Darwinian embodied evolution of the learning ability for survival

Stefan Elfwing, Eiji Uchibe, Kenji Doya and Henrik I Christensen

Adaptive Behavior 2011 19: 101 originally published online 28 February 2011
DOI: 10.1177/1059712310397633

Original Paper

Adaptive Behavior

Adaptive Behavior
19(2) 101–120

DOI: 10.1177/1059712310397633
adb.sagepub.com

SAGE

Darwinian embodied evolution of the learning ability for survival

Stefan Elfwing[1,2], Eiji Uchibe[2], Kenji Doya[2] and Henrik I Christensen[1]

Abstract

In this article we propose a framework for performing embodied evolution with a limited number of robots, by utilizing time-sharing in subpopulations of virtual agents hosted in each robot. Within this framework, we explore the combination of within-generation learning of basic survival behaviors by reinforcement learning, and evolutionary adaptations over the generations of the basic behavior selection policy, the reward functions, and meta-parameters for reinforcement learning. We apply a biologically inspired selection scheme, in which there is no explicit communication of the individuals' fitness information. The individuals can only reproduce offspring by mating—a pair-wise exchange of genotypes—and the probability that an individual reproduces offspring in its own subpopulation is dependent on the individual's ''health,'' that is, energy level, at the mating occasion. We validate the proposed method by comparing it with evolution using standard centralized selection, in simulation, and by transferring the obtained solutions to hardware using two real robots.

Keywords

Embodied evolution, evolutionary robotics, reinforcement learning, meta-learning, shaping rewards, metaparameters

1 Introduction

Evolutionary robotics (Nolfi & Floreano, 2000) is a framework for automatic creation of control systems of autonomous robots, inspired by the Darwinian principle of selective reproduction of the fittest. In standard evolutionary robotics, reproduction is not integrated with the other autonomous behaviors, and the selection and execution of genetic operations are performed in a centralized manner.

Watson, Ficici, and Pollack (2002) introduced the embodied evolution methodology for evolutionary robotics. Embodied evolution was developed for the case where a large number of robots freely interact with each other while performing some task in a shared environment. The robots reproduce offspring by mating, that is, a pairwise exchange of genotypes, and, naturally, the probability for a robot to produce offspring is regulated by the robot's performance of the task. In short, embodied evolution is a methodology for evolutionary robotics that mimics the distributed, asynchronous and autonomous properties of biological evolution. The evaluation, selection, and reproduction are carried out through the interaction of the robots, without any need for human intervention.

In this article we propose a framework for performing embodied evolution with a limited number of robots. In the original embodied evolution formulation, one robot corresponds to one individual in the population. This may be the ideal case, but it makes the methodology inapplicable for most evolutionary computation tasks, because of the large number of robots that are required for an appropriate population size. In our framework, each robot contains a subpopulation of virtual agents

[1]Centre for Autonomous Systems, Numerical Analysis and Computer Science, Royal Institute of Technology (KTH), Sweden.
[2]Neural Computation Unit, Initial Research Project, Okinawa Institute of Science and Technology, JST, Japan.

Corresponding author:
Stefan Elfwing, Neural Computation Unit, Initial Research Project, Okinawa Institute of Science and Technology, JST, 12–22 Suzaki, Uruma, Okinawa 904-2234, Japan
Email: elfwing@irp.oist.jp

that are evaluated by time-sharing. Within our framework, our objective is to explore the combination of within-generation learning of basic behaviors by reinforcement learning and evolutionary adaptations over the generations of parameters that modulate the learning of the behaviors. Our method consists of three, largely independent, parts: (1) a general embodied evolution framework, where each robot contains a subpopulation of virtual agents that are evaluated by time-sharing for the survival task; (2) a two-layered control architecture, where a top-layer neural network selects learning modules, corresponding to basic survival behaviors, according to the current environmental state and the virtual agent's internal energy level; and (3) the reinforcement learning algorithm Sarsa, which is used for learning the basic behaviors during the lifetimes' of the virtual agents. The genotypes of the virtual agents consist of the weights of the top-layer neural network controller, and parameters that modulate the learning ability of the learning modules. The learning ability is evolutionarily optimized by tuning additional reward signals, implemented as potential-based shaping rewards, and global metaparameters, shared by all learning modules, such as the learning rate α, the discount factor of future rewards, γ, the trace-decay rate, λ, and the temperature, τ, controlling the trade-off between exploration and exploitation in the action selection.

Shaping rewards are a frequently used way of accelerating reinforcement learning by providing an additional and richer reward signal. In this study we have used potential-based shaping rewards as defined by Ng, Harada, and Russell (1999), which have several advantages from our perspective. The potential-based shaping rewards have a sound theoretical basis, guaranteeing convergence to the optimal policy for the original problem without shaping rewards. Their formulation does not add any extra parameters to the system and it is only dependent on the states of the agent, which makes it easy to approximate the potential function for the shaping rewards with standard function approximation methods used in reinforcement learning.

We apply an implicit and biologically inspired evolutionary selection scheme. Differential selection is achieved by virtual agents reproducing more offspring having higher probabilities of transferring their offspring to the next generation. In our selection scheme there is no explicit representation or communication of the individuals' fitness information. A virtual agent can only reproduce offspring by mating with virtual agents controlling other robots in the environment. In addition, the probability that a virtual agent reproduces offspring in its own subpopulation is dependent on the virtual agent's "health," that is, the virtual agent's internal energy level, at the mating occasion.

2 Method

Our proposed method consists of three parts (see Figure 1): (1) an embodied evolution framework that utilizes time-sharing for evaluation of subpopulations of virtual agents inside each robot; (2) a top-layer neural network controller that selects basic behaviors according to the current state; and (3) learning modules that learn basic behaviors using the reinforcement learning algorithm Sarsa(λ). The optimization of the virtual agents' overall behaviors occurs on two levels: learning of the basic behaviors by reinforcement learning within each generation, and evolutionary adaptations of the genotypes of the virtual agents over the generations. The genotype of a virtual agent controls the selection of basic behaviors, by the weights of the neural network controller. It also modulates the learning of the basic behaviors, by the additional reward function, implemented as potential-based shaping rewards, and the global metaparameters, such as the learning rate α, the discount factor of future rewards, γ, the trace-decay rate, λ, and the temperature, τ, controlling the trade-off between exploration and exploitation in softmax action selection.

Below we explain the outlines of the three parts of our method. The details of the implementations are explained in the Appendices.

2.1 Embodied evolution framework

The two main goals of our proposed embodied evolution framework (illustrated in Figure 2) are to

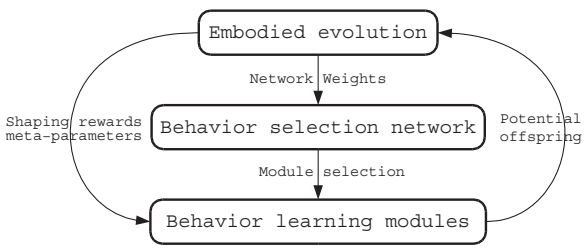

Figure 1. Overview of our proposed method, which consists of three parts: (1) an embodied evolution framework that utilizes time-sharing for evaluating subpopulation of virtual agents hosted in each robot; (2) a top-layer neural network that selects basic behaviors; and (3) learning modules that learn basic behaviors using reinforcement learning. The evolution optimizes the weights of the neural network, and the metaparameters and the shaping reward functions, which modulate the learning of the basic behaviors. To create new generations of virtual agents in the subpopulations, the embodied evolution framework uses random selection of potential offspring, which are produced in proportion to the reproductive success of the parental virtual agents, during the execution of their basic behaviors.

implement embodied evolution with a limited number of robots and to develop a selection scheme without the need for explicit communication of the individuals' fitness information. These two goals are realized by:

- Evaluation by time-sharing in subpopulations of virtual agents within each robot. The population consists of N_{sub} robots, each of which contains a subpopulation of N_{va} virtual agents. The virtual agents are evaluated, in random order, by time-sharing, that is, taking control over the robot for a limited number of T_{ts} time steps. In each generation, each virtual agent performs N_{ts} time-sharings, which gives a total lifetime of $N_{ts} \times T_{ts}$ time steps.
- Random selection of potential offspring, which are produced in proportion to the reproductive ability of the parental virtual agents. Each subpopulation (i.e., each robot) contains a list, $\boldsymbol{O}$, of potential offspring reproduced during the current generation. After all virtual agents in a subpopulation have survived for a full lifetime or died a premature death, a new subpopulation is created by randomly selecting N_{va} potential offspring from $\boldsymbol{O}$. If the number of reproduced potential offspring is less than N_{va}, then the remaining part of the new subpopulation is created by adding randomly created genotypes.

A virtual agent that controls a robot can reproduce potential offspring by performing a reproductive mating, which consists of a pair-wise exchange of genotypes with a virtual agent controlling another robot (hereafter called a successful mating) and also by satisfying a predefined reproduction condition (hereafter called a reproductive mating). For example, in our experiments the probability of reproducing potential offspring is proportional to the virtual agent's internal energy at the mating occasion. This means that the reproductive success of a virtual agent is a combination of the virtual agent's ability to perform genotype exchanges and the ability to keep its internal energy level high, by recharging from external batteries in the environment. A reproductive mating creates two potential offspring that are appended to the list of potential offspring $\boldsymbol{O}$ in the virtual agent's subpopulation, by applying genetic operations to the virtual agent's genotype and the genotype received from the mating partner. Note that our mating scheme is asymmetrical.

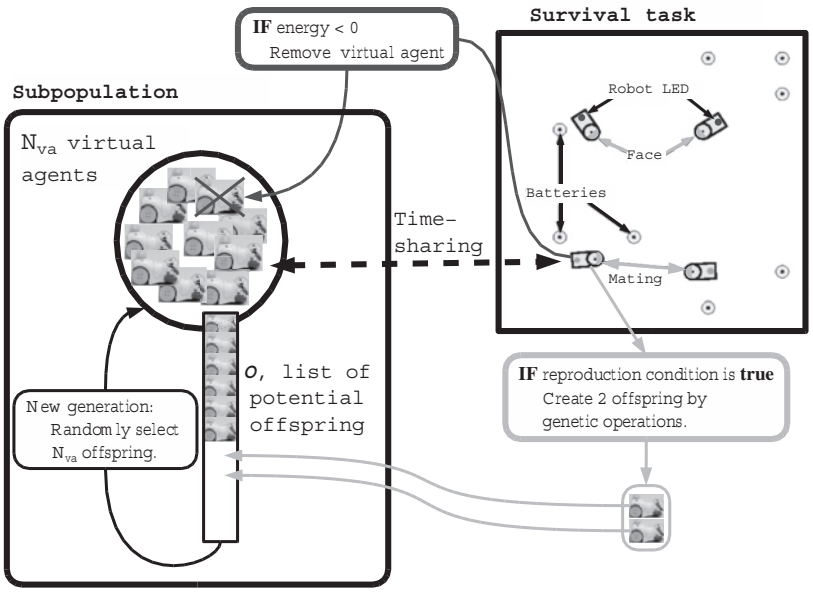

Figure 2. Overview of the embodied evolution framework. Each robot contains a subpopulation of virtual agents of size N_{va}. The virtual agents are evaluated by time-sharing for the survival task, that is, taking control over the robot for a limited period of time. If, during the evaluation, a virtual agent runs out of energy, then the agent dies and it is removed from its subpopulation. If a virtual agent performs a successful mating, that is, an exchange of genotypes with a virtual agent controlling another robot, and if a predefined reproduction condition is fulfilled, then two potential offspring are created by applying genetic operations to the virtual agent's own genotype and the genotype received from the mating partner. The two potential offspring are then inserted in the list of potential offspring, $\boldsymbol{O}$, of the subpopulation containing the virtual agent. When all virtual agents in a subpopulation have either survived for a full lifetime or died a premature death, a new subpopulation is created by randomly selecting N_{va} potential offspring from $\boldsymbol{O}$. The robot used to illustrate the virtual agents is the Cyber Rodent robot used in the hardware experiments and the picture used to visualize the survival task is a snapshot from the simulation environment used in the evolutionary experiment.

If two virtual agents perform a successful mating the following outcomes are possible depending on whether each of the two virtual agents satisfies the reproduction condition: they both reproduce potential offspring, only one of the them reproduces potential offspring, or neither of them reproduce potential offspring. The function of the reproduction condition is to achieve differential selection by determining the reproductive ability of the virtual agents. The details of the implementation of the embodied evolution framework are described in Appendix A.

The main limitation of our proposed scheme, from an evolutionary computation perspective, is that a virtual agent can only reproduce potential offspring by mating with virtual agents controlling the other robots. The number of potential mating partners is, therefore, limited by the number of robots, N_{sub}, and the number of time-sharings per generation, N_{ts}. During its lifetime, a virtual agent can mate with about $(N_{sub}-1)\cdot N_{ts}$ different mating partners. The exact number of potential mating partners for a virtual agent is dependent on the asynchronous timing of the time-sharings in the subpopulations.

2.2 Control architecture

Figure 3 shows an overview of our proposed two-layered control architecture. In each time step, the top-layer linear neural network selects one of the N_m reinforcement learning modules according to: arg max($\boldsymbol{wx}$) (line 7 in Algorithm 2 in Appendix B), where $\boldsymbol{w}$ is the neural network weights matrix (of size $N_m \times N_x$) and $\boldsymbol{x}$ is the current state input vector (of size $N_x \times 1$). The selected reinforcement learning module then executes an action based on its learned behavior. It also updates its learning parameters based on the global reward function (predefined and equal for all modules), the metaparameters (shared by all modules), and its shaping reward function (unique for each module). In relation to the embodied evolution framework, the genotypes of the virtual agents code the weights of the neural network, the metaparameters, and the parameter of the shaping reward functions. The details of the implementation of the control architecture are described in Appendix B.

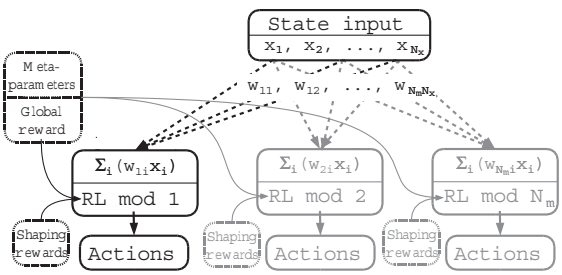

Figure 3. Overview of the two-layered control architecture. The top-layer linear feed-forward neural network uses the N_x-dimensional state space as input. In each time step, the reinforcement learning module represented by the output unit with the largest value is selected: $\arg\max_j\left(\sum_{i=1}^{N_x} w_{ji}x_i\right)$. The selected module then executes an action based on its learned behavior. The learning is modulated by the predefined global reward function, the shaping rewards, unique for each module, and the metaparameters, shared by all modules.

Our reason for choosing this type of architecture is mostly pragmatic. Reinforcement learning has proved to be an effective learning paradigm for many tasks, but the learning performance is directly related to the dimension of the state space, "*the curse of dimensionality*." The state space is therefore decomposed, and the reinforcement learning modules learn their behaviors given smaller subsets of the state space, preferably one or two dimensions for feasible learning speeds during the limited evaluation time in an embodied evolution context. The designer only needs to provide the state and action representations of the modules and the global reward functions shared by all modules. The specific behavior of a learning module is genetically determined by the states it is active in, as selected by the top-layer neural network, the metaparameters, and the shaping rewards.

Neural networks can easily be applied to input spaces of relatively large dimension and have been widely used in both evolutionary robotics and discriminatory tasks. It therefore seemed natural to use a neural network for the high-level discriminatory task of selecting the appropriate learning module according to the current overall state, and optimizing the neural network weights by the evolutionary process. The assumption that no learning occurs in the top-layer neural network is mostly based on pragmatic reasons, that is, to reduce the evolutionary search space, but it also seems natural that the agents have an innate ability to select between, for example, mating and foraging.

2.3 Reinforcement learning algorithm

Reinforcement learning (Sutton & Barto, 1998) is a computational approach to learning from interaction with the environment. An agent learns a policy, state-to-action mapping, based on scalar reward signals received from the environment. At time t the agent observes the current state s_t. The agent selects an action, a_t, according to the current stochastic policy, $\pi(s_t, a_t) = P(a_t|s_t)$, that is, the probability of selecting an action, a_t, given the current state, s_t. The environment makes a state transition from s_t to s_{t+1}, according to the state transition probability $P(s_{t+1}|s_t, a_t)$, and the agent receives a scalar reward r_t. The goal is to learn a policy, π, that maximizes the cumulative discounted

future reward. The value of a state s_t, given the state-value function V^π under policy π, is the solution to the Bellman equation, defined as

$$V^\pi(s_t) = \sum_{a_t} \pi(s_t, a_t) \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t)[R(s_{t+1}, s_t, a_t) + \gamma V^\pi(s_{t+1})], \quad (1)$$

where $R(s_{t+1}, s_t, a_t)$ is the reward for taking action a_t in state s_t, resulting in the new state s_{t+1}, and γ is the discount factor of future rewards. Similarly, the value of selecting action a_t in state s_t, given the action-value function Q^π, is defined as

$$Q^\pi(s_t, a_t) = \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t)[R(s_{t+1}, s_t, a_t) + \gamma \sum_{a_{t+1}} \pi(s_{t+1}, a_{t+1}) Q^\pi(s_{t+1}, a_{t+1})]. \quad (2)$$

2.3.1 Sarsa(λ) with tile coding. In this study, we use the Sarsa reinforcement learning algorithm (Rummery & Niranjan, 1994; Sutton, 1996). Sarsa is an on-policy reinforcement learning algorithm, which learns an estimate of the action-value function, Q^π, while the agent follows policy π. In the basic one-step version of Sarsa the Q-value for the current state, s_t, and action, a_t, is updated as

$$Q^\pi(s_t, a_t) \leftarrow Q^\pi(s_t, a_t) + \alpha[r_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t)], \quad (3)$$

where α is the learning rate.

Eligibility traces are a basic mechanism to increase the efficiency of reinforcement learning algorithms. For action-value based algorithms, each state–action pair is associated with a memory, the eligibility trace, $e_t(s, a)$. The temporal-difference error (TD-error), $\delta_t = r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$, is propagated back along the trajectory of state–action pairs leading to the current state, s_t, and action, a_t, decaying by $\gamma\lambda$ per time step, where λ is the trace decay rate. The eligibility traces version of Sarsa is called Sarsa(λ), and updates the Q-values according to

$$Q^\pi(s, a) \leftarrow Q^\pi(s, a) + \alpha\delta_t e_t(s, a) \quad \text{for all } s, a. \quad (4)$$

The implementation of the eligibility traces used in this study, as recommended by Singh and Sutton (1996), is called replacing traces with optional clearing, and is defined as

$$e_t(s, a) = \begin{cases} 1 & \text{if } s = s_t \text{ and } a = a_t; \\ 0 & \text{if } s = s_t \text{ and } a \neq a_t; \\ \gamma\lambda e_{t-1}(s, a) & \text{if } s \neq s_t. \end{cases} \quad \text{for all } s, a. \quad (5)$$

The optional clearing (the second line) sets the traces for all non-selected actions from the revisited state to 0.

To handle the continuous state information received by the robots' sensors, we use tile coding (Sutton, 1996) to approximate the Q-values. Tile coding represents the value of a continuous variable as a large binary feature vector with many 0s and a few 1s. The idea is to partition the state space multiple times, where the partitions are called tilings and the elements of the tilings are called tiles. For each state exactly one tile is active in each tiling, corresponding to the 1s in the binary feature vector. The computation of the action-values is therefore very simple, by summation of the components of the parameters representing the approximated Q-values, $\boldsymbol{\Theta}$, corresponding to the nonzero features, $\boldsymbol{I}_s$, in state s: $Q(s, a) = \sum_{i \in \boldsymbol{I}_s} \Theta(i, a)$.

For Sarsa the policy, π, is derived from the action values. The most common action-selection method is probably ϵ-greedy, where the agent selects the greedy action (the action corresponding to the largest Q-value, $a^* = \arg\max_a Q(s, a)$) most of the time, but with a small probability, ϵ, the agent selects a random action. An alternative, used in this study, is softmax action selection, where the actions are ranked and weighted according to their action values. The most common softmax method uses a Boltzmann distribution and selects an action a, with a probability of

$$P(a|s) = \frac{e^{\frac{Q(s, a)}{\tau}}}{\sum_b e^{\frac{Q(s, b)}{\tau}}}, \quad (6)$$

where the positive metaparameter τ is called the temperature.

2.3.2 Potential-based shaping rewards. Shaping rewards are a popular technique for improving the performance of reinforcement learning algorithms. The agent is provided with knowledge in the form of an additional reward signal that guides the agent to states with large rewards and/or from states with small rewards. However, if the shaping is not carefully designed the agent can be trapped in a suboptimal behavior, that is, the learning converges to a solution that is optimal in the presence of the shaping rewards, but sub-optimal for the original problem. Ng et al. (1999) proved that any policy optimal for a Markov decision process (MDP) with potential-based shaping rewards, where the shaping rewards depend only on the difference of a function of successive states, will also be optimal for the MDP without shaping rewards. They define a potential function $\Phi(\cdot)$ over states, where

the shaping reward, $F(s_t, s_{t+1})$, for the transition from state s_t to s_{t+1} is defined as

$$F(s_t, s_{t+1}) = \gamma\Phi(s_{t+1}) - \Phi(s_t), \quad (7)$$

where γ is the discount factor of future rewards. The shaping rewards are added to the original reward function for every state transition, changing the update of the Q-values in Sarsa (Equation 3) into

$$Q^\pi(s_t, a_t) \leftarrow Q^\pi(s_t, a_t) + \alpha[r_t + \gamma\big(Q^\pi(s_{t+1}, a_{t+1}) + \Phi(s_{t+1})\big) - Q^\pi(s_t, a_t) - \Phi(s_t)] \quad (8)$$

Wiewiora (2003) complemented this result by showing that using potential-based shaping rewards is equivalent to using the same potential function as a nonuniform initial action-value function, $Q(s, a) = \Phi(s)$. If the agent uses an advantage-based policy, which is defined as a policy that selects an action in a given state with a probability that is determined by differences in the Q-values, not their absolute magnitude, then the updates of the Q-values are equivalent in both cases. Given the fact that almost all reinforcement learning policies are advantage-based, for example, greedy, ϵ-greedy, and softmax, this result is applicable for most reinforcement learning applications.

The details of the implementation of the Sarsa algorithm in our embodied evolution context are described in Appendix C.

2.3.3 Metaparameters. The performance of reinforcement learning algorithms depends critically on a few metaparameters that directly influence the learning updates or the exploration of the environment (Doya, 2002). In this study we use four metaparameters that are coded as real-valued genes and are optimized by the evolution:

- α ($0 \le \alpha \le 1$), the learning rate in the updates of the Q-values (Equation 3). For larger values of α, the estimated target of the learning update, $r_t + \gamma Q(s_{t+1}, a_{t+1})$, is weighted more strongly in the computation of the updated Q-value, but it also makes the learning more unstable due to stochastic variations in the state transitions. If $\alpha = 0$, then the Q-values are not updated at all. If $\alpha = 1$, then the current Q-value is replaced by the estimated learning target. In convergence proofs for reinforcement learning algorithms, a standard assumption is either that α is sufficiently small or that α is decreasing over time.
- γ ($0 \le \gamma \le 1$), the discount factor of future rewards (see Equations 1 and 2). γ determines the value of future rewards at the current state, and thereby how farsighted the agent is. If γ is zero, the agent only tries to maximize the immediate rewards, r. As γ approaches one, future rewards are more strongly weighted in the calculation of the expected accumulated discounted reward.
- λ ($0 \le \lambda \le 1$), the trace decay rate that controls the exponential decay of the eligibility traces (Equation 5). If $\lambda = 0$, then all traces are zero except for the trace corresponding to the current state and action, $e_t(s_t, a_t)$, and the update of the Q-values is equal to the one-step version of the learning (see Equation 3 for Sarsa-learning). For increasing values of λ, but still $\lambda < 1$, the Q-values of preceding state–action pairs are changed more, but more temporally distant state–action pairs are changed less since the traces have decayed for more time steps. If $\lambda = 1$, then the traces are only decayed by γ per time step.
- τ ($\tau > 0$), the temperature that controls the trade-off between exploration and exploitation in softmax action selection (Equation 6). Higher temperatures decrease the differences between the action selection probabilities, and make the selection more stochastic. Lower temperatures increase the differences between the action selection probabilities and make the selection less stochastic. In the limit, $\tau \to 0$, the action selection becomes deterministic and the agent always selects the greedy action.

3 Experimental setup

To validate the proposed embodied evolution method, we used a simulation environment developed for the Cyber Rodent robot (see the picture used to illustrate the survival task in Figure 2). The four robots can detect each other by the green tail LEDs, and by the red faces on the front of the robots, where the infrared ports for genotype exchanges are located. The gray circles with dark centers represent the eight battery packs used for recharging the virtual agents' internal batteries. The two robots in the bottom of the environment have performed a successful mating and their LEDs are therefore turned off.

3.1 Cyber rodent robot

The Cyber Rodent robot platform (see the pictures used to illustrate the virtual agents in Figure 2) was developed for the Cyber Rodent project (Doya & Uchibe, 2005). The main objective of the Cyber Rodent project is to study adaptive mechanisms of artificial agents under the same fundamental constraints as biological agents, namely self-preservation and self-reproduction. The Cyber Rodent is a rat-like mobile robot, 22 cm in length and 1.75 kg in weight. The robot has a variety of sensors, including a wide-angle

C-MOS camera, an infrared range sensor, seven infrared proximity sensors, gyros, and accelerometers. It has two wheels and a maximum speed of 1.3 m s^{-1}, and a magnetic jaw that latches onto battery packs. It also has a speaker, two microphones, a three-color LED for audio-visual communication, and an infrared communication port.

The field of view of the visual system is approximately [−75°, 75°], and within the angle range [−45°, 45°] the robot can detect batteries (blue LED) and the green tail LED of another robot up to approximately 1.2 m, and the red face of another robot up to 0.8 m. Outside this range the detection capability decreases rapidly, for example, for the angles ±75° the robot can only detect the batteries, tail LEDs, and faces up to approximately 0.2 m.

3.2 Simulation environment

The simulation environment was developed to mimic the properties of the real Cyber Rodent robot. The dimensions of the environment were set to 2.5 m × 2.5 m, and the simulated vision system used the estimated properties described in Section 3.1 above. In the simulator, Gaussian noise was added to the angle and distance state information received from the simulated visual system, with zero mean and a standard deviation of 1° for all angle states, and a standard deviation of 2 cm for all distance states. The infrared port for exchange of genotypes is located slightly to the right of the center in the front of the Cyber Rodent. In the simulator, the maximum range of the infrared communication was set to 1 m and the angle range was set to [−30°, 30°] (calculated from the position of the infrared port). For a mating to be successful, at least one of the robots had to initiate the infrared communication and both of the robots had to be within each others mating range, both before and after the virtual agents controlling the robots executed the actions of the currently selected reinforcement learning modules.

In the simulations, there were eight battery packs in the environment. To capture a battery back, thereby charging the virtual agent's internal battery, the virtual agent had to turn on the latching magnet of the robot it was controlling. After a successful capture of a battery pack, the battery pack was moved to a new, randomly selected position in the environment.

3.3 Reinforcement learning modules

In the experiments, we used two reinforcement learning modules: *mating*, for learning to exchange genotypes with another robot, and *foraging*, for learning to capture external battery packs in the environment to charge the virtual agent's internal battery. To learn the two behaviors, we used the Sarsa algorithm with tile coding and potential-based shaping rewards (for implementation details see Algorithm 3 in Appendix C), with softmax action selection (see Equation 6). The time step of the learning was set to 240 ms. The state was defined by the angle and distance to the closest battery pack, tail LED, or face of another robot. The state information was normalized to the interval [0, 1]: $(s + 75°)/150°$ for angle states, and $1 - s/d_{max}$ for distance states, where d_{max} was set to 1.2 m for batteries and tail LEDs, and 0.8 m for faces of other robots. In addition, each module used three extra discrete states for the situations where the target (tail LED and face for mating, and battery pack for foraging) was not visible: (1) if a target was visible at an angle $< 0°$ in the last three time steps; (2) if a target was visible at an angle $> 0°$ in the last three time steps; and (3) if a target has not been visible for more than three time steps. The virtual agents received a global reward, r_t, of +1 for a successful battery capture, +1 for a successful mating, and 0 for all other state transitions. The tile coding consisted of three tilings offset by a constant amount in each state dimension. In each tiling, the angle dimension was divided into 11 equidistant tiles and the distance dimension was divided into four equidistant tiles. In addition, each module used three tiles representing the three states in which the target was not visible. The Q-values were initialized to 0 for all states and actions in the beginning of each generation ($\boldsymbol{\Theta} = \mathbf{0}$ in line 3 in Algorithm 1 in Appendix A).

The potential functions, $\Phi(\cdot)$, representing the shaping rewards were approximated by normalized Gaussian radial basis networks with equidistant basis functions, ϕ_i, in each state dimension, defined as

$$\Phi(s) = \sum_i v_i \cdot \phi_i(s), \tag{9}$$

$$\phi_i(s) = \frac{e^{-\frac{|s-c_i|^2}{2\sigma_i^2}}}{\sum_j e^{-\frac{|s-c_j|^2}{2\sigma_j^2}}}, \tag{10}$$

where c_i is the center position of ϕ_i with width σ_i (equal to $\sqrt{2}/4$ for all basis functions in the experiments). In addition, each module had three extra shaping reward parameters, v_i, corresponding to the potential values of the three states in which the target was not visible.

The state of the foraging behavior was of two dimensions and defined as

s_1: the normalized angle to the closest battery pack;
s_2: the normalized distance to the closest battery pack.

The foraging behavior could execute five actions, pairs of velocities (mm s^{-1}) of the left and the right

wheel: {(167, −167), (250, 167), (250, 250), (167, 250), (−167, 167)}. The latching magnet for capturing external batteries was always turned on when a virtual agent executed the foraging behavior. For the foraging behavior, the potential function of the shaping rewards was parameterized by 15 parameters, $\boldsymbol{v}^1$: 12 for the states in which a battery was visible (four basis functions in the angle dimension and three basis functions in distance dimension), and 3 parameters corresponding to the states in which no battery was visible.

The state of the mating behavior was of three dimensions and defined as

s_1: discrete state that was set to 1 if the face of another robot was visible, otherwise it was set to 2;
s_2: if $s_1 = 1$, then the normalized angle to the closest face of another robot, otherwise the normalized angle to the closest tail LED;
s_3: if $s_1 = 1$, then the normalized distance to the closest face of another robot, otherwise the normalized distance to the closest tail LED.

The mating behavior could execute seven actions, pairs of velocities (mm s^{-1}) of the left and the right wheel: {(0, 0), (167, −167), (250, 167), (250, 250), (167, 250), (−167, 167), (−167, −167)}. The first action, (0, 0), was defined as the *mating action*. If a virtual agent executed the mating action, then it would initiate a mating. If another robot was within the range of the infrared communication, then it would always answer the mating invitation, even if the virtual agent controlling the other robot was executing the foraging behavior. If both robots were within each others mating range, both before and after they had executed their actions, then the mating was considered successful, and the virtual agents controlling the mating robots would exchange genotypes. After a successful mating, a virtual agent could not execute the mating behavior again until it had captured a battery pack, or until 50 time steps had passed. During this time the tail LED was turned off. For the mating behavior, the potential function of the shaping rewards was parameterized by 27 parameters, $\boldsymbol{v}^2$: 12 for the states in which the face of another robot was visible (four basis functions in the angle dimension and three basis functions in distance dimension), 12 for the states in which only the tail LED of another robot was visible, and 3 parameters corresponding to the states in which no other robot was visible.

3.4 Neural network controller

The input state of the top-layered neural network, $\boldsymbol{x}$, was of five dimensions and defined as

x_1: constant bias of 1;
x_2: normalized internal energy level, that is, the virtual agent's current energy level divided by the maximum energy level;
x_3: the normalized distance to the closest battery pack. If no battery was visible, then x_3 was set to −1;
x_4: the normalized distance to the closest tail LED. If no tail LED was visible, then x_4 was set to −1;
x_5: the normalized distance to the closest face. If no face was visible, then x_5 was set to −1.

In the experiments, we only used two learning modules, and, therefore, we were able to use only one output unit. Instead of selecting the learning module corresponding to the output unit with the largest value, we selected the learning module according to a predefined threshold of the output from one output unit. This changes line 7 in Algorithm 2 in Appendix B into

7a if $\boldsymbol{wx} \leq 0$ **then**
7b $m = 1$ `/* foraging behavior */`
7c else
7d $m = 2$, `/* mating behavior */`

and the dimension of the weight vector, $\boldsymbol{w}$, is halved $(1 \times N_x)$, that is, equal to the dimension of the input state vector, $\boldsymbol{x}$.

In the experiment, each virtual agent had an internal battery with a maximum capacity, E_{max}, of 500 energy units, which was initialized to 400 energy units in each generation. In each time step, the energy of a virtual agent controlling a robot was decreased by 1 energy unit, and if the robot captured a battery pack the energy level was increased by 100 energy units (the `UpdateEnergy()` function in line 13 in Algorithm 2 in Appendix B).

3.5 Embodied evolution framework

In the experiments, we used four robots (N_{sub}) with 20 virtual agents (N_{va}) inside each robot. The number of time-sharings (N_{ts}) was set to three, and each time-sharing lasted for 400 time steps (T_{ts}), giving a total lifetime of 1200 time steps. The genotype of a virtual agent, G consisted in total of 51 real-valued genes:

- $G_{1:5}$: the weights of neural network controller, $\boldsymbol{w}$;
- $G_{6:20}$: the shaping reward parameters of the foraging behavior, $\boldsymbol{v}^1$;
- $G_{21:47}$: the shaping reward parameters of the mating behavior, $\boldsymbol{v}^2$;
- G_{48}: α (randomly initialized in the interval [0, 1/3]);
- G_{49}: γ (randomly initialized in the interval [0, 1]);
- G_{50}: λ (randomly initialized in the interval [0, 1]);
- G_{51}: τ (randomly initialized in the interval [0, 1]).

The initial neural network weights and the initial parameters for the shaping rewards were drawn from

a Gaussian distribution with zero mean and a standard deviation of 0.1.

We defined the reproduction condition (the `ReprodCond()` function in line 13 in Algorithm 1 in Appendix A), that is, the condition that a virtual agent would reproduce potential offspring in its own subpopulation, as a linear function of the virtual agent's internal energy level, E, at the mating occasion. The probability, p_{mate}, that a mating was reproductive was defined as

$$p_{mate} = \frac{E}{E_{max}}. \tag{11}$$

If a mating was reproductive, then two potential offspring were created by crossover and mutation (the `GenOps()` function in line 14 in Algorithm 1 in Appendix A). We applied one-point crossover with a probability of 1, by switching substrings of the genotypes at a randomly selected crossover gene. The genes in the two newly created genotypes were then mutated with a probability of 0.1, which added a value drawn from a Gaussian distribution with zero mean and a standard deviation of 0.1. The genes representing the metaparameters were limited to their initial ranges, except for τ, for which only the lower initial limit, 0, was enforced. If τ was evolved to become less than 0.001, then the action selection was considered to be greedy, and the action corresponding to the largest Q-value was selected.

3.6 Comparison with centralized evolution

To validate the effectiveness of our proposed method, we compared the result of our proposed selection scheme with experiments in which we used standard centralized selection. In the centralized scheme, there was no list of offspring reproduced during the execution of the survival task. Instead, after all virtual agents in all subpopulations had been evaluated for a full lifetime (three time-sharings of 400 time steps), the new population was created by elitism and tournament selection. The fitness was computed as the number of reproductive matings and the fittest 10% of the individuals were transferred directly to the new population (elitism). The remaining 90% of the new population was then selected by tournament selection with a tournament size of two, that is, two individuals were randomly chosen from the old population and the fittest of the two was then transferred to the new population. The genotypes in the new population were randomly paired and one-point crossover was applied with a probability of 1, and each gene was mutated with a probability of 0.1 (identical to the genetic operations in our proposed framework). The genotypes in the new population were then randomly placed in the four subpopulations.

4 Simulation results

This section presents the results of the simulation experiments. We first show (Section 4.1) the overall evolutionary performance and compare the performance of our proposed method with the performance of centralized evolution. We then present the obtained learning module selection policy (Section 4.2) and the obtained potential functions of the shaping rewards for the foraging and mating behaviors (Section 4.3). Finally, we present and discuss (Section 4.4) the evolution of the metaparameters.

4.1 Evolutionary performance

Figure 4 shows the overall performance of the evolutionary experiments. The presented results are the

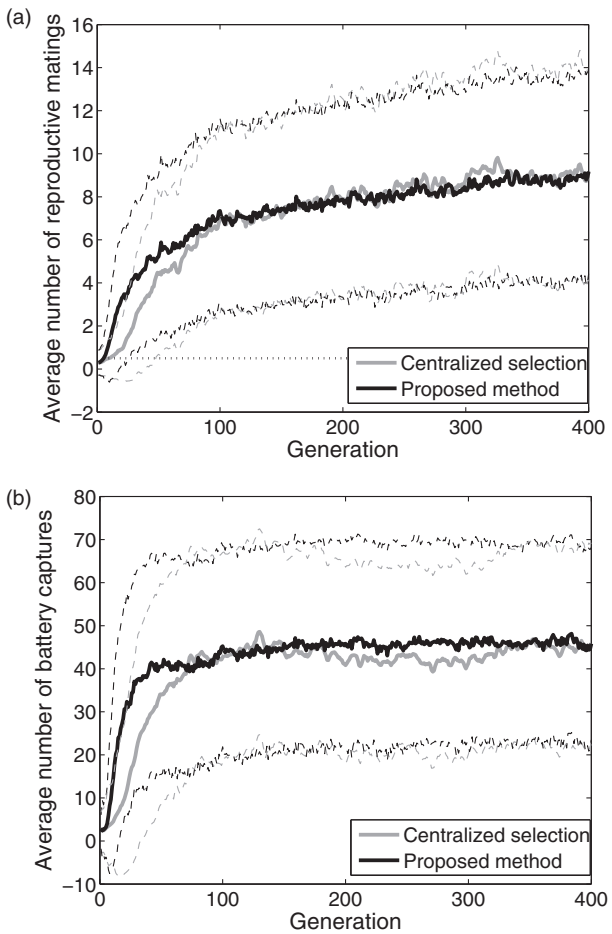

Figure 4. The average performance over 20 simulation runs for the proposed embodied evolution framework (black lines) and the centralized selection scheme (gray lines), with standard deviations (dashed lines). (a) Average number of reproductive matings (b) Average number of battery captures.

average performance over 20 simulation runs of all individuals in the four subpopulations for both our proposed embodied evolution framework (black lines) and the evolution with centralized selection (gray lines), with standard deviations (dashed lines). For the number of reproductive matings (Figure 4a), the average results are very similar. The performance increases steadily from approximately 0.3 reproductive matings in the first generation to approximately 7 reproductive matings after 100 generations. The performance then slowly increases to reach the final level of approximately 9 reproductive matings after 300 generations. The constant dotted black line with a value of 0.5 indicates the number of reproductive matings needed for a sustainable population, that is, that each virtual agent reproduces on average at least one potential offspring. Our proposed method reaches this threshold after approximately 5 generations, while the evolution with the centralized selection scheme needs approximately 10 generations to reach this performance level. The average number of batteries captured per virtual agent (Figure 4b) increases rapidly to about 42 batteries with a large variance in the populations, and it then stays at this level for the remaining time of the evolution. Our proposed method reaches this stable level after about 45 generations and the evolution with centralized selection reaches the stable level after about 75 generations.

The proposed embodied evolution method limits the number of mating partners a virtual agent can reproduce potential offspring with. Our intuition was therefore that our proposed method would be less effective than centralized selection, where all individuals potentially can reproduce offspring with all other individuals. It is therefore very encouraging that results show that our proposed method does not reduce the evolutionary performance. The results give strong evidence that the settings used in the experiments, four robots and three time-sharings per virtual agent, at least do not have a negative effect on the evolutionary performance.

For the remaining part of this section we only present the results for the proposed embodied evolution framework.

4.2 Obtained neural network selection policy

Figure 5 shows the average selection policies for generations 10, 40, and 400. To visualize the selection policies, we divided the state input to the neural network into five distinct state types:

- S_1: A battery and a face of another robot are visible;
- S_2: A battery and a tail LED are visible;
- S_3: A battery is visible;

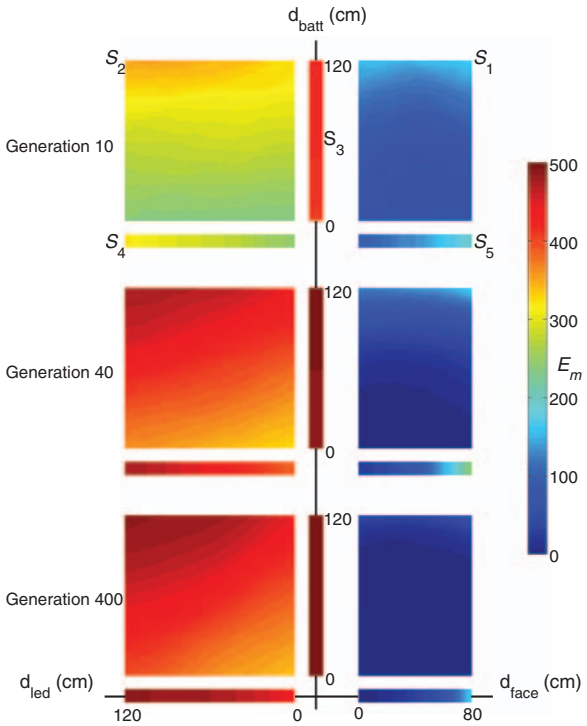

Figure 5. The evolution of the neural network module selection policy, shown as the average energy threshold E_m for mating in the five different state types, S_i, at generation 10, 40, and 400. The *y*-axis corresponds to distance to the battery in cm, the "positive" *x*-axis corresponds to distance to the face in cm, and the "negative" *x*-axis corresponds to the distance to tail LED in cm. In each state type, a virtual agent selects the foraging module if the current energy level is less than or equal to the threshold value, and the mating modules if the current energy level is greater than the threshold value.

- S_4: A tail LED is visible;
- S_5: A face of another robot is visible.

The *y*-axis corresponds to distance to the battery in cm, the "positive" *x*-axis corresponds to distance to the face in cm, and the "negative" *x*-axis corresponds to the distance to tail LED in cm. The selection policies are visualized as energy thresholds E_m for mating. If the current energy level is less than or equal to the threshold value, then the virtual agent selects foraging, otherwise the virtual agent selects mating. In the figure, dark blue color means that the agent always selects mating and dark red means that the agent always selects foraging. Note that since the virtual agents on average captured a battery every 30th time step, the energy levels very rarely fell below, say, half the maximum energy level, 250 units, during the experiments.

For the state types in which the face of another robot is visible (S_1 and S_5), we assume that the tail LED of the same robot is visible and that the angle to the tail LED is the same as for the face. The selection policy of

the neural controller converges relatively quickly. After about 40 generations, the average obtained policy is very similar to the final obtained solution, which corresponds to the time when the average number of captured batteries reaches a stable level (see Figure 4b).

The obtained average selection module selection policy can be summarized as

- If a face of another robot is visible, S_1 and S_5 (the two right panels), then select mating.
- Else if only a battery is visible, S_3 (top middle panel), then select foraging. The only exception is if the energy level is very high, above approximately 495 energy units (not visible in the figure). This means that in first few time steps after a virtual agent has recharged fully it will try to look for a mating partner, by selecting the mating module, even if there is a battery pack close at hand and no other robot is visible.
- Else if a tail LED is visible (the two left panels), and if the energy level is high, then wait for the potential mating partner to turn around by selecting mating. A virtual agent is more likely (at lower energy thresholds) to wait for the mating partner for shorter distances to the tail LED and shorter distances to the battery pack. For example, for the states in which a tail LED and a battery pack are visible, S_2 (top left panel), the energy thresholds are 337, 400, and 490 energy units for the distances 0, 40, and 120 cm to both targets. For the states in which only the tail LED is visible, S_4 (bottom left panel), the energy threshold is shifted upwards: 440, 471, and 490 energy units for the distances 0, 40, and 120 cm to the tail LED.

4.3 Obtained shaping rewards

Figure 6 shows the average potential function of the shaping rewards, $\Phi(\cdot)$, for the foraging module in generations 50, 100, 200, and 400. The obtained potential function seems intuitively correct, higher potential values for shorter distances and smaller absolute angles to the battery pack. This shape is already approximately obtained after 50 generations. However, this is very difficult to see in the figure, because the difference between the largest average potential ($a_{batt} \approx -12^\circ$, $d_{batt} = 0$ cm) and the smallest average potential ($a_{batt} = -75^\circ$, $d_{batt} = 120$ cm) is only 0.07. The amplitude of the potential function is gradually increasing throughout the evolutionary process, and in the last generation the difference between the maximum average potential ($a_{batt} \approx 0^\circ$, $d_{batt} = 0$ cm) and the minimum average potential ($a_{batt} = \pm 75^\circ$, $d_{batt} = 120$ cm) is 0.385.

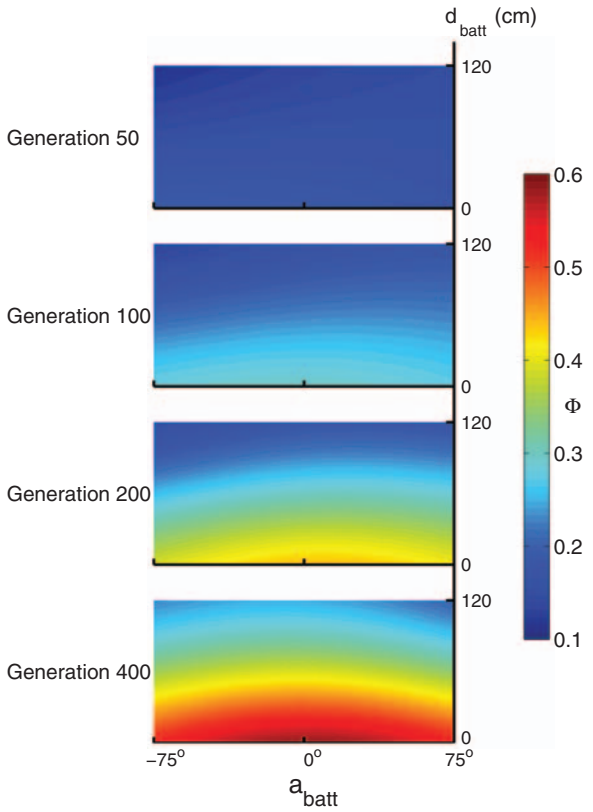

Figure 6. The average potential function, $\Phi(\cdot)$, for the foraging module in generations 50, 100, 200, and 400. The *y*-axis corresponds to distance to the battery in cm and the *x*-axis corresponds to the angle to the battery. The color coding shows the values of the average potential function, where more red colors correspond to larger values and more blue colors correspond to smaller values.

The number of matings varies in the 20 different simulation runs. For example, in the last generation, the average number of matings ranges from 8.1 ± 4.6 to 12.4 ± 5.1. The primary explanation for the differences in the number of matings is the differences in the obtained potential functions of the shaping rewards of the mating module. We therefore only present the average results of the best simulation run (see Figure 7).

The right column shows the potential functions for the states in which the face of another robot is visible, and the left column shows the potential functions for the states in which only the tail LED is visible. The dashed black lines in the right panels represent the angle limits of the infrared communication for the exchange of genotypes. The curvatures of the lines are caused by the infrared port being located slightly ahead and to the right of the camera. For the states in which the face is visible, the average obtained potential function in the last generation gives higher potential for longer distances and negative angles to the face of the other robot. The angles of the maximum potentials

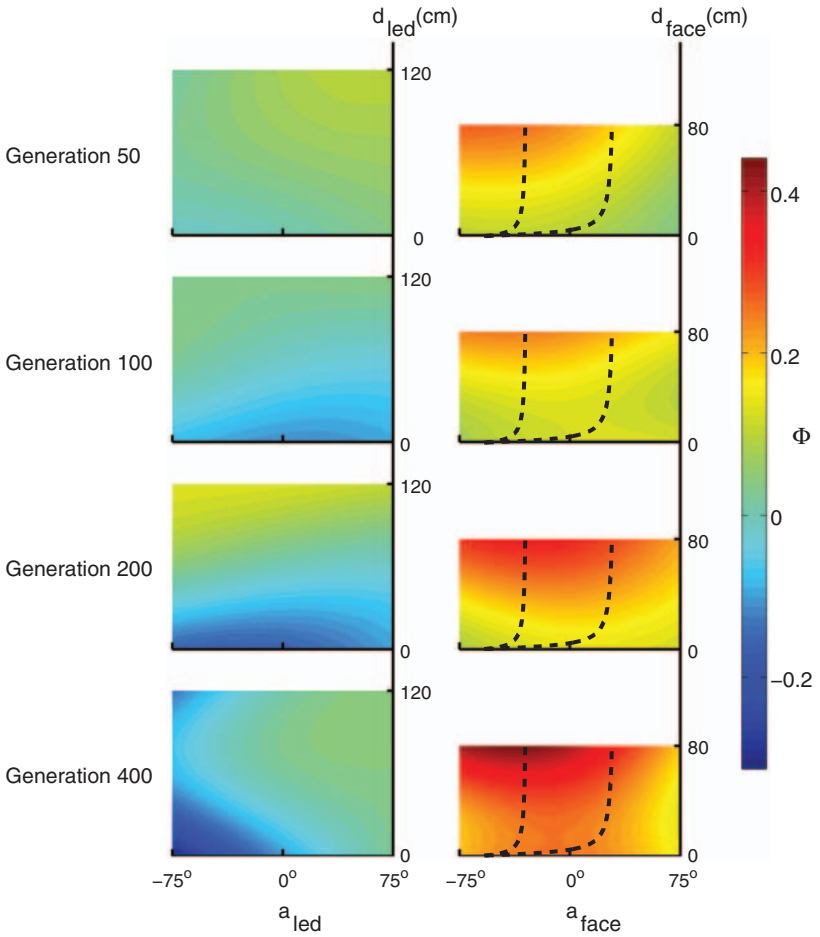

Figure 7. The average potential functions, $\Phi(\cdot)$, for the mating module in generations 50, 100, 200, and 400 in the best simulation run. The right column shows the potential functions for the states in which the face of another robot is visible. The *y*-axis corresponds to distance to the face in cm and the *x*-axis corresponds to the angle to the face. The left column shows the potential functions for the states in which only the tail LED is visible. The *y*-axis corresponds to distance to the tail LED in cm and the *x*-axis corresponds to the angle to the tail LED. The color coding shows the values of the average potential function, where more red colors correspond to larger values and more blue colors correspond to smaller values.

ranges from approximately -10° to -32° for the minimum distance (0 cm) and the maximum distance (80 cm), respectively. That the evolution assigns higher potentials to negative angles is explained by the infrared port for the exchange of genotypes being located to right of the camera system. The preference for longer distances to the face is probably explained by longer distances between the two robots making it less likely that one of virtual agents will move its robot outside the angle range of the infrared communication of the mating partner. During the first half of the evolution (the top three right panels) the amplitude of the potential function is gradually increasing and the shape of function is almost opposite to the obtained potential function for foraging, giving higher potentials for longer distances and smaller absolute angles. During the last half of evolution (bottom right panel) the potential function becomes less dependent on the distance to the face. In the last generation, for distances between 10 cm and 70 cm, the states with the highest potentials correspond roughly to the states within the angle range of the infrared communication.

For the states in which only the tail LED is visible (left panels), it is more difficult to interpret the obtained potential function in the last generation. Positive angles and longer distances correspond to higher potentials, with the highest potentials between approximately 80 cm and 120 cm. Not surprisingly, the obtained potential functions always prefer states in which the face is visible. The potential values are in the ranges of -0.31 to 0.05 when only the tail LED is visible, and 0.14 to 0.44 when the face is visible. During the first half of the evolution (the top three left panels) the potential function seems to change rather randomly. In generations 50 and 200, there is also a considerable overlap in the values for the potential function for the states in which

the face is visible and the states in which only the tail LED is visible.

4.4 Obtained metaparameters

Figure 8 shows the average values of the four metaparameters. An interesting result is the evolution of τ, controlling the trade-off between exploration and exploitation in the softmax selection (black line in Figure 8a). τ very quickly drops to approximately 0 (in median, τ is exactly zero after generation 194). This result means that action selection tends to be equal to the greedy action selection strategy, and always selects the action corresponding to the largest Q-value. This result indicates that, in the presence of sufficiently good shaping rewards, additional exploration in the form of a stochastic policy is unnecessary and actually decreases the learning performance. This result is consistent with the result in our study (Elfwing, Uchibe, Doya, & Christensen, 2008) on the coevolution of metaparameters and shaping rewards. It is also consistent with theoretical analyses performed by Laud and DeJong (2003) which show that shaping rewards can reduce the amount of exploration required for efficient learning. They analyzed shaping in terms of a reward horizon, which is a measure of the number of decisions the agent must take before experiencing accurate reward feedback. For example, if the agent only receives a delayed reward at the goal state, then the reward horizon is equal to the task length. If the potential function of the shaping rewards is equal to the optimal value function, V^*, then the reward horizon is equal to 1. However, it is surprising that the τ-value tends to be zero so early on in the evolutionary process, when the potential functions of the shaping rewards have not converged. Note that $\tau = 0$ does not mean that the action selection is deterministic in the early stages of the learning. For each state, the action selection becomes deterministic after a virtual agent has experienced positive reward feedback. The Q-values are uniformly initialized to 0 in each generation, which means that before the agent receives positive reward feedback it will select a random action among the actions that have not been tested.

The average value of λ (black line in Figure 8b) decreases gradually throughout the evolutionary process, from approximately 0.5 in the first generation to 0.3 in the last generation. The variance in the λ-values is very large and remains relatively constant during the last half of the evolution, with an approximate standard deviation of 0.21. A possible explanation for the gradual decrease in the average λ-value is that it corresponds to the gradually more optimized shaping rewards. More optimized shaping rewards suggest that the shaping rewards, $\gamma\Phi(s_{t+1}) - \Phi(s_t)$, adapt to become more equal to the discounted difference in optimal state values, $V^*(s)$, between successive states, $\gamma V^*(s_{t+1}) - V^*(s_t)$. This means that the agent receives a more accurate reward feedback in each state transition, and, therefore, reduces the need to propagate the received reward feedback to preceding state–action pairs. This hypothesis has support in the experimental data. Figure 9 shows the negative correlation ($r = -0.68$ and $p = 0.0009$) in the last generation between the average λ-value and the average number of matings, used as an estimate of the efficiency of the obtained shaping rewards for mating in each simulation run. The circles correspond to the average obtained values in each simulation run and the line corresponds to the mean squares linear fit of the data.

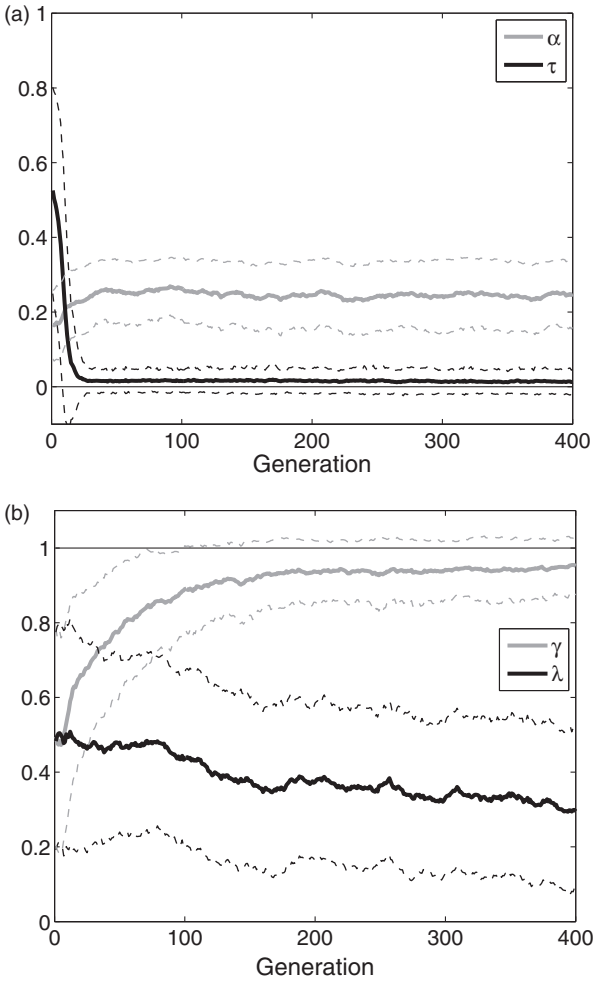

Figure 8. Evolution of the average metaparameters over the 20 simulation runs, with standard deviations (dashed lines). (a) α and τ (b) γ and λ.

The results for the evolution of α (gray line in Figure 8a) and γ (gray line in Figure 8b) are similar. The two metaparameters reach relatively stable levels of 0.24 ± 0.09 for α and 0.94 ± 0.08 for γ, after about 150 generations.

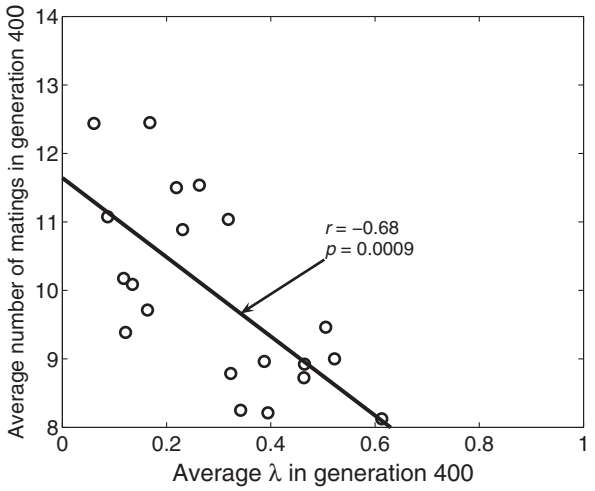

Figure 9. Correlation in the last generation between the average λ-value and the average number of matings, used as an estimate of the efficiency of the obtained shaping rewards for mating in each simulation run. The circles correspond to the average obtained values in the last generation in each simulation run and the line corresponds to the mean squares linear fit of the data.

5 Hardware results

To validate the performance of the solutions that were evolved in simulation, we performed hardware experiments using two robots that are presently available and a smaller environment (1.25 m × 1.75 m) with six battery packs (see Figure 10). In our experience, it is very difficult to evolve the virtual agents from scratch when there are only two subpopulations. Therefore, we first evolved the individuals in four simulated robots for 100 generations in simulation, under the same conditions as in the previously described experiments. Two of the subpopulations were then moved to the smaller simulation environment and evolved for 100 additional generations. The best performing individuals in the two subpopulations in the last generation were then transferred to the real Cyber Rodent robots.

The main difference between the hardware and simulation environments is that there is more uncertainty in the distance states corresponding to the tail LED and the face of the other robot in the hardware setting. In the idealized case in the simulator, the estimated distances to the LED and face are independent of the relative angle of the other robot. However, in the hardware, where we used blob detection to estimate the distances, the extracted blob size corresponding to the face decreases with larger relative angles to the other robot. The camera system located in the front of the Cyber Rodent can limit the view of the tail LED, and thereby decrease the extracted blob size corresponding to the tail LED. Another difference is that it is not possible to move a battery pack to new random position after a successful battery capture in the hardware setting. To prevent the robots from capturing the same battery pack repeatedly, we forced the robots to move away from a captured battery pack. After a successful battery capture, the robot was first moved backwards for a small distance and then rotated in the opposite direction (approximately 180°).

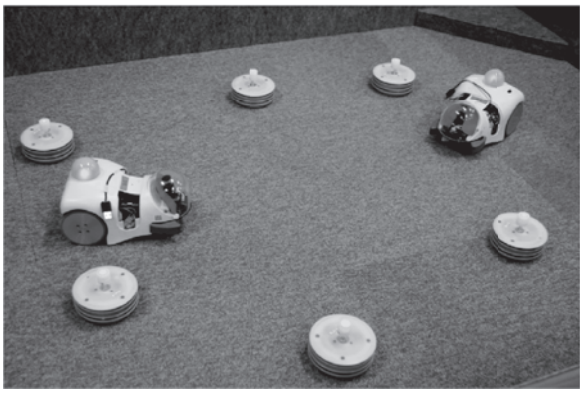

Figure 10. The hardware environment.

In the hardware experiments, we tested the two solutions obtained in simulation for a lifetime (1200 time steps). To investigate the importance of the shaping rewards we also performed experiments without the obtained shaping rewards included in update of the Q-values. To be able to validate the hardware results, we also performed simulation experiments under the same conditions. The results from these experiments are summarized in Figure 11. The hardware results are averages over 10 experiments and simulation results are averages over 100 experiments. Overall, the experimental results are very encouraging. The learning performance is very similar in the simulation environment and the hardware environment. With shaping rewards, the robots perform 6.1 ± 3.1 successful matings in hardware and 6.3 ± 3.4 successful matings in simulation. Without shaping rewards, the robots perform 1.6 ± 2.5 successful matings in hardware and 2.1 ± 2.4 successful matings in simulation. The only exception is that one of the robots, CR1, captures significantly more batteries in the hardware environment. The main cause of this is probably differences in the camera properties and in the calibration of the parameters for the blob detection, which made it more difficult for CR1 to detect the tail LED and the face of the other robot, compared with CR2. A minor cause of the difference in the number of captured batteries between the two robots in the hardware environment is probably due to differences in the obtained neural network selection policy and metaparameters, because CR1 also captured more batteries, both with and without shaping rewards, in the simulation environment. The results also show

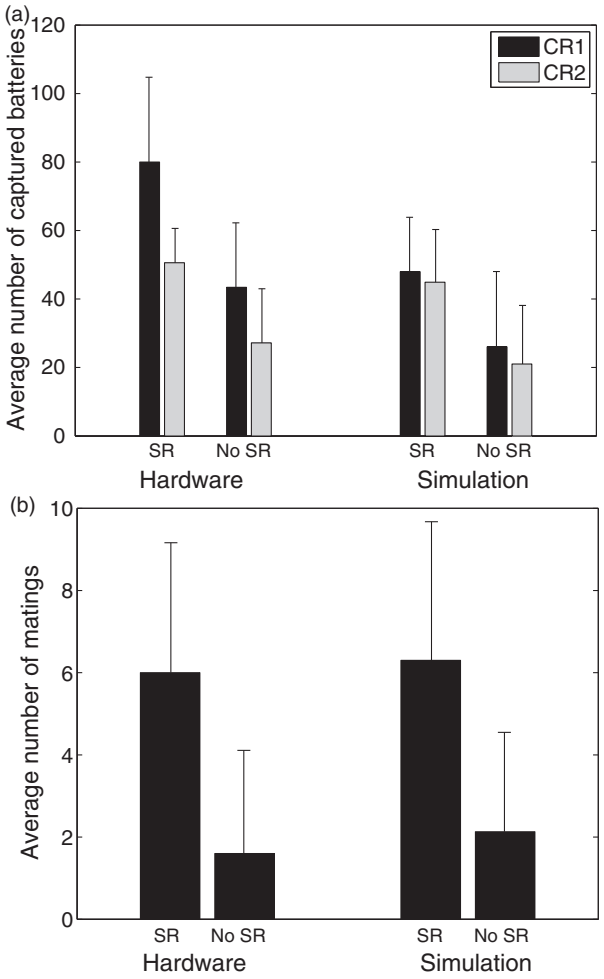

Figure 11. The average performance, with and without the obtained shaping rewards, in the hardware and the simulation environments. (a) Average number of captured batteries for the two robots, CR1 and CR2 (b) Average number of matings.

the great impact of the evolutionarily optimized shaping rewards. With shaping rewards, the robots capture almost twice as many battery packs and perform three times as many successful matings.

6 Discussion

In this article we propose an embodied evolution framework that utilizes time-sharing of subpopulations of virtual agents inside a limited number of robots. Within this framework we combine within-generation learning of basic behaviors, by reinforcement learning, and evolutionary adaptation of parameters that modulate the reinforcement learning. A top-layer neural network controller selects basic behaviors according the current environmental state and the virtual agent's internal energy level. The basic behaviors are learned by the reinforcement learning algorithm Sarsa(λ) and the learning is evolutionarily optimized by tuning an additional reward signal, implemented as potential-based shaping rewards, and the global metaparameters shared by all learning behaviors, such as the learning rate, α, the discount factor of future rewards, γ, the trace-decay rate, λ, and the temperature, τ, controlling the trade-off between exploration and exploitation in softmax selection. We apply a biologically inspired selection scheme, in which there is no explicit communication of the fitness information. The virtual agents can only reproduce offspring by mating with virtual agents controlling other robots in the environment, and the probability that a virtual agent will reproduce offspring in its subpopulation is dependent on the virtual agent's internal energy level at the mating occasion.

In standard centralized selection, an individual can potentially reproduce offspring with all other individuals. In contrast, in our proposed embodied evolution framework the number of potential mating partners a virtual agent can reproduce offspring with is limited by the number of time-sharings and subpopulations. It is therefore very encouraging that the simulation results showed that the evolutionary performance of the proposed method was almost identical to the performance of evolution with standard centralized selection. The experimental results show that the shaping rewards can reduce the amount of required exploration. τ, which controls the trade-off between exploration and exploitation in softmax action selection, tends to be zero very early on in the evolutionary process and, thereby, makes the action selection greedy. There is also a highly significant negative correlation between the trace decay rate λ and the obtained mating performance, which indicates that more optimized shaping rewards reduce the need to propagate reward feedback to preceding state–action pairs. We also transferred the best evolved solutions to a hardware environment, using two real robots. The performance in the hardware environment was similar to the performance in simulation under the same environmental conditions.

An interesting result is that there are variations in the obtained shaping rewards for the mating behavior in different simulation runs. Since the shaping rewards are gradually improving throughout the evolution, this suggests a feedback dynamic between the within-generation learning of the mating behavior and the evolutionary adaptations of the learning ability for mating behavior, which will be explored in subsequent work. If this is related to Baldwinian evolution (Baldwin, 1896), then the result of the within-generation learning of the mating behavior should influence the shape of the potential function of shaping rewards in subsequent generations.

One of the main points of the original embodied evolution methodology, as formulated by Watson et al. (2002), was that it should be applied to a colony

of physical robots. A natural extension of this study is to fully implement our proposed framework in hardware. One issue in achieving this goal is that it requires that the virtual agents keep the real robots "alive" for an extended period of time. It can be realized by, for example, tuning the time the robots recharge from the external battery packs according to the robots' internal energy levels.

Another problem in the implementation of the framework in hardware is the relatively long time needed for evaluating the learning performance of the basic behaviors. We therefore need to investigate the settings of the parameters in our framework required to achieve efficient evolution, such as the number of virtual agents in each subpopulation, the number of time-sharings, and the length of each time-sharing. Whiteson and Stone (2006) proposed a method that could be used to improve the evolutionary performance of our framework. They introduced explicit within-generation selection of the individuals chosen for evaluation. Instead of selecting individuals randomly, they selected individuals for evaluation according to standard reinforcement learning action selection strategies, for example, ϵ-greedy or softmax selection. They showed, for evolutionary function approximation in Q-learning, that explicit selection within the generations can improve the efficiency of the evolution.

6.1 Related studies

From a biological point of view, the evolutionary optimization of shaping rewards can be seen as a form of the Baldwin effect (Baldwin, 1896; Hinton & Nowlan, 1987; Turney, Whitley, & Anderson, 1996), that is, indirect genetic assimilation of learned traits. In this context, the shaping rewards represent an agent's innate learning ability and the learning process represents the lifetime adaptations of the agent's behavior. The main difference between our approach and the standard application of the Baldwin effect in evolutionary computation is that the genetic adaptations do not directly effect the agents' behaviors, that is, the genotypes do not code the policies of the behaviors explicitly. Instead, the genetic adaptations in the form of gradually improving shaping rewards increase the agents' learning abilities by accelerating the learning process. In this regard this study is related to the approach used by Floreano and Mondada (1996) and Urzelai and Floreano (2000) for evolving learning rules in neural controllers. Instead of evolving the weights of the neural controller directly, they evolved the learning rules and learning rates of the synaptic weights of the neural controller. Each weight of the neural network was changed continuously during an individual's lifetime, according to one of four genetically determined Hebbian learning rules and the change of the weight was computed as a function of the activations of the presynaptic and postsynaptic units. An obvious difference between these approaches is algorithmic. In the evolution of learning rules, the weights of the neural controller are modified by Hebbian learning and the learning has no explicit goal of its own. In our approach the learning is accomplished by a specific reinforcement learning algorithm and each learning module has its own goal defined by a reward function. However, on a more abstract level the two approaches are similar as demonstrated in the work by Niv, Joel, Meilijson, and Ruppin (2002) and Ruppin (2002). They evolved optimal learning rules for a reinforcement learning agent, where the learning occurred within a generalized Hebbian learning framework.

This study builds on our earlier embodied evolution study (Elfwing, Uchibe, Doya, & Christensen, 2005). The main differences between this study and the earlier work are that in the earlier study:

- there was no within-generation learning of the basic behaviors. Instead, the basic behaviors were learned in advance, by reinforcement learning;
- we used a rather ad-hoc selection scheme, where a virtual agent reproduced one offspring at the end of its lifetime. The offspring then replaced the parental agent in the subpopulation;
- the fitness information (the number of captured batteries) had to be represented explicitly and communicated to the mating partners;
- standard centralized selection had significantly higher performance, compared with the proposed selection scheme;
- the performance was significantly decreased when the solutions obtained in simulation were transferred to hardware.

We are interested in the interaction of learning and evolution. In general, we have used reinforcement learning to learn the task at hand and the evolutionary search process to optimize meta-aspects of the learning, such as the metaparameters α and τ for a battery capturing task where the number of battery packs changed over the learning time (Eriksson, Capi, & Doya, 2003) and hierarchical learning structures (Elfwing, Uchibe, Doya, & Christensen, 2007). The method of coevolving the shaping rewards and the metaparameters is explored in our recent work (Elfwing et al., 2008) using the mountain-car task and a battery capturing task as testbeds.

There have been few studies conducted in the field of embodied evolution apart from the pioneering work by Watson et al. (2002). They used eight small mobile robots to evolve a very simple neural network controller for a phototaxis task. The controller had two input nodes. One binary input node, indicating which of the

two light sensors was receiving more light, and one bias node. The input nodes were fully connected to two output motor neurons, controlling the speed of the wheels, giving a total of four integer weights. In their experiments, mating was not a directed behavior. Instead, an agent broadcast its genotype according to a predefined scheme, and the other robots within the communication range could then pick up the genotype. Differential selection was achieved by having more successful agents broadcast their genotypes more often and were less inclined to accept genotypes broadcast by other agents. Nehmzow (2002) used embodied evolution to evolve three different sensorimotor behaviors, using two small mobile robots. In the experiments, the two robots first evaluated their current behavioral strategies, and after a fixed amount of time the robots initiated a robot seeking behavior. The robots then performed an exchange of genetic material via IR-communication, and genetic operations were applied according to the fitness values. Each robot stored two strings: the ''active'' one and the ''best'' one found so far. The latter was used as a fallback option if the genetic algorithm did not produce a fitter solution than the ''best'' so far. Usui and Arita (2003) used six Khepera robots to evolve an avoidance behavior. Their method is, though, more an ''island model'' parallel genetic algorithm than an embodied evolution method. Each physical robot ran an independent genetic algorithm for a subpopulation of virtual agents, where the virtual agents were evaluated by time-sharing. Migrated genotypes, broadcast by other robots, were re-evaluated for the new robot.

Funding

Stefan Elfwing's part of this research was supported by a shared grant from the Swedish Foundation for Internationalization of Research and Education (STINT) and the Swedish Foundation for Strategic Research (SSF). The funding is gratefully acknowledged.

References

Baldwin, J. (1896). A new factor in evolution. *American Naturalist*, *30*, 441–451.

Doya, K. (2002). Metalearning and neuromodulation. *Neural Networks*, *15*(4), 485–506.

Doya, K., & Uchibe, E. (2005). The Cyber Rodent project: Exploration of adaptive mechanisms for self-preservation and self-reproduction. *Adaptive Behavior*, *13*(2), 149–160.

Elfwing, S., Uchibe, U., Doya, K., & Christensen, H. I. (2005). Biologically inspired embodied evolution of survival. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC2005)* (Vol. 3)IEEE Press, Piscataway, NJ (pp. 2210–2216).

Elfwing, S., Uchibe, U., Doya, K., & Christensen, H. (2007). Evolutionary development of hierarchical learning structures. *IEEE Transactions on Evolutionary Computation*, *11*(2), 249–264.

Elfwing, S., Uchibe, U., Doya, K., & Christensen, H. (2008). Coevolution of shaping rewards and meta-parameters in reinforcement learning. *Adaptive Behavior*, *16*(6), 400–412.

Eriksson, A., Capi, G., & Doya, K. (2003). Evolution of meta-parameters in reinforcement learning algorithm. In C. S. G. Lee & J. Yuh (Eds.), *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS2003)* (pp. 412–417). Piscataway, NJ: IEEE Press.

Floreano, D., & Mondada, F. (1996). Evolution of plastic neurocontrollers for situated agents. In P. Maes, M. Mataric, J.-A. Meyer, J. Pollack & S. Wilson (Eds.), *Proceedings of the International Conference on Simulation of Adaptive Behavior (SAB1996)* (pp. 401–410).

Hinton, G., & Nowlan, S. (1987). How learning can guide evolution. *Complex Systems*, *1*, 495–502.

Laud, A., & DeJong, G. (2003). The influence of reward on the speed of reinforcement learning: An analysis of shaping. In T. Fawcett & N. Mishra (Eds.), *Proceedings of the International Conference on Machine learning (ICML2003)* (pp. 440–447).

Nehmzow, U. (2002). Physically embedded genetic algorithm learning in multi-robot scenarios: The PEGA algorithm. In C. G. Prince, Y. Demiris, Y. Marom, H. Kozima & C. Balkenius (Eds.), *Proceedings of the International Workshop on Epigenetic Robotics and Robotics (EPIROB2002)* (pp. 115–123).

Ng, A. Y., Harada, D., & Russell, S. J. (1999). Policy invariance under reward transformations: theory and application to reward shaping. In I. Bratko & S. Dzeroski (Eds.), *Proceedings of the International Conference on Machine learning (ICML1999)* (pp. 278–287). San Francisco, CA: Morgan Kaufmann Publishers Inc.

Niv, Y., Joel, D., Meilijson, I., & Ruppin, E. (2002). Evolution of reinforcement learning in uncertain environments: A simple explanation for complex foraging behaviors. *Adaptive Behavior*, *10*(1), 5–24.

Nolfi, S., & Floreano, D. (2000). *Evolutionary robotics. The biology, intelligence, and technology of self-organizing machines*. Cambridge, MA: MIT Press.

Rummery, G. A., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems*. (Tech. Rep. CUED/F-INFENG/TR 166). Cambridge University Engineering Department.

Ruppin, E. (2002). Evolutionary autonomous agents: A neuroscience perspective. *Nature Review Neuroscience*, *3*, 132–141.

Singh, S. P., & Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, *22*(1-3), 123–158.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In D. S. Touretzky, M. C. Mozer & M. E. Hasselmo (Eds.), *Advances in neural information processing systems 8* (pp. 1038–1044). Cambridge, MA: MIT Press.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Turney, P., Whitley, D., & Anderson, R. (1996). Introduction to the special issue: Evolution, learning, and instinct: 100 years of the Baldwin effect. *Evolutionary Computation*, *4*(3), iv–viii.

Urzelai, J., & Floreano, D. (2000). Evolutionary robotics: Coping with environmental change. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2000)* (pp. 941–948).

Usui, Y., & Arita, T. (2003). Situated and embodied evolution in collective evolutionary robotics. In M. Sugisaka & H. Tanaka (Eds.), *Proceedings of the International Symposium on Artificial Life and Robotics (AROB2003)* (pp. 212–215).

Watson, R., Ficici, S., & Pollack, J. (2002). Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, *39*(1), 1–18.

Whiteson, S., & Stone, P. (2006). Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, *7*, 877–917.

Wiewiora, E. (2003). Potential-based shaping and Q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, *19*, 205–208.

About the Authors

Stefan Elfwing received his M.Sc. degree in 2003 and Ph.D. degree in 2007 from the Royal Institute of Technology. Since 2007 he has been a researcher at the Neural Computation Unit, Initial Research Project, Okinawa Institute of Science and Technology. He does research on robotics with the focus on reinforcement learning and embodied evolutionary systems.

Eiji Uchibe received his Ph.D. degree in mechanical engineering from Osaka University in 1999. He worked as a research associate of the Japan Society for the Promotion of Science, in the Research for the Future Program titled Cooperative Distributed Vision for Dynamic Three-Dimensional Scene Understanding. Then he joined ATR as a researcher in 2001. Since 2004 he has been a researcher at the Neural Computation Unit, Initial Research Project, Okinawa Institute of Science and Technology. His research interests are in learning robots, reinforcement learning, evolutionary computation, and computational neuroscience, and their applications. *Address*: Neural Computation Unit, Initial Research Project, Okinawa Institute of Science and Technology, JST, 12–22 Suzaki, Uruma, Okinawa 904-2234, Japan. *E-mail*: uchibe@oist.jp.

Kenji Doya received his B.S. in 1984, M.S. in 1986, and Ph.D. in 1991, at the University of Tokyo. He became a research associate at the University of Tokyo in 1986, at the University of California, San Diego, in 1991, and at the Salk Institute in 1993. He joined ATR in 1994 and is currently the head of Computational Neurobiology Department, ATR Computational Neuroscience Laboratories. In 2004, he was appointed as a principal investigator of Initial Research Project, Okinawa Institute of Science and Technology. He is interested in understanding the functions of basal ganglia and neuromodulators based on the theory of reinforcement learning. *Address*: Neural Computation Unit, Initial Research Project, Okinawa Institute of Science and Technology, JST, 12–22 Suzaki, Uruma, Okinawa 904-2234, Japan. *E-mail*: doya@oist.jp.

Henrik I. Christensen is the KUKA Chair of Robotics and a Distinguished Professor of Computing at Georgia Institute of Technology. He is also the director of the Center for Robotics and Intelligent Machines. Dr. Christensen was awarded M.Sc. and Ph.D. degrees from Aalborg University, Denmark, in 1987 and 1989, respectively. He has since then held positions at Aalborg University, the University of Pennsylvania, and Swedish Royal Institute of Technology before moving to Georgia Tech in 1996. He was the founding coordinator of the European Network of Excellence in Robotics - EURON (1999–2006). Dr Christensen does research on systems integration, mapping, and estimation. He serves on numerous editorial boards and is editor in chief of Foundation and Trends in Robotics journal. He has published more than 250 publications and is the cofounder of four companies. *Address*: Centre for Autonomous Systems, Numerical Analysis and Computer Science, Royal Institute of Technology (KTH), S-10044 Stockholm, Sweden. *E-mail*: hic@kth.se.

Appendix A Embodied Evolution Implementation

Algorithm 1: The embodied evolution frame work.

Constants: N_{va}, number of virtual agents in the subpopulations.
N_{ts}, number of time-sharings per generation.
T_{ts}, number of time steps per time sharing.
E_{init}, the initial energy level of the virtual agents.
Variables : $\boldsymbol{A}$, boolean list of alive virtual agents of size N_{va}.
$\boldsymbol{E}$, the energy levels of the N_{va} virtual agents.
$\boldsymbol{G}$, the N_{va} genotypes in the subpopulation.
G_{rec}, genotype received from another robot.
$\boldsymbol{\Theta}$, the learning parameters of the N_{va} virtual agents.
$\boldsymbol{O}$, list of reproduced potential offspring.

```
1  Initialize the genotypes, G, randomly.
2  while termination condition is not fulfilled do
3      Initialize the learning parameters, Θ, arbitrarily.
4      O ← []
5      for n ← 1 to N_va do
6          A_n ← true
7          E_n ← E_init
       /* Evaluation:  N_ts time-sharings of T_ts time steps.  */
8      for i ← 1 to N_ts do
9          for n ← 1 to N_va in random order do
10             if A_n then
11                 for t ← 1 to T_ts do
12                     [Θ_n, E_n, G_rec ] ← Controller( G_n, Θ_n, E_n, t)
                       /* Reproduce potential offspring?          */
13                     if ExistGen(G_rec) & ReprodCond() then
14                         [o_1, o_2 ] ← GenOps(G_n, G_rec)
15                         AppChild(o_1, o_2).
                       /* Kill virtual agent?                     */
16                     if E_n < 0 then
17                         A_n ← false
18                         break
       /* New generation.                                         */
19     G ← randomly select min(N_va,length(O)) offspring from O
20     if length(O) < N_va then
21         for n ← length(O)+1 to N_va do
22             G_n ← random genotype
```

Algorithm 1 shows the pseudocode of the implementation of the embodied evolution framework, which runs in each subpopulation. The genotypes, $\boldsymbol{G}$, consist of the parameters that either control the virtual agents' behaviors directly or modulate the learning of their behaviors. Before the evolution, the genotypes are randomly initialized (line 1). The evolution runs until a predefined termination condition is fulfilled (line 2), for example, in our experiments the evolution was run until it produced 400 generations. At the beginning of each generation, the learning parameters $\boldsymbol{\Theta}$ are arbitrarily initialized (in the case of lifetime learning), the list of reproduced potential offspring $\boldsymbol{O}$ is initially assigned to be empty, and all virtual agents are set to be alive ($A_n \leftarrow$ **true**) and their internal energies are set to a predefined level E_{init} (lines 3–7).

In each time step, the embodied evolution framework invokes the control architecture to execute an action, update the learning parameters $\boldsymbol{\Theta}_n$, and then updates the energy level E_n of the virtual agent that controls the robot (line 12). If the virtual agent performs a successful mating, then the controller returns the genotype of the mating partner, G_{rec}. If the virtual agent successfully exchanged genotypes with a mating partner and if the predefined reproduction condition is satisfied (i.e., if both the `ExistGen()` and the `ReprodCond()` functions return true in line 13), then two potential offspring are created and they are appended to $\boldsymbol{O}$ (lines 14–15). If the energy of the virtual agent is depleted, then the virtual agent is killed and it will not be evaluated for any more time steps (lines 16–17). After all virtual agents have survived or died premature death, a new subpopulation is created by randomly selecting N_{va} potential offspring from $\boldsymbol{O}$ (line 19). If the number of reproduced potential offspring is less than N_{va}, then the remaining part of the new population is created by adding randomly created genotypes (lines 20–21).

Appendix B Control Architecture Implementation

Algorithm 2: The two layered control architecture.

Constants: N_m, number of learning modules.
N_x, dimension of the neural network input state space.
Input : $\boldsymbol{w}$, weights of the neural network ($N_m \times N_x$).
$\boldsymbol{v}^i$, shaping reward parameters of the ith module.
mp, meta parameters modulating the learning.
$\boldsymbol{\Theta}^i$, learning parameters of the ith module.
E, current energy level.
t, time step of the current time sharing.
Variables : $\boldsymbol{x}$, neural network input state ($N_x \times 1$).
G_{rec}, genotype received from another robot.
m, selected learning module (static).

1 **Function** `Controller()`
  `/* First time step in the time-sharing? */`
2 **if** $t = 1$ **then**
3 $m_{old} \leftarrow$ -1
4 **else**
5 $m_{old} \leftarrow m$ `/* m is preserved between calls */`
6 $\boldsymbol{x} \leftarrow$ observe current state
7 $m \leftarrow$ `argmax`$(\boldsymbol{wx})$
8 $\boldsymbol{\Theta}^m \leftarrow$ `RLAlg`$(\boldsymbol{\Theta}^m, \boldsymbol{v}^m, mp, m, m_{old})$
9 **if** *a successful mating occurred* **then**
10 $G_{rec} \leftarrow$ received genotype
11 **else**
12 $G_{rec} \leftarrow []$
13 $E \leftarrow$ `UpdateEnergy`(E)
14 **return** $\boldsymbol{\Theta}$, E, G_{rec}

Algorithm 2 shows the pseudocode of the proposed control architecture. The selection of reinforcement learning modules is controlled by a top-layer linear feed-forward neural network. In each time step, the neural network selects the learning module with the largest output, m (line 7 in Algorithm 2). The selected module then executes an action based on its learned behavior (line 8). It also updates its learning parameters, $\boldsymbol{\Theta}^m$, based on the global reward function (predefined and equal for all modules), the metaparameters, mp (shared by all modules and genetically determined), and its shaping reward function (unique for each module and genetically determined by $\boldsymbol{v}^m$). After invoking the selected learning module, the algorithm checks if a successful mating occurred (lines 9–12), and updates the energy level of the virtual agent (line 13). In relation to the implementation to the embodied evolution framework (Algorithm 1), the weights of the neural network, $\boldsymbol{w}$, the parameters of the shaping rewards, $\boldsymbol{v}$, and the metaparameters, mp, correspond to the genotype, G_i, of the virtual agent, i, controlling the robot.

Appendix C Reinforcement Learning Implementation

Algorithm 3: Sarsa (λ) with tile coding, replacing traces with the optional clearing and potential-based shaping rewards.

Constants: n_t^m, number of features (tiles) of module m.
N_t^m, number of tilings of module m.
Input : $\boldsymbol{\Theta}^m$, parameter matrix of module m ($n_t^m \times |\mathcal{A}^m|$).
$\boldsymbol{v}^m$, shaping reward parameters of module m.
α, learning rate.
γ, discount factor of future rewards.
λ, trace-decay rate.
m, current selected learning module.
m_{old}, previous selected learning module.
Variables : $\mathbf{e}$, traces for all features and actions ($n_t^m \times |\mathcal{A}^m|$).
$\boldsymbol{I}_s$, Indexes of nonzero features in state s ($N_t^m \times 1$).
$\boldsymbol{Q}_s$, Q-values for actions $a \in \mathcal{A}^m$, $\boldsymbol{a}$, in state s ($|\mathcal{A}^m| \times 1$).

`/* e, s, I_s, Q_s, and a are preserved between calls. */`
1 **Function** `RLAlg()`
  `/* Changed learning module? */`
2 **if** $m \neq m_{old}$ **then**
3 $\mathbf{e} \leftarrow \mathbf{0}$
4 $s \leftarrow$ observe current state for module m
5 $\boldsymbol{I}_s \leftarrow$ set of nonzero features present in s
6 $\boldsymbol{Q}_s \leftarrow \sum_{i \in \boldsymbol{I}_s} \Theta^m(i, \boldsymbol{a})$
7 Select action $a \in \mathcal{A}^m$ using policy derived from $\boldsymbol{Q}_s$
8 $\mathbf{e} \leftarrow \gamma\lambda\mathbf{e}$ `/* update traces */`
9 **foreach** $i \in \boldsymbol{I}_s$ **do**
10 $e(i, a) \leftarrow 1$ `/* replacing traces */`
11 **forall** $a' \in \mathcal{A}^m$ **do**
12 **if** $a' \neq a$ **then**
13 $e(i, a') \leftarrow 0$ `/* optional clearing */`
14 Execute action a, observe global reward r, and next state s'
15 $\delta \leftarrow r - \sum_{i \in \boldsymbol{I}_s} \Theta^m(i, a)$ - $\Phi(s, \boldsymbol{v}^m)$
16 $s \leftarrow s'$
17 $\boldsymbol{I}_s \leftarrow$ set of nonzero features present in s
18 $\boldsymbol{Q}_s \leftarrow \sum_{i \in \boldsymbol{I}_s} \Theta^m(i, \boldsymbol{a})$
19 Select next action $a \in \mathcal{A}^m$ using policy derived from $\boldsymbol{Q}_s$
20 $\delta \leftarrow \delta + \gamma\left(\sum_{i \in \boldsymbol{I}_s} \Theta^m(i, a) + \Phi(s, \boldsymbol{v}^m)\right)$
21 **for** $i \leftarrow 1$ **to** n_t^m **do**
22 **forall** $a' \in \mathcal{A}^m$ **do**
23 $\Theta^m(i, a') \leftarrow \Theta^m(i, a') + \alpha\delta e(i, a')/N_t^m$
24 **return** $\boldsymbol{\Theta}^m$

Algorithm 3 shows the pseudocode for Sarsa(λ) with tile coding, replacing traces with optional clearing, and potential-based shaping rewards, used in our embodied evolution context. If a new learning module is selected, then the traces, $\mathbf{e}$, the current state, s, the set of nonzero tiles, $\boldsymbol{I}_s$, and the action, a, are re-initialized (lines 2–7). The shaping reward is added to TD-error, δ, by subtracting the potential value, $\Phi()$, of the current state in line 15 and adding the potential value of the next state times γ in line 20. In relation to the control architecture, α, γ, λ, and any parameters modulating the action selection in lines 7 and 19 (the temperature, τ, in softmax selection, in our case) corresponds to the metaparameters, mp, in Algorithm 2.