While it has a low resource cost, it does not allow for vector composition in the Boids sense (Reynolds, 1987), but only sequential and exclusive skill use. This is also the case in Niël and Wiering (2018) which despite using real number outputs of neural networks still chooses exclusively based on the highest value (much like the classification use of neural networks).

Vector composition is integrated to the hierarchical architecture in works on open-ended evolution of virtual creatures (Lessin et al., 2013, 2015). As the name implies, this approach lets low level signal oriented components evolve to fit a task, the resulting organisation solidifies into a skill which can then be used for combination. This contrasts with RSH declaration of skills whose controller are then trained to perform tasks, coming from a more supervised "shaping" philosophy. Both methods are faced with the problem of coordinating subskills, in the evolving virtual creatures, commands are transmitted as signal and combined using various operators (including the "pandemonium" which is used where mutual exclusion is required), RSH also uses a predetermined set of combination operators from which the coordinator chooses the most appropriate. Still, this set of combination operators has to be designed in advance and hand crafted, which is a limitation for open ended development.

The alternative we propose dispenses the use of predetermined operators by using real numbers as skill outputs to serve both vector composition and combination operation mechanisms. Any particular combination can be learned for each specific case and is the responsibility of the higher level skill.

Using a single type of output, real numbers, for commands and coordination will allow any combination between the elements of the architecture. Unlike RSH which makes a clear distinction between the base skills (that output motor commands) and the coordinators (that output coordination commands), the use of a unified communication method between modules makes this distinction only conceptual. In our approach, a complex skill is able control base skills and actuators at the same time.

Extending this principle of using real numbers for communication to the input signals will open a number of possibilities. Among the further research suggested by RSH was mentioned finding a way to deal with memory systems. Input and output sharing the same numerical representation will help the integration of memory modules, in a way comparable to the method of dealing with time series with recurrent neural networks (Connor et al., 1994; Lukoševičius and Jaeger, 2009).

Understanding control as a signal influencing behaviour, from low level sensory-motor actions all the way to abstract level operations, lead us to design a unified mechanism suited to the developmental process and emergent intelligent behaviour.

3 Modular Influence Network Design

Our method relies on mirroring the modularity of progressive learning in a modular architecture. While other solutions focus on improving the learning process

of a single task, we designed our architecture using a representation of knowledge suited to the accumulation of a great diversity of skills. This cumulative process will over time improve the learning of subsequent tasks, by providing many possible starting points for the new behaviours to learn. MIND (Modular Influence Network Design) is a hierarchy of modules able to coordinate separate behaviours, or skills, to accomplish a complex task. The benefits of this approach reside in the following properties:

1. **Encapsulation**: generic behaviours will be encapsulated and combined with others in various ways to achieve different goals, rather than specialize a global behaviour to a specific task and losing the ability to branch from the original generic behaviour.
2. **Identifiable behaviours**: MIND gives the ability to identify and modify behaviour locally, working on a single aspect at a time.
3. **Unifying methods**: MIND places no constraints on the decision method used by each module, except for the input and output domains. This enables MIND to use neural networks, programming procedures and various other functions in the same network. MIND provides a way for the modules to organize with each other as a network, driven by influence.
4. **Flexibility of MIND**: behaviour modules can be replaced either by a new module or a hierarchy of modules favouring constant evolution of the system.
5. **Flexibility of body**: MIND provides a generic solution to the organization of sensors and actuators. The method used to coordinate related sensors and actuators into local groups is also used to coordinate all the groups in the system together. This also means new sensors and actuators can easily coordinate with an already existing system without losing previously acquired behaviours.

3.1 Base skill, complex skill, and influence

In the following we consider an agent whose sensory information and motor commands are represented as vectors of real numbers, normalized between 0 and 1. It is possible to create a module that encapsulates a function $f(x)$ that reads the input vector $V_I = [I_1, I_2, ..., I_n]$ and outputs the 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. Using the definitions of section 2.1, such a module is called a *skill*, and the module whose output vector is used directly as motor commands a *base 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.

Braitenberg vehicles (Braitenberg, 1986), are examples of agents that directly associate an input vector of analog signals to a similar output vector. A MIND agent could use a single *base skill* to represent the wiring of a Braitenberg vehicle.

Using a single *base skill* provided with all the sensory inputs and all the motor outputs of the agent would be sufficient to learn how to perform a complex task, each lesson of the curriculum being memorized in the same unique structure. This monolithic skill would be the sum of all the different experiences, with no way to differentiate what has been taught.

Instead of performing a complex task by a single skill, the complex task can be divided into subtasks, some even conflicting, to be performed by separate *base skills*. Each *base skill* only associates the inputs and outputs necessary to accomplish its designated task. To perform the complex task, a *complex skill* is created which will coordinate several *skills*, that we call its *subskills* (Figure 4). A *complex skill* accomplishes coordination by sending to its *subskills* a signal called *influence* which determines how much weight (influence) a *subskill* has on the resulting action. This can be understood as delegating to one or a combination of *subskills* the resolution of the current task in the same fashion as the Boid brain coordinates its sub behaviours to accomplish the task of flocking (Reynolds, 1987).

A *Complex skill*, as any skill, encapsulates a function that takes an input vector from the sensors V_I and outputs a vector of real numbers V_O, but the output is directed to its subskills. This output vector is called the influence vector $V_{Infl} = [Infl_1, Infl_2, ..., Infl_m]$, and its elements $Infl_x$ are called *influences*.

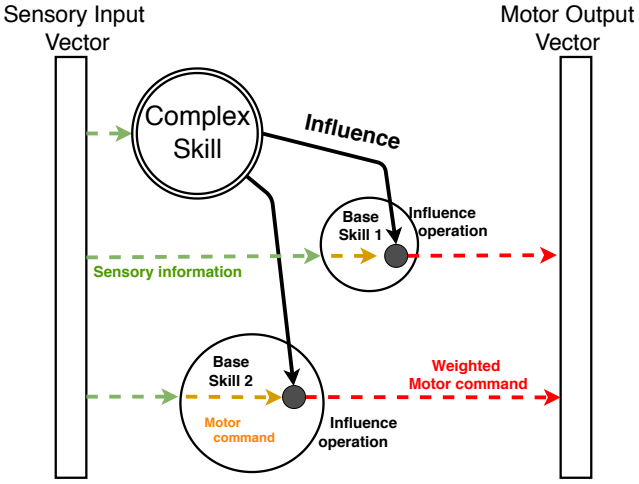

Fig. 4: A *complex skill* influencing two *base skills*.

A complex skill can have other complex skills as its subskills, thus creating hierarchies of skills. At the top of the hierarchy is the *master skill*, a complex skill whose only particularity is to receive a constant influence of 1.0, an impulse setting the whole process in motion.

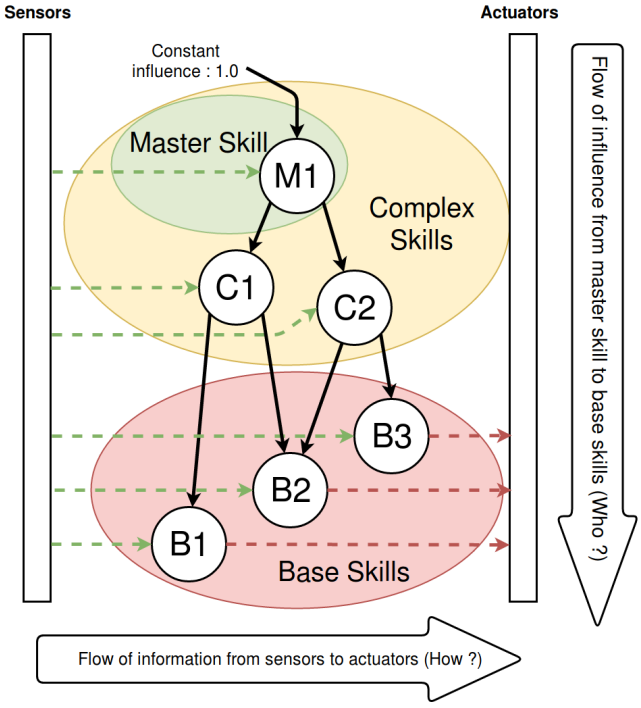

Fig. 5: A skill hierarchy, a *master skill* influences *complex skills* which in turn influence the *base skills*.

This hierarchy of skills forms a Directed Acyclic Graph (Figure 5). The influence flows along a vertical axis from the master skill down to the base skills and determines *who* (and with which magnitude) is in charge of the resulting action. 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 resulting action is going to be executed.

Figures 4 and 5 show that sensory inputs are available to every skill, including complex skills. This enables a complex skill to perform subtle coordination based on information that is not needed by the subskills.

3.2 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 (figure 6).

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

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 $\sum_{c \mapsto s=1}^{Cs \mapsto s} Infl_{c \mapsto s}$ the sum of all influences the skill received (also noted $\Sigma Infl$).

The base skill, like any other 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 module then computes the corresponding motor command for its actuator as a normalized weighted sum:

Equation 4 gives the complete computation of a motor command from the master skill to the actuator in a three level hierarchy

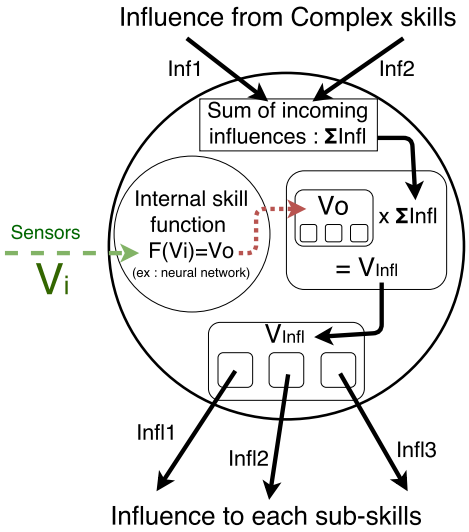

Fig. 6: Internal architecture of a skill.

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

Eq.3: With M 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.

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

Eq.4: With M the resulting final motor command, $Bs \mapsto_M$ 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, $F(Vi) \mapsto_X$ the element directed to X of the output vector of the skill internal function F processing the input vector Vi.

4 Experimental setup

To prove the effectiveness of the proposed architecture, we experimented in a simulator with a reactive agent, a (simulated) robot, using neural networks as internal functions trained by a simple genetic algorithm. The experiments will involve the task of collecting an object, which we divided into the following hierarchy of skills:

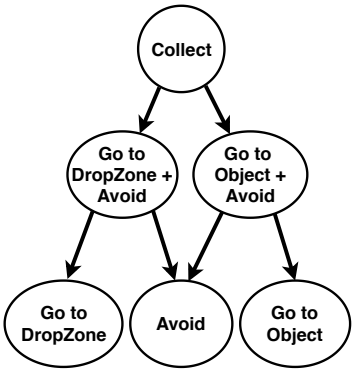

Fig. 7: An overview of the division into skills of the Collect task.

We planned the experiments as follows:

- **Scenario 1: Curriculum learning.** We build a hierarchy of skills of increasing complexity, from reaching a target and simple avoidance to collecting and bringing back objects in an environment with obstacles.
- **Scenario 2: Focused retraining.** We then observe the effects of the curriculum and learning strategies, and the limitations of hand crafted reward functions. We also experiment with focused skill retraining and learning in broader context strategies for an established MIND hierarchy.
- **Scenario 3: Flexibility.** Finally, we add new skills to the already trained MIND hierarchy to demonstrate its flexibility and show its merits as an architecture for open ended and cumulative learning.

Videos of the results are available at the following addresses:

Base skills.
`www.lirmm.fr/~suro/videos/BaseSkills.mp4`
Complex skills:
`www.lirmm.fr/~suro/videos/ComplexSkills.mp4`
Collect:
`www.lirmm.fr/~suro/videos/CollectPS.mp4`

4.1 Skill internal function and the learning algorithm

The skill internal functions we use are neural networks, using a genetic algorithm *as* a function optimizer (De Jong (1992), Rudolph (1994)).

The internal function is implemented as multi-layered perceptron, for which the input layer

corresponds to the input vector of the skill and the output layer to its output vector. Depending on the skill, its neural network will use 2 or 3 hidden layers of N_{HIDDEN} neurons, with:

$$N_{HIDDEN} = Max(N_{Vi}, N_{Vo}) + 2 \quad (5)$$

Eq.5: With N_{Vi} and N_{Vo} the number of neurons on the input and output layers respectively.

The transfer function of the layers of the neural network is sigmoid, except for the last layer which uses a linear transfer function that is clamped between 0 and 1. This configuration is more suited to our problem which is closer to function approximation than classification.

We acknowledge that this configuration has a high convergence time for the genetic algorithm we use but it is generic enough to cover any kind of skill.

Unlike the traditional non-hierarchical approach which uses a unique skill with a single neural network that connects all sensors and actuators, in our hierarchical approach, each skill has its own neural network using only the appropriate actuators, sensors or subskills.

To learn the tasks, we used a simple genetic algorithm (Russell and Norvig, 2009). We choose this solution for its simplicity, exploratory properties and good performance with delayed rewards. By nature, there is no need to provide it with a description of how to achieve a goal (unlike methods which use Backpropagation that requires an input-output training set) but only with a way to measure if the performance of an individual is better or worse than the performance of other individuals. Evaluating the quality of the behaviour at the end of the life cycle of an agent allows it to persist in a penalizing behaviour if it eventually leads to a greater reward, thus circumventing the issue of delayed reward.

One of the major drawbacks of the genetic algorithm is its high computational cost, specifically the evaluation of the fitness function which requires running each individual, i.e. agent, in a simulated environment for a sufficient period of time. However, as the evaluation of each agent is completely independent, it can be run in parallel and allows the use of High Performance Computing solutions. Since each skill uses a small network we do not suffer from much communication overhead, as a result the real time evaluation of an entire generation is significantly reduced.

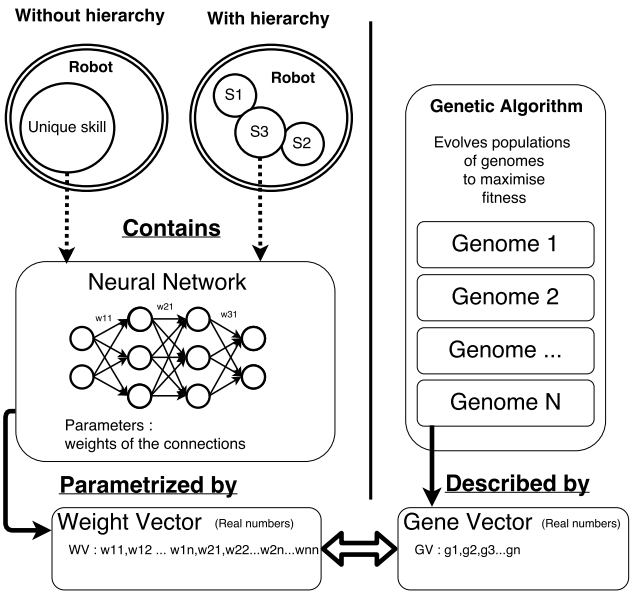

Fig. 8: Relation between the skills and the genetic algorithm.

To use a genetic algorithm on a neural network, the weights of the neural network's connections are ordered in a vector, which corresponds to the genome of the individual from the point of view of the genetic algorithm. The genome will be bred and mutated according to the value of the fitness function the environment provides as feedback. The relation between the skills and the genetic algorithm is illustrated in Figure 8.

4.2 The robot and simulation environment

The simulated robot is composed of two motors, one for each wheel, and a claw to grab the target object. It also has 18 sensors:

- 10 obstacle sensors (range finders) placed around the robot.
- 3 sensors giving respectively the orientation of the target object, the drop zone and the power supply (a zone to recharge the batteries).
- a sensor indicating if the target object is in range of the claw.
- 2 sensors indicating respectively if the robot is inside the drop zone and inside the power supply zone.
- a sensor indicating the remaining charge of the batteries.
- a simple inertial sensor indicating if any movement occurred.

The robot's software, through what we call a *Sensor Module*, is able to compute the derivative of any sensor

input and provide the result as a virtual sensor. It can also generate a sinusoidal wave that can be used in the same way (useful, in conjunction with the inertial sensor, to solve deadlock situations).

The motor command, issued by the *Motor Module* to its actuator as a real number between 0 and 1, is interpreted as a percentage of the actuator's capability (e.g. power, speed, angular position, etc.). In the case of the wheels, 1 corresponds to full speed forward, 0 to full speed backwards and 0.5 to stopped. In the case of the claw, the number corresponds to its position (treated as binary): above 0.5 is closed, under 0.5 is opened.

The simulation environment is based on a 2D physics engine, the JBox2d physics library[2]. This allows us to be closer to real world constrains, using thrust vectors and inertia simulation instead of discrete actions steps. It will also make the benefits of MIND and its influence mechanism obvious through the composition of thrust vectors in a continuous environment.

Figure 9 shows the environments for the *GoToObject*, *GoToDropZone*, *Avoid* and *Collect* with power management tasks. The red circle represents our robot, green squares and borders represent obstacles. The long white lines connecting the robot to the targets represents the orientation sensors. The shorter white lines represent the obstacle detection sensors. A full line means that the sensor has not collided with an obstacle within its range. A shorter line means the sensor has detected an obstacle and will give its distance as a fraction of the maximum range.

Notice the difference in size between the object and the drop zone, grabbing the object will require finer motor control, the target being smaller. Also, grabbing the object requires to align the front of the robot (where the claw is mounted) with the object.

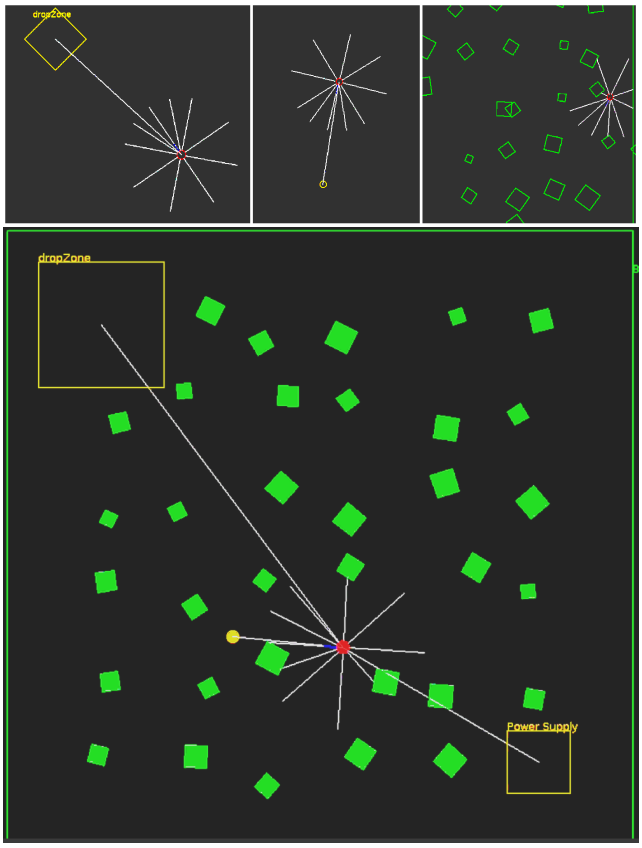

Fig. 9: Top: *GoToDropZone*, *GoToObject* and *Avoid* environments. Bottom: the complete training environment for Collect with power management (section 7).

4.3 Genome evaluation

To evolve a population, each genome of a generation must be evaluated and given a fitness score.

The genome to be evaluated sets the weights of a neural network using its gene vector (Fig. 8). The neural network is then placed inside the skill module corresponding to the task to learn. Finally, this skill and its instance of the MIND hierarchy take control of a simulated robot placed in an environment where it can run for a given time.

At each tick of the simulation, a set of reward functions are evaluated and their result added to the current score. At the end of the evaluation, this accumulated score will represent the fitness score of the evaluated genome.

For instance, the *GoToObject* task has two reward functions:

1. *Closing on target*: $-\Delta_{DistanceTarget} * 0.001$, a very small reward given each tick that can be negative if the distance to the target increases.
2. *Reaching target*: +0.5, a large reward given each time the robot reaches the target. The target is then moved to a new place for the robot to reach (random placement with minimum distance constraint).

The small *Closing on target* reward helps to differentiate genomes that result in an attraction behaviour towards the target from genomes that cause repulsion from the target. This helps to speed up the early stages of evolution. However, the reward remains small compared to the *Reaching target* reward to favour the outcome we desire over the process we think would lead to that outcome. It also helps in optimizing the final stages of evolution, favouring genomes that result in a sharper turn when facing the target.

[2] JBox2d: `www.jbox2d.org`

Indirectly the *Reaching target* reward combined with the time limit also leads to optimization. Genomes that reach more targets within the time limit will have better scores. This leads to sharper turns, straight trajectories and moving at top speed.

The *GoToObject* task is a simple example, and yet we can already see subtle interactions between all the constraints and rewards. When designing the *Avoid* task we met the same difficulties mentioned in the Robot Shaping experiments: the iterative process of designing the reinforcement program, the difficulty of finding an appropriate shaping policy which is at time as difficult as directly coding the control program (Dorigo and Colombetti, 1994, 1998).

In complex behaviour where it is difficult to even picture an optimal solution out of many acceptable ones, learning from a hand crafted lesson (or reinforcement program) will quickly show its limits. Scenario 2 (sec. 6) will show indeed that there is no need to tune the reward functions to get the optimal behaviour, optimization can be achieved by simpler means.

5 Scenario 1: Curriculum learning

Building a MIND hierarchy: Collect

In this first scenario, we aim to demonstrate that the MIND architecture is able to organize simple skills into complex behaviours, and is able to do this reliably even when using a simple genetic learning process.

5.1 Protocol

In this scenario the goal is to teach the robot a collecting task which consists in picking up an object and bringing it back to a drop zone without colliding with obstacles.

Each of the six skills needed to accomplish this task will be trained in 10 independent attempts. This will give us a fair sample of possible outcomes and convergence times, and allow us to draw conclusions in spite of the stochastic nature of genetic algorithms (Gen and Lin, 2007; Rudolph, 1994).

The scores given by the different sets of reward functions are used by the selection process of the genetic algorithm as a relative measure of performance. The result section will provide a benchmark for the final behaviour, however a complete evaluation and interpretation of such complex behaviours requires observation.[3]

[3] Videos of the results are available at the following addresses:
`www.lirmm.fr/~suro/videos/BaseSkills.mp4`
`www.lirmm.fr/~suro/videos/ComplexSkills.mp4`
`www.lirmm.fr/~suro/videos/CollectPS.mp4`

5.1.1 Curriculum and the MIND hierarchy

To create an initial hierarchy there are many possible ways to organize the different subskills, all of which are valid as long as they are able to learn from the curriculum.

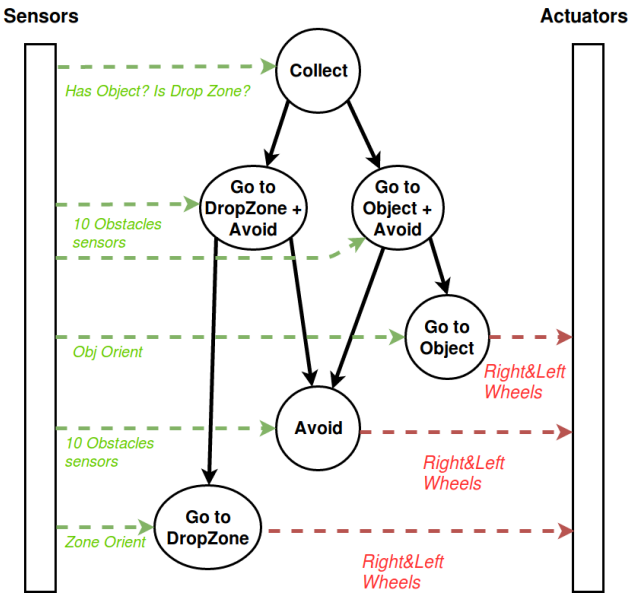

Fig. 10: The complete hierarchy for the initial collection task, with sensor and motor information shown.

We choose to divide the complex *Collect* task into a curriculum of five subtasks, three of which are base tasks to be learned by the corresponding base skills:

1. *GoToObject*: Going to the object in an empty environment. The agent receives a small reward for approaching the object and a large reward for reaching the object.
2. *GoToDropZone*: Going to the drop zone in an empty environment. The agent receives a small reward for approaching the drop zone and a large reward for reaching it.
3. *Avoid*: Avoiding collision while moving in an environment with obstacles. A collision ends the evaluation and gives the agent a large negative reward. A number of positive rewards are given (in order of importance): visiting new areas of the environment, keeping a straight path, keeping distance with obstacles and travelling forward.

On these base tasks we build two higher level complex tasks:

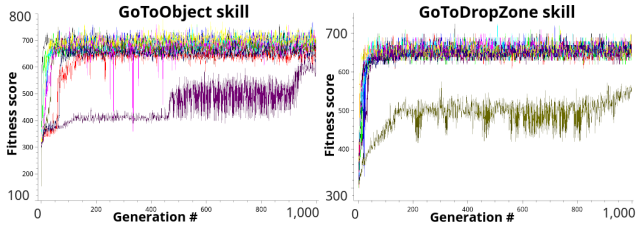

Fig. 11: *GoToObject* and *GoToDropZone* base skills: best individual score for each generation, 10 separate attempts over 1000 generations.

1. *GoToObject+Avoid*: going to the object while avoiding collision in an environment with obstacles. A collision ends the evaluation, a positive reward is given each time the agent reaches the object.
2. *GoToDropZone+Avoid*: going to the drop zone while avoiding collision in an environment with obstacles. A collision ends the evaluation, a positive reward is given each time the agent reaches the zone.

Finally, *Collect* is learned in the final environment containing an object, a drop zone and obstacles. A collision ends the evaluation, a positive reward is given for each object collected (picked up and brought to the drop zone).

Learning *Collect*, the largest complex task, is an interesting challenge. Its subtasks, *GoToObject* and *GoToDropZone*, require to use the motor functions in a completely opposite way and have to be performed sequentially and exclusively. Conversely, the *Avoid* task would be best used as a composition of vectors, by altering the set trajectory to smoothly avoid an obstacle.

5.2 Results

Figures 11 and 12 show that most attempts of the *GoToDropZone* and *GoToObject* skills reach their maximum value in 100 generations, whereas most attempts of the *Avoid* skill continue to evolve past 500 generations (the figure 12 shows the attempt over 2500 generations which we will discuss in section 6). This difference in convergence time is coherent with the complexity of the networks to train, *Avoid* has more than 10 inputs while the other base skills only 1.

We see that in both *GoToObject* and *GoToDropZone* skills, one attempt in ten did not reach a satisfactory result within the number of generations given. However, it is interesting to note that they are still improving with each generation, which is positive in the context of our work aimed at supporting open ended and lifelong development of artificial agents.

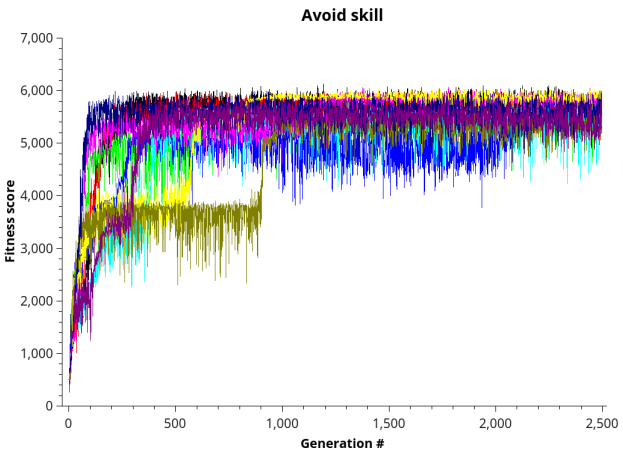

Fig. 12: *Avoid* base skill: best individual score for each generation, 10 separate attempts over 2500 generations.

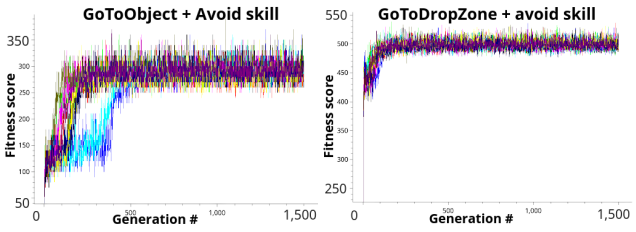

Fig. 13: *GoToObject+Avoid* and *GoToDropZone+Avoid* complex skills: best individual score for each generation, 10 separate attempts over 1500 generations.

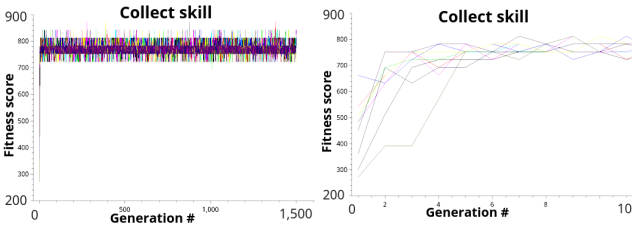

Fig. 14: *Collect* master skill: best individual score for each generation, 10 separate attempts over 1500 generations (Right: showing only the first 10 generations scores).

Setting aside the failed attempts, we selected successful base skills to learn the complex skills *GoToObject+Avoid* and *GoToDropZone+Avoid*. The *GoToObject+Avoid* is slightly more difficult to learn than *GoToDropZone+Avoid*, which can be explained by the fact that the object is smaller than the drop zone and requires more precision (the "claw" must be aligned with the object to grab it, see Sec. 4.3). Both skills still succeed on each of their respective 10 attempts within the given number of generations.

Finally, the master skill *Collect*, having a very simple network to train, succeeds on each of its 10 attempts in under 10 generations.

To illustrate the effect of influence we trained a similar hierarchy using a set of 4 constant base skills: *moveForward* always sets both wheels to full forward, *moveBackwards* always sets both wheels to full backwards, *turnLeft* and *turnRight* set one wheel to full forward and one wheel to full backwards. Using these base skills with the influence mechanism it is possible to obtain any combination of values for the wheels. Figure 15 presents our agent in a situation similar to the one used to illustrate vector composition (Fig.2) where it must avoid an obstacle to reach its goal. The right panel shows the environment and the left panel shows the state of the hierarchy. The current influence is displayed under the name of the skill, the influence links are represented by a gradient from black for the value 0 to green for the value 1, the central line represents the command of the skill alone and the outer line represents the influence actually transmitted.

The following describes and comments on the sequence shown in figure 15:

Step 1: Our agent is approaching an object to collect. The obstacle is not close enough to be considered a collision risk by *GoToObject+Avoid* thus it transmits its influence to *GoToObject* exclusively (1.0 x 1.0). *GoToObject* transmits an influence of 0.11 to *moveForward*. Since no other constant skill receives any influence, *moveForward* completely controls the output (both wheels are set to full forward).

Step 2: The agent came too close to the obstacle, *GoToObject+Avoid* transmits its influence to *Avoid* exclusively. *Avoid* transmits an influence of 1.0 to *moveForward* and *turnLeft*. In this simple case, both skills transmit a motor command of full forward to the right wheel, the commands being identical, their average result is full forward. In the case of the left wheel, *moveForward* transmits a motor command of full forward with the influence of 1.0 (weighted motor command: 1.0 x 1.0), *turnLeft* transmits a motor command of full backwards with the influence of 1.0 (weighted motor command: 0.0 x 1.0). According to the equation 3, the resulting motor command for the left wheel is:

$$\frac{(1.0 \times 1.0) + (0.0 \times 1.0)}{1.0 + 1.0} = 0.5 \qquad (6)$$

0.5 corresponds to stop. With the right wheel full forward and the left wheel stopped, the resulting behaviour is a sharp left turn centred on the left wheel.

Step 3: As the agent recovers some distance with the obstacle, the influence to *Avoid* decreases while

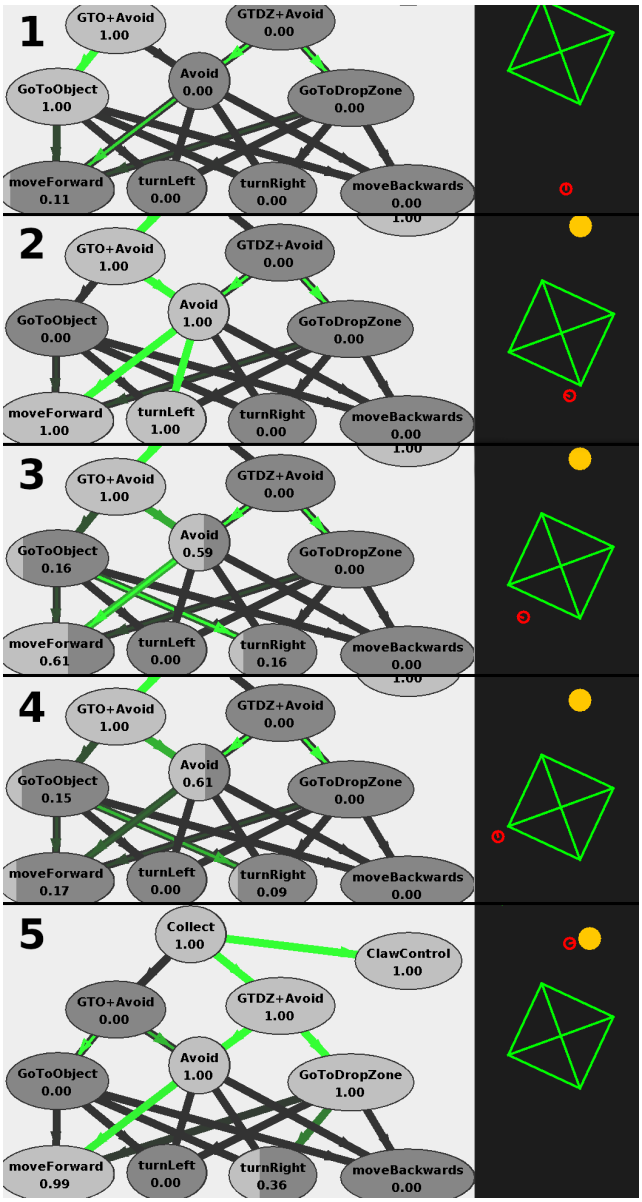

Fig. 15: Five steps of avoiding an obstacle and reaching a goal using influence.

the influence to *GoToObject* increases. *GoToObject* transmits a high influence (1.0) to *turnRight* to make a sharp turn right to reach the object. But since *GoToObject* only has a relatively small influence (0.16) *turnRight* can at most receive as much as *GoToObject* (0.16 x 1.0). Since *moveForward* receives a much higher influence, the result is a trajectory slightly curved to the right.

Step 4: With the forward obstacle sensors clear of the obstacle, *Avoid* lowers its influence to *moveForward* which increases relatively the impact of *turnRight* on the wheels (even if the influence to *turnRight*

decreased). The slight curve is now a deliberate turn around the obstacle.

Step 5: Eventually, our agent picks up the object. *Collect* sends its influence exclusively to *GoToDropZone+Avoid*. *GoToObject+Avoid* is still transmitting influence to its subskills, but since its own influence is 0.0, it has no effect. It is interesting to note that *GoToDropZone+Avoid* chooses to keep *Avoid* active at a distance from the obstacle where *GoToObject+Avoid* would have it inactive. This can be explained by the fact that *GoToObject+Avoid* may have to approach an obstacle in order to collect an object, while *GoToDropZone+Avoid* is more at risk of colliding with an obstacle when carrying the object (due to the object protruding in front).

Interruption on collision (average runtime)		
Structure	**Curriculum**	**Final task only**
Hierarchical	4.63 (54%)	0.06 (51%)
Monolithic	5.19 (68%)	0.31 (80%)

No interruption (equal runtime)		
Structure	**Curriculum**	**Final task only**
Hierarchical	9.55	0.18
Monolithic	7.70	0.39

Table 1: Top table: number of objects collected and percentage of runtime before the evaluation ends. Bottom table: number of objects collected when a collision does not end the evaluation. Results are given for our hierarchy and for a single skill monolithic structure. Each structure was trained using the curriculum and the final task only.

5.3 Analysis

Our initial MIND hierarchy succeeded in quickly learning the curriculum we designed, and reached satisfactory scores on most of our attempts. Table 1 shows benchmark scores for this Collect hierarchy against a monolithic structure learning the same curriculum. The monolithic structure is a single neural network which uses the same inputs and outputs used by our hierarchy (see Fig. 10), and 3 hidden layers, instead of the 2 used by each separate skills of the hierarchy. The monolithic structure uses 1610 parameters (i.e. neural network weights), while the sum of the parameters of all the skills of the hierarchy is 1510.

In the first benchmark (interruption on collision) 1 point is given for each object collect, any collision stops the simulation (the agent keeps its current score). The second benchmark (no interruption) does not stop in the event of a collision (but does not help the agent should it get stuck on an obstacle).

The higher score of the monolithic structure on the first benchmark is a consequence of its longer runtime. This shows that the monolithic structure is better at avoiding obstacle than our hierarchy. This can be explained by the fact that the monolithic structure can improve its avoid behaviour during the *Avoid* task, but also during both the *GoToObject+Avoid* and the *GoToDropZone+Avoid* tasks, thus benefiting from more generations and from in-context training. In our hierarchy the complex skills are able to use *Avoid*, but the training only affects the complex skills themselves and not its subskills. Compared to complete modularity which separates a complex skill from its subskills and requires each to learn its own internal function, a monolithic structure has the opportunity to share intermediate results. The structures described in Schrum and Miikkulainen (2015) show policies (subskill) and preference neurons (complex skill) sharing neural paths. We can reasonably assume that in the case of "how to avoid" (*Avoid*) and "when to avoid" (*GoToObject+Avoid*), both behaviours could share some analysis on the surrounding obstacles. This would however come at the cost of modularity.

These results raise the question of evaluating the resources needed to learn a task. The complexity of tasks can be difficult to assess, but it is easy to understand that the neural network assigned to the *GoToDropZone* skill, which uses two inputs, has far fewer connections, and thus fewer weights to adjust, than the neural network assigned to the *Avoid* skill, which uses more than 10 inputs. Figures 11 and 12 show how this difference in complexity impacts learning. Most attempts of the *GoToDropZone* and *GoToObject* skills reach their maximum value in 100 generations, whereas most attempts of the *Avoid* skill continue to evolve after 500 generations.

It is worth noting that while the complexity of the network to train has an impact on the learning time, so does the nature of the task. *Avoid* takes a longer time to learn, but all the attempts seem to reach an acceptable level of training. This can be explained by the fact that avoiding an obstacle has many solutions, all of which being valid. On the other hand *GoToObject* is represented by a much smaller network and takes less time to learn, but accomplishing this task has only two valid solutions:

1. The optimal solution: if the object is on my left, turn left; if the object is on my right, turn right.
2. The suboptimal solution: turn either always left or always right until the object is in front of me.

The suboptimal solution tends to lock the learning process in a local maximum.

The second column of table 1 shows the results of learning the whole curriculum as a single task, all reward signals cumulated in the final environment. This is of course impossible without a teaching entity analysing the context and prioritizing rewards, however the runtime scores show the agent managed to learn some form of *Avoid* behaviour. Learning as a single task using a hierarchical structure is a coevolutional approach similar to Whiteson et al. (2003), with the difference that a MIND hierarchy requires to learn the high level controllers. Without the given decision tree used in coevolution, which in effect only makes base skills evolve, no role is attributed to each skill and the decomposition of the task does not take place. For this method to succeed we would need a way to initiate the specialization of each skill, a process of reinforcement would then naturally take place.

The variation in quality of behaviours also led to the questions of whether it is relevant to continue the learning process, that is training the complex skills, using sub-optimal subskills. Would MIND be able to improve and retrain subskills after the whole hierarchy has been trained ? The second scenario was designed to study these issues.

6 Scenario 2: Focused retraining

Learning with sub-optimal subskills, retraining and learning in broader context

In this scenario we will take advantage of the property of MIND that offers identifiable behaviour to work on the aspect of the curriculum that the agent had the most trouble assimilating.

After the Scenario 1 resulted in a trained and functional hierarchy, we observed that the *Avoid* skill was causing a bottleneck for the performance of the hierarchy. This section covers retraining strategies for subskills.

6.1 Protocol

In this scenario we use the already trained hierarchy of Scenario 1 and try to improve its performance by retraining the *Avoid* skill which, by our observation of the behaviour, seems to be causing a bottleneck. We experimented with two different methods:

1. **Allocate more resources** to the original training of the *Avoid* skill and measure the performance improvement.
2. Use an alternative training method: **Learning in a broader context**. In this method we will retrain the *Avoid* skill from scratch while the agent is driven by the *Collect* skill and its hierarchy, using the final *Collect* task as the environment and reward function.

Structure	score	runtime
Monolithic	5.19	68%
Avoid 1000 gen.	4.63	54%
Avoid 2500 gen.	5.31	56%
Avoid 1500 gen.+context	6.03	90%

Table 2: Number of objects collected and percentage of runtime before failure for the different versions of the *Avoid* skill.

6.2 Results

Allocate more resources: We initially set up the learning process of each skill with 1000 generations. The resulting *Collect* skill did work, but still regularly failed due to collisions. Naturally, we considered allocating more resources to the *Avoid* task, expecting that its efficiency, and thus the efficiency of the *Collect* behaviour, would improve.

We retrained *Avoid* from scratch, increasing the number of generations from 1000 to 2500, and retrained all the higher level skills based on this new version. With 2500 generations, the final fitness score of the *Avoid* skill increased by 10% compared to the *Avoid* skill trained with 1000 generation (Fig. 12). Consequently, the *Collect* skill based on the retrained *Avoid* skill increased its final benchmark score by 14%.

This result conforms with our observation of the behaviour and indicates that the *Avoid* skill is limiting the performance of the hierarchy under the control of the *Collect* skill.

Learning in a broader context: We then used our method of learning in a broader context, using the hierarchy of scenario 1 as a starting point. The *Collect* skill was set as master skill, the final *Collect* environment used, and the reward signal came from accomplishing the *Collect* task. In this context, we retrained the *Avoid* skill from scratch (ignoring the hand crafted *Avoid* reward function in favour of the *Collect* reward function). By only using 1500 generations on the *Avoid* skill with this method, the final benchmark score of the *Collect* skill improved by 30%.

6.3 Analysis

By allocating more resources to the *Avoid* skill (more than doubling it), the overall quality of the Collect task

was only slightly improved. This leads us to question the quality of the *Avoid* learning task in the curriculum.

The choices made in the elements of the reward function are certainly in question. What we find reasonable to guide the development of a behaviour, it is the nature of evolution to exploit it to achieve the best score, regardless of the spirit of the law. For instance, in very early experiments the only reward for the *Avoid* skill was a negative one which the individual received upon colliding with an obstacle, and so the fittest individual simply did not move. Each subsequent laws added was in turn exploited until we found a set of laws which gave us a behaviour close to what was expected. This illustrates the problem of defining the reward function, which can be as difficult as simply programming the behaviour we want (Dorigo and Colombetti, 1994).

We needed an *Avoid* skill to build the hierarchy, but now that each skill has its own defined role in the hierarchy, instead of trying to improve the quality of *Avoid* on its own, we can train it in the final context of the *Collect* skill, as an element of the hierarchy.

The experimental success of the method of learning in a broader context suggests that there could be many more learning strategies to consider when teaching to a hierarchy, beyond a simple curriculum from basic to complex tasks. In the scope of our work, we found out that going back to further improve the subskills in the final context, once the whole curriculum has been roughly taught, can yield better results than trying to maximize each subskill before moving on to the next.

The initial progressive training of the hierarchy is still of great importance, even if each skill does not have to be trained to perfection, this first run through the curriculum is where each skill will specialize in its role within the hierarchy. When observing the resulting hierarchy learned without curriculum (Hierarchical/Monolithic in Tab.1) we noticed that most branches of the hierarchy were ignored and a single skill attempted to learn the complete task.

Having established each role in the hierarchy, a natural reinforcement process can take place, with the added benefit of being provided on-the-job training (or less constrained solution space). The benefits of this approach was described as "relaxing" the hierarchy in Robot Shaping (Dorigo and Colombetti, 1994) or as the Coevolution mechanism in Whiteson et al. (2003).

The ability to focus the retraining on a particular aspect of the behaviour is one of the benefits of the property of identifiable behaviours (item 2 sec. 3), resulting from a modular approach.

7 Scenario 3: Flexibility

The modularity of MIND: Collect with power management

In this scenario we demonstrate that the MIND architecture is best suited to open ended development by adding new skills, and even new sensors, to expand the abilities of an already trained hierarchy. When using MIND, the acquisition of a new skill does not alter previously acquired skills, which remain available for other combinations. Our architecture is also able to integrate new inputs, which would require a monolithic structure to alter its topology and possibly lead to its retraining from scratch. This shows another advantage of MIND as an approach for open-ended development of agents over monolithic architectures.

7.1 Protocol

In this scenario we add to the initial collection task the energy consumption and the necessity to recharge the robot's battery. A sensor providing the current power level of the battery and sensors giving information about the direction of the power source are added to the robot. This scenario adds to the previous one the base task of going to the power source to recharge and the complex task of going to the power source without colliding with obstacles, re-using the previously learned *Avoid* task.

The benchmark for this scenario is the same as that in scenario 1 with the addition of power management. The battery of the robot discharges unless the robot stands in the power source area, in which case the battery quickly recharges. The benchmark is run for a given time frame or until the robot fails (if a collision occurs or the power level of the robot reaches zero). Each time the robot brings back the object to the drop zone, it scores one point and a new object is placed randomly in the environment.

We did not include a comparison to a monolithic structure, as the change in topology occasioned by the new sensors would require us to retrain the network from scratch, using the whole curriculum. Although a monolithic structure would certainly succeed in learning this new task, the computational cost and the drawbacks of having to keep the training material available for every subsequent extension of the agent's abilities leave this method out of our consideration for developmental purposes.

Variant	Original Avoid	Retrained Avoid
Score (average runtime)		
Retrained	1.73 (28%)	3.99 (69%)
Encapsulated	1.62 (36%)	3.73 (90%)

Table 3: Number of objects collected and percentage of runtime before failure for the two variants of the hierarchy.

7.1.1 The hierarchies

When attempting to coordinate a new skill with an already existing complex skill in a hierarchy two possible ways can be considered:

1. Modify the existing complex skill so that it integrates the new one, and retrain it to fit (*Retrained variant*),
2. Create a new complex skill that coordinates the new skill with the existing complex skill. The new complex skill will be the one that undergoes the training and not the pre-existing one (*Encapsulated variant*).

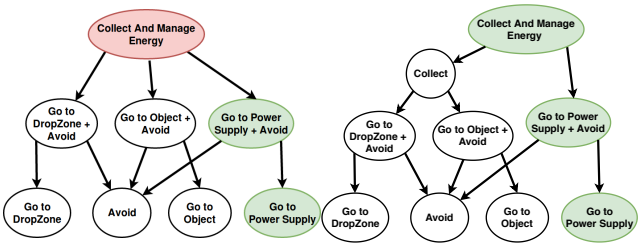

Fig. 16: The hierarchy for the collection task with energy consumption. Left: Retrained variant: The old master skill is replaced. Right: Encapsulated variant: The old master skill becomes a subskill of the new master skill.

7.2 Results and analysis

Table 3 shows that both variants of the hierarchy (*Retrained* and *Encapsulated* (fig: 16)) obtain comparable scores, using the original and the retrained Avoid skill. As pointed out before, in this benchmark the robot can fail if a collision occurs or if its power level reaches zero.

No indication of what constitutes a low battery level was given to the robot, only its current energy level. The robot determined by itself when to abandon its current collection task and head for the power source based on its own experience with its battery discharge speed and the size and complexity of the environment.

Both variants of the hierarchy are valid, however we want to point out that the *Encapsulated variant* preserves the original *Collect* skill, making it available for future combinations, which is one of the desired properties of the MIND architecture (item 1 sec. 3).

Another property of MIND that is successfully demonstrated here, by the integration of the sensors related to power management into the hierarchy, is the flexibility of body (item 5 sec. 3). The local coordination of the added physical elements with the new behaviour did not negatively impact the previously acquired behaviours. The new sensors did not require altering the existing internal functions and their topology, the existing skills remain available for future combination and will not have to be learned again.

8 Using MIND on a real world robot

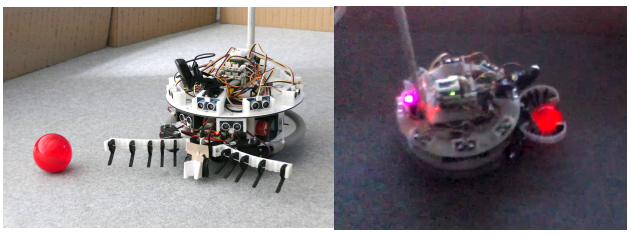

Fig. 17: Left: Our crude robot, Right: Basic object detection using a luminous ball, a webcam, OpenCV and low lights.

Given that the MIND hierarchy learned in simulation is entirely reactive we did not anticipate any problem using it on a real world application.

We designed a simple task using the two base skills *GoToObject* and *Avoid*, and the complex skill (in this case also the master skill) *GoToObject+Avoid*.

Our robot used a Raspberry PI3[4] running Linux and our Java implementation of MIND. Sensory-motor equipment was made from plug and play hobby kit elements[5] and included 10 ultrasonic range finders, a dual motor control board, a 9dof sensor, a basic switch, a servo control card to control a twin servo pan and tilt, a basic webcam, and two servos to control the claw.

The pan and tilt arm and webcam were used, with the OpenCV computer vision library[6], to find and track a luminous ball. The position of the pan arm was converted to a sensor information replicating the orientation sensor used in the simulation.

[4] Raspberry PI3: `www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/`

[5] Grove Pi: `www.dexterindustries.com/grovepi/`

[6] OpenCV computer vision library: `https://opencv.org/`

Finally, the robot was driven by two shopping trolley wheels, actuated by friction motor rollers, and the chassis was balanced by two free rotating wheels.

The only difficulty we encountered was the delay in sensor acquisition and motor response, most probably because of the hobby grade hardware being used. This problem was solved by setting appropriate sensor dead zones and maximum ranges.

Despite the noisy sensor inputs, the acquisition delays and the imprecision of the actuators, our crude robot demonstrated the expected behaviour using the hierarchy and skills learned in a simulation that was not calibrated in any way to fit the robot or real world environment.[7]

9 Limits

Our experiments have shown the ability of MIND to handle small hierarchies. However, there could be some concerns with large hierarchies, specifically deep hierarchies where the successive transfer of influence could result in a form of noisy motor output caused by several unrelated subskills receiving a very small amount of influence. In the context of our experiments, the motor tasks do not require extreme precision, and the hierarchy is not deep enough to show any hint of this phenomenon.

The reason this problem could arise is mainly due to the use of genetically trained artificial neural networks as skill internal functions giving a "good enough answer" (for a given input that should result in a left turn, if the output is 0.99 for left turn and 0.01 for right turn, the impact of right turn is not noticeable). If an influence command is transmitted from master skill to base skill with a value of 1.0 and all other influences remain at 0.0, then no matter how many complex skills are involved in the successive transfer, the final output will be the exact motor command of the base skill. What it means in practice is that if noisy outputs become noticeable, it is due to imprecisions in the neural networks that should be finely retrained (as the depth of the hierarchy increases, the acceptable imprecision of the internal functions decreases).

Future works will tell us if this concern is justified. Should noisy output become a problem and finer training of the internal function be costly and impractical, we would investigate ways to reduce noise in large networks such as filters, threshold layers, or logit functions (invert sigmoid).

[7] Videos of the results are available at the following address: `www.lirmm.fr/~suro/videos/clawDemo.mp4`

10 Conclusions

Our approach to open ended or lifelong agent development is built upon the works of authors (Whiteson et al., 2003; Narvekar et al., 2016; Gülçehre et al., 2016) who already established the advantages of progressive learning, either by guiding the learning entity through increasing the complexity of the task or by learning a curriculum of complementary tasks aimed towards the completion of a complex task. To extend this cumulative learning process to agent development, we drew inspiration from other works that departed from monolithic architectures to reflect the increasing complexity of a multi-stage curriculum into modular architectures able to coordinate simple skills into complex ones (Dorigo and Colombetti, 1994).

In this article we introduced MIND, the Modular Influence Network Design, an architecture tailored to open ended development and progressive learning for artificial agents. The MIND architecture encapsulates sub behaviours into modules and combines them using a generic control signal, the influence, to form a multi layered hierarchy reflecting the modular and hierarchical nature of complex tasks.

We have first motivated and explained the main design principles and desired features of MIND. Then, through experiments, we demonstrated MIND's ability to learn from a curriculum. The positive results obtained on the *Collect* task are encouraging, the MIND architecture was able to encapsulate each step of the curriculum to form a multi-level hierarchy of skills with both coordination and mutual exclusion of skills. MIND was able to handle both low level sensory-motor tasks and complex skill influence operations, without providing any information on the nature of the task to the internal skill function.

We demonstrated MIND's flexibility by extending the physical and behavioural abilities of an agent to meet new constrains beyond its original design. MIND handled the new sensors and actuators without incidence on the already trained skills, and by combining a few new modules with the established hierarchy, the agent was able to master a new task with minimal training.

We also experimented with techniques for the focused retraining of a module inspired from Robot Shaping (Dorigo and Colombetti, 1994). The experiment we conducted on the retraining of subskills indicates the advantage of going back and forth from simple to more complex lessons rather than trying to attain perfect results on the basic tasks before moving on to more complex ones. We are currently using this technique, along with the ability of some learning algorithms to stop and

resume training from a saved state, to optimize MIND hierarchies.

The focus of this article was to establish that MIND is an architecture suited to open ended development and progressive learning, as such we used structures capable of learning, multi-layer perceptrons, as our skills internal functions. However, the encapsulation of the internal function into a skill and the use of the influence mechanism together represents a unifying method which gives any function the ability to integrate into a MIND hierarchy. As long as a wrapper can convert inputs and outputs into vectors of real numbers, any kind of function can be used, including non learning ones. For instance, in our experiment, the function controlling the "claw" of the robot is a trivial Java procedure that has been wrapped into a base skill module.

We are currently improving the implementation of the MIND architecture, most notably by using the NEAT algorithm (Stanley and Miikkulainen, 2002) to supplement our basic genetic algorithm. This removes the need to define the neural network topology of the skill's internal function (eq.5 sec. 4.1). We are also integrating other neural network topologies (CNN, RNN) and other improved learning algorithms.

We are also studying social learning by using MIND hierarchies on a foraging task using a multi-agent system. The learning technique remains genetic, agents use an instance of the same hierarchy of skills, but the fitness score is evaluated collectively. Our preliminary results show that multi-agent coordination and emergent distribution of roles can be learned by the MAS using an already established single agent MIND hierarchy and a curriculum adding the required multi-agent skills.

As mentioned before, we are considering the suggestions for further research of Robot Shaping (Dorigo and Colombetti, 1994) on memory systems. We began adding a number of simple memory modules conforming to the MIND system of influence to give the ability to the agent to evolve beyond simple reactive behaviour.

We are currently investigating a way to teach a knowledge representation without allowing the teaching entity direct access to the memory modules. This will benefit the instructor by providing a method that is not dependent on the configuration of the agent being taught. We also hope that not violating the barrier of the interiority of the agent's mind will lead to emergent memory representations.

Beyond that, we plan to use these memory modules in a connectionist fashion on a situated agent having learning capabilities, as a test-bed for new research on evolving low level cognitive functions from the ground up. Using this developmental approach we intend to propose an alternative to symbolic AI, investigating the fundamental scientific issue of the emergence of symbols, or "symbol grounding problem", as suggested by Oudeyer (2012).

Our next step towards the creation of an artificial agent capable of open ended and lifelong development through curriculum learning under human supervision is the automation or streamlining of the different steps required by the MIND approach.

For a new lesson given by the instructor, with a well defined goal, a MIND agent should be able to determine what are the relevant sensor inputs and actuator outputs or subskills that should be used to accomplish the task. The new skill could be created from scratch, or it could be mutated from an existing module (Schrum and Miikkulainen, 2015), only changing a few inputs or outputs, or altering the behaviour.

From the instructor's point of view, what languages and strategies in formulating a lesson would help the agent understand what is expected of it while at the same time remaining natural and simple to define for the human instructor?

Acknowledgements We thank our anonymous reviewers for their many constructive comments and suggestions which greatly improve the present article.

We also thank Eric Bourreau and Marianne Huchard (LIRMM) and the members of the SMILE team (LIRMM) for their comments that helped improve our initial work.

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.

References

Arkin, R. C. and Balch, T. (1997). Aura: Principles and practice in review. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2-3):175–189.

Barto, A. G., Singh, S., and Chentanez, N. (2004). Intrinsically motivated learning of hierarchical collections of skills. In *Proceedings of the 3rd International Conference on Development and Learning*, pages 112–19.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 41–48. ACM.

Blaes, S., Pogančić, M. V., Zhu, J., and Martius, G. (2019). Control what you can: Intrinsically motivated task-planning agent. In *Advances*

in Neural Information Processing Systems, pages 12520–12531.

Braitenberg, V. (1986). *Vehicles: Experiments in synthetic psychology*. MIT press.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23.

Connor, J. T., Martin, R. D., and Atlas, L. E. (1994). Recurrent neural networks and robust time series prediction. *IEEE transactions on neural networks*, 5(2):240–254.

De Jong, K. A. (1992). Are genetic algorithms function optimizers? In *PPSN*, volume 2, pages 3–14.

Devin, C., Gupta, A., Darrell, T., Abbeel, P., and Levine, S. (2017). Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2169–2176. IEEE.

Dorigo, M. and Colombetti, M. (1994). Robot shaping: Developing autonomous agents through learning. *Artificial intelligence*, 71(2):321–370.

Dorigo, M. and Colombetti, M. (1998). *Robot shaping: an experiment in behavior engineering*. MIT press.

Felleman, D. J. and Van Essen, D. C. (1991). Distributed hierarchical processing in the primate cerebral cortex. *Cerebral cortex*, 1(1):1–47.

Foglino, F., Christakou, C. C., and Leonetti, M. (2019). An optimization framework for task sequencing in curriculum learning. In *Joint IEEE 9th International Conference ICDL-EpiRob*, pages 207–214. IEEE.

Forestier, S., Mollard, Y., and Oudeyer, P.-Y. (2017). Intrinsically motivated goal exploration processes with automatic curriculum learning. *arXiv preprint arXiv:1708.02190*.

Gen, M. and Lin, L. (2007). Genetic algorithms. *Wiley Encyclopedia of Computer Science and Engineering*, pages 1–15.

Gülçehre, Ç., Moczulski, M., Visin, F., and Bengio, Y. (2016). Mollifying networks. *CoRR*, abs/1608.04980.

Heess, N., Wayne, G., Tassa, Y., Lillicrap, T. P., Riedmiller, M. A., and Silver, D. (2016). Learning and transfer of modulated locomotor controllers. *CoRR*, abs/1610.05182.

Hester, T. and Stone, P. (2017). Intrinsically motivated model learning for developing curious robots. *Artificial Intelligence*, 247:170–186.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Kruger, N., Janssen, P., Kalkan, S., Lappe, M., Leonardis, A., Piater, J., Rodriguez-Sanchez, A. J., and Wiskott, L. (2013). Deep hierarchies in the primate visual cortex: What can we learn for computer vision? *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1847–1871.

Larsen, T. and Hansen, S. T. (2005). Evolving composite robot behaviour-a modular architecture. In *Proceedings of the Fifth International Workshop on Robot Motion and Control, 2005. RoMoCo'05.*, pages 271–276. IEEE.

Lessin, D., Fussell, D., and Miikkulainen, R. (2013). Open-ended behavioral complexity for evolved virtual creatures. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 335–342.

Lessin, D., Fussell, D., Miikkulainen, R., and Risi, S. (2015). Increasing behavioral complexity for evolved virtual creatures with the esp method. *arXiv preprint arXiv:1510.07957*.

Lopes, M. and Oudeyer, P.-Y. (2012). The strategic student approach for life-long exploration and learning. In *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, pages 1–8. IEEE.

Lukoševičius, M. and Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149.

Lungarella, M., Metta, G., Pfeifer, R., and Sandini, G. (2003). Developmental robotics: a survey. *Connection science*, 15(4):151–190.

Narvekar, S., Sinapov, J., Leonetti, M., and Stone, P. (2016). Source task creation for curriculum learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 566–574. International Foundation for Autonomous Agents and Multiagent Systems.

Niël, R. and Wiering, M. A. (2018). Hierarchical reinforcement learning for playing a dynamic dungeon crawler game. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1159–1166. IEEE.

Oudeyer, P.-Y. (2012). Developmental robotics. In *Encyclopedia of the Sciences of Learning*, pages 969–972. Springer.

Oudeyer, P.-Y. and Kaplan, F. (2007). What is intrinsic motivation? a typology of computational approaches. *Frontiers in Neurorobotics*, 1(6).

Piaget, J. (1954). *The construction of reality in the child*. Basic Books, New York.

Piaget, J. and Duckworth, E. (1970). Genetic epistemology. *American Behavioral Scientist*, 13(3):459–480.

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH computer graphics*, volume 21, pages 25–34. ACM.

Rudolph, G. (1994). Convergence analysis of canonical genetic algorithms. *IEEE transactions on neural networks*, 5(1):96–101.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

Santucci, V. G., Baldassarre, G., and Cartoni, E. (2019). Autonomous reinforcement learning of multiple interrelated tasks. In *2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 221–227. IEEE.

Santucci, V. G., Baldassarre, G., and Mirolli, M. (2016). Grail: a goal-discovering robotic architecture for intrinsically-motivated learning. *IEEE Transactions on Cognitive and Developmental Systems*, 8(3):214–231.

Schrum, J. and Miikkulainen, R. (2015). Discovering multimodal behavior in ms. pac-man through evolution of modular neural networks. *IEEE transactions on computational intelligence and AI in games*, 8(1):67–81.

Simonin, O. and Ferber, J. (2000). 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*, volume 2, pages 314–323.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.

Stone, P. and Veloso, M. (2000). Layered learning. In *European Conference on Machine Learning*, pages 369–381. Springer.

Whiteson, S., Kohl, N., Miikkulainen, R., and Stone, P. (2003). Evolving keepaway soccer players through task decomposition. In *Genetic and Evolutionary Computation Conference*, pages 356–368. Springer.

A Appendix

The following tables give the settings used in scenario 1. All values are given using the units of the physics engine (Jbox2d). These values were used to obtain the results shown in this article. However, other settings can be used with substantial improvements in convergence time and computing cost.

World data	
Time step	1/60
Velocity iterations	8
Position iterations	3
Robot	
size	2
proximity sensor range	12
angularDamping	20
linearDamping	5
max thrust per wheel	+- 80

Table 4: Setting used in scenario 1. Jbox2d engine.

GoToObject environment	
Runtime in steps	10000
Environment size	∞
Target size	3 to 1 (-0.1 per collect)
GoToObject rewards	
10 points for reaching the target 0.001Δ distance to object per tick 0.01 for forward motion per tick	
GoToDropZone environment	
Runtime in steps	10000
Environment size	∞
Zone size	5
GoToDropZone rewards	
10 points for reaching the target 0.001Δ distance to object per tick 0.01 for forward motion per tick	
Avoid environment	
Runtime in steps	10000 or collision
Environment size	100
Avoid rewards	
10 points for each new patch visited (size 4) -200 for collision 0.01 for forward motion per tick for each proximity sensor (per tick):	
If distance <1/4 max range	-0.03(1-4 range/maxrange)
If distance >1/4 max range	0.001 range/maxrange
-0.4($\Sigma\Delta$ thrust exceeding 20% over 10 ticks) per tick	
GTO + Avoid environment	
Runtime in steps	10000 or collision
Environment size	100
Target size	3 to 1 (-0.1 per collect)
GTO + Avoid rewards	
10 points for reaching the target 0.001Δ distance to object per tick	
GTDZ + Avoid environment	
Runtime in steps	10000 or collision
Environment size	100
Zone size	5
GTDZ + Avoid rewards	
10 points for reaching the target 0.001Δ distance to zone per tick	
Collect environment	
Runtime in steps	30000 or collision
Environment size	100
Zone size	10
Target size	1
Collect rewards	
30 points per collected object $0.001(+)\Delta$ or $0.0005(-)\Delta$ distance to zone per tick $0.001(+)\Delta$ or $0.0005(-)\Delta$ distance to object per tick	

Table 5: Exhaustive list of the settings and rewards of the curriculum used in scenario 1.