

A comparison of controller architectures and learning mechanisms for arbitrary robot morphologies

Jie Luo¹, Jakub Tomczak², Karine Miras¹, and Agoston E. Eiben¹

Computer Science Dept.

¹Vrije Universiteit Amsterdam, Netherlands

²Eindhoven University of Technology, Netherlands

Email: j2.luo@vu.nl

Abstract—The main question this paper addresses is: What combination of a robot controller and a learning method should be used, if the morphology of the learning robot is not known in advance? Our interest is rooted in the context of morphologically evolving modular robots, but the question is also relevant in general, for system designers interested in widely applicable solutions. We perform an experimental comparison of three controller-and-learner combinations: one approach where controllers are based on modelling animal locomotion (Central Pattern Generators, CPG) and the learner is an evolutionary algorithm, a completely different method using Reinforcement Learning (RL) with a neural network controller architecture, and a combination ‘in-between’ where controllers are neural networks and the learner is an evolutionary algorithm. We apply these three combinations to a test suite of modular robots and compare their efficacy, efficiency, and robustness. Surprisingly, the usual CPG-based and RL-based options are outperformed by the in-between combination that is more robust and efficient than the other two setups.

Index Terms—evolutionary robotics, Reinforcement learning, controller, learning algorithm, CPG

I. INTRODUCTION

Enabling robots to learn tasks automatically is an important feature on its own, and also necessary within an evolutionary robot system, where both the morphologies (bodies) and the controllers (brains) are developed by evolution. In such systems, ‘newborn’ robots should undergo a learning phase to fine-tune the inherited brain to the inherited body quickly after birth [1], [2]. This raises the question: what combination of a robot controller and a learning method should be used in the robots’ morphology which is not known in advance? In general, a robot’s ability to learn a task depends on three major system components, namely, the body (morphology, hardware), the brain (controller, software), and the learning algorithm.

In the current literature, the majority of studies investigate controller optimization using multiple learning algorithms, but focusing on a specific control architecture [3], [4]; comparisons of different control architectures and learning methods for learnable controllers and arbitrary modular robots are rarely carried out.

This study makes a step towards closing this gap by comparing three different combinations of a specific control architecture and a learning algorithm. The possible control architectures are Central Pattern Generator (CPG), Artificial Neural Network (ANN) and Deep Reinforcement Learning

(DRL) policy controller. The possible learning algorithms are Reversible Differential Evolution (RevDE) [5] representing semi-supervised learning and Proximal Policy Optimization (PPO) [6] representing reinforcement learning. The combinations we compare here are CPG+RevDE, DRL+PPO, and ANN+RevDE. The motivation behind these choices is as follows. Using CPGs is a well-established, biologically plausible option to control modular robots actuated through joints, where learning can be performed by any heuristic black-box optimization method. RevDE is one such method that proved to be successful in the past for this application. Deep Reinforcement Learning is also a straightforward and increasingly popular option for robot learning with implications for the appropriate controller architecture, namely the use of ANNs. Additionally, we test the ANN+RevDE combination as an ‘in-between’ option that, to our best knowledge, has not been investigated previously.

The main contribution of this work is threefold:

- 1) It demonstrates a test-suite based approach to experimental research into robot learning, where the robots that make up the test suite are not only hand-picked, but also generated algorithmically.
- 2) Furthermore, the controller-and-learner combinations are not only compared by the usual performance measures, efficiency and efficacy, but also by robustness, i.e., stability or consistency over the different robot morphologies.
- 3) It provides an empirical assessment of three options, including two ‘usual suspects’ that researchers in the field are likely to consider: CPG-based controllers with a good weight optimizer and a Deep Reinforcement Learning method. The results indicate a surprising outcome, both of these methods are outperformed by the third one, ANN+RevDE.

II. RELATED WORK

A. Robot Controllers

A popular class of controllers is based on utilizing Artificial Neural Networks (ANN). The optimization of an ANN is typically done by approximating gradients for gradient-based methods or by applying derivative-free methods to alter internal weights and biases of all neurons within the ANN [7]–[13]. Alternatively, reinforcement learning (RL) could be used to

update the controller [14]–[16]. Here, we focus on one specific implementation of RL that utilizes two networks: a controller network (also called a policy controller), and a surrogate model, an additional neural network – a critic network – to update the parameters of the controller.

A popular approach relies on the idea inspired by biology that aims at creating rhythmic patterns to control the motion of the robots. These approaches use different controller types and learning algorithms for creating rhythmic patterns. Early approaches used Control Tables [17], [18], where each column of a table contains a set of actions for a module in the configuration, and Simple Sinusoidal, in which a specific sinusoidal function is utilized for each motor providing an easy way to parameterize a control pattern [19]–[21]. These methods were followed by a controller architecture called Cyclic Splines [22], [23] in which a spline is fitted through a set of action points in time to define a periodic control sequence (*i.e.* control policy). Another successful (bio-inspired) controller called Central Pattern Generators (CPGs) [24] was based on the spinal cord of vertebrates and can produce stable and well-performing gaits on both non-modular robots [25], [26] and modular robots [26]–[28]. CPGs are biological neural circuits that produce rhythmic output in the absence of rhythmic input [29]. In this work, we use CPGs to parameterize a controller and create biologically plausible motion patterns.

B. Controller Learning Algorithms

The problem of controller learning in robotics could be phrased as the *black-box optimization* problem [30], [31] since we need to either run a simulation or a physical robot to obtain a value of the objective function (or the fitness function). There is a vast amount of literature on learning algorithms on only one type of controller [4], [32]–[34], naming only a few.

In [33], a comparison of three learning algorithms in modular robots is performed where *NIP-Evolutionary Strategies*, *Bayesian Optimization* and *Reversible Differential Evolution* (RevDE) [35] are tested. The outcome of this study indicates that the shape of the fitness landscape in evolutionary strategies hints at a possible bias for morphologies with many joints. This could be an unwanted property for the implementation of lifetime learning because an algorithm should work consistently on different kinds of morphologies. Bayesian Optimization is good at sample efficiency, however, it requires much more time compared to the other two methods due to the higher time complexity (cubic complexity). The best-performing algorithm in this comparison was RevDE which scales well in terms of complexity and generalizes well across various morphologies. Therefore, we use RevDE in this paper. Moreover, we apply Proximal Policy Optimization (PPO) in the context of RL. PPO is a family of model-free RL learning algorithms that search the space of policies rather than assigning values to state-action pairs [36]. It was used in recent research [37] and performs well across various morphologies.

III. METHODOLOGY

A. Robot Controllers

In this research, our task is gait learning, therefore the controllers we use are open-loop controllers without steering. The choice of a robot controller is a crucial design decision and determines the resulting search space and, as a consequence, the behaviour of a robot. Different types of controllers may require different inputs, *e.g.* DRL-Policy controller and ANN controller need observations from the environment as input, however, CPG does not reply on observation in an open-loop controller. Moreover, the number of parameters to be optimized in each type of controller can be different. Last but not least, the outputs differ too. CPG and ANN controllers output actions to the hinges directly while the DRL-Policy controller output the action distribution.

1) *CPG controller*: Each robot hinge i is associated with a CPG that is defined by three neurons: a x_i -neuron, a y_i -neuron, and an out_i -neuron, which are recursively connected to produce oscillatory behaviour.

The CPG network structure we used has two layers:

- 1) Internal connection: The change of the x_i and y_i neurons' states with respect to time is calculated by multiplying the activation value of the opposite neuron with a weight. To reduce the search space, we define $w_{x_i y_i}$ to be $-w_{y_i x_i}$ and call their absolute value w_i and set $w_{x_i o_i} = 1$. The resulting activations of neurons x and y are periodic and bounded. The initial states of all x and y neurons are set to $\frac{\sqrt{2}}{2}$ because this leads to a sine wave with amplitude 1, which matches the limited rotating angle of the joints.
- 2) External connection: CPG connections between neighbouring hinges. Two hinges are said to be neighbours if their tree-based distance (how many edges between one node and the other) is less than or equal to two. x neurons depend on neighbouring x neurons in the same way as they depend on their y partner. Let i be the number of the hinge, N_i the set of indices of hinges neighbouring hinge i , and w_{ij} the weight between x_i and x_j . Again, w_{ji} is set to be $-w_{ij}$. The extended system of differential equations is then:

$$\begin{aligned}\dot{x}_i &= w_i y_i + \sum_{j \in N_i} w_{x_j x_i} x_j \\ \dot{y}_i &= w_i x_i\end{aligned}\tag{1}$$

Because of this addition, x neurons are no longer bounded between $[-1, 1]$. To achieve this binding, we use a variant of the sigmoid function, the hyperbolic tangent function (tanh), as the activation function of out_i -neurons.

The total number of the weights parameters per robot we have to optimise for the CPG network is the sum of weights of these two connections: $CPG_N_{param} = N_{hinges} + N_i$. Take the spider for example, it has 8 CPGs (hinges) and 10 pairs of neighbouring connections between CPGs, therefore the total number of the weights parameters is 18.

2) *ANN controller*: In an ANN robot controller, the ANN internally connects an input layer of neurons to an output layer that triggers the actuators, possibly via a layer of hidden neurons. The output of a previous layer is multiplied by corresponding weights before being summed with a bias term and thus serves as input for the next layer. Here, the main components of the ANN (a.k.a. Actor network) are:

- 1) *Single Observation Encoders*. A sub-network for encoding a single type of observation. In our research, we use two types of observations: state of each hinge (activation of the hinge between -1 and 1 which is its motion range) and the orientation of the robot (based on the core modular of the robot). The input of the coordinates observation network is $N_{\text{hinges}} \cdot 3$ dimensions and the input of the orientation observation network which is 4 dimensions. The output of both networks is a 32-dimensional vector through a linear layer followed by a tanh activation function.
- 2) *Observation Encoder* A network that concatenates the encoded observations. It receives inputs from the two Single Observation Encoders and passes the encoded observations which are $[32+32=64]$ dimensional through a linear layer followed by a tanh activation function to produce the final output of a 32-dimensional vector.
- 3) *Actor* Takes the concatenated encoded observations as input and outputs the action to be taken by the robot. The dimension of the action is based on the number of the robot's hinges.

The total number of parameters per robot to be optimized is equal to the sum of the parameters of the Single Observation Encoder, Observation Encoder, and Actor: $ANN_N_{\text{param}} = 32 \cdot (N_{\text{hinges}} \cdot 3 + 4 + 1) + 32 \cdot (64 + 1) + N_{\text{hinges}} \cdot (32 + 1)$.

3) *DRL-Policy controller*: The Deep Reinforcement Learning (DRL) paradigm provides a way to learn efficient representations of the environment from high-dimensional sensory inputs, and use these representations to interact with the environment in a meaningful way. At each time-step, the robot senses the world by receiving observations o_t provided by the simulator, then it takes an action a_t , and is given a reward r_t . A policy $\pi_{\theta}(a_t | o_t)$ models the conditional distribution over action $a_t \in A$ given an observation $o_t \in O(s_t)$. The goal is to find a policy which maximizes the expected cumulative reward R under a discount factor $\gamma \in (0, 1)$.

Policy controller Policy π_{θ} , as the robot's behaviour function, tells us which action to take in state s . In our research, the implementation of the policy controller has an Actor network, a Critic network, and an ActorCritic network that merges the two.

- 1) *Actor network*: a deep neuron network that outputs a Gaussian distribution over the possible actions given an observation. Similar to the ANN controller, observations are encoded into a 32-dimensional vector, but instead of producing actions directly, it produces the action probability using two hidden layers (mean_layer and std_layer).

- 2) *Critic network*: a deep neuron network which outputs a single scalar value that approximates the expected return of the current state of the input observation.
- 3) *ActorCritic network*: the primary component that combines the Actor and Critic networks and allows for sampling actions or computing their probabilities and the value of an observation. It can either output the action distribution, the state-value function or both, along with the log-probability of the actions taken and the entropy of the action distribution.

The robot chooses its action via the policy π_{θ} where θ are the parameters of these three NNs which will be optimized by a DRL algorithm called Proximal Policy Optimization (PPO).

The total number of parameters per robot we have to optimize for ActorCritic Network is equal to the sum of the parameters of the Actor, Critic, and ObservationEncoder sub-modules: $DRL_N_{\text{param}} = (N_{\text{hinges}} \cdot (32 + 1) + 2 \cdot N_{\text{hinges}} \cdot (N_{\text{hinges}} + 1)) + (1 \cdot 32 + 1) + (32 \cdot (N_{\text{hinges}} \cdot 3 + 4 + 1) + 32 \cdot (64 + 1))$.

B. Learning Methods

The problem of learning a robot controller is stated as a maximization problem of a function (reward or fitness) that is non-differentiable and could be given only after running a real-world experiment or a simulation. Since we cannot calculate the gradients concerning the controller weights, we must apply other learning methods that either utilize approximate gradients (e.g., through surrogate models) or derivative-free methods. In the following paragraphs, we present details of a specific derivative method (RevDE) and an instance of RL (PPO).

1) *RevDE*: In a recent study on modular robots [33], it was demonstrated that Reversible Differential Evolution (RevDE) [5], an altered version of Differential Evolution, performs and generalizes well across various morphologies. This method works as follows [35]:

- 1) Initialize a population with μ samples (n -dimensional vectors), \mathcal{P}_{μ} .
- 2) Evaluate all μ samples.
- 3) Apply the reversible differential mutation operator and the uniform crossover operator.

The reversible differential mutation operator: Three new candidates are generated by randomly picking a triplet from the population, $(\mathbf{w}_i, \mathbf{w}_j, \mathbf{w}_k) \in \mathcal{P}_{\mu}$, then all three individuals are perturbed by adding a scaled difference in the following manner:

$$\begin{aligned} \mathbf{v}_1 &= \mathbf{w}_i + F \cdot (\mathbf{w}_j - \mathbf{w}_k) \\ \mathbf{v}_2 &= \mathbf{w}_j + F \cdot (\mathbf{w}_k - \mathbf{v}_1) \\ \mathbf{v}_3 &= \mathbf{w}_k + F \cdot (\mathbf{v}_1 - \mathbf{v}_2) \end{aligned} \quad (2)$$

where $F \in R_+$ is the scaling factor. New candidates y_1 and y_2 are used to calculate perturbations using points outside the population. This approach does not follow the typical construction of an EA where only evaluated candidates are mutated.

The uniform crossover operator: Following the original DE method [38], we first sample a binary mask

$\mathbf{m} \in \{0,1\}^D$ according to the Bernoulli distribution with probability CR shared across D dimensions, and calculate the final candidate according to the following formula:

$$\mathbf{u} = \mathbf{m} \odot \mathbf{w}_n + (1 - m) \odot \mathbf{w}_n. \quad (3)$$

Following general recommendations in literature [39] to obtain stable exploration behaviour, the crossover probability CR is fixed to a value of 0.9 and the scaling factor F is fixed to a value of 0.5.

- 4) Perform a selection over the population based on the fitness value and select μ samples.
- 5) Repeat from step (2) until the maximum number of iterations is reached.

As explained above, we apply RevDE here as a learning method for our robot zoo. In particular, it will be used to optimize the weights of the CPGs and the parameters of ANN controllers of our modular robots for the task.

2) *PPO*: We use the Proximal Policy Optimization (PPO) [6] algorithm to optimize a policy. It improves training stability by using a clipped surrogate objective enforcing a divergence constraint on the size of the policy update at each iteration so that the parameter updates will not change the policy too much per step. Let us denote the probability ratio between old and new policies as follows:

$$r(\theta) = \frac{\pi_{\theta}(a_t|o_t)}{\pi_{\theta_{old}}(a_t|o_t)} \quad (4)$$

Then, the objective function of PPO (on policy) is the following:

$$L^{CLIP}(\theta) = \mathbb{E}_{s,a}[\min\{r_t(\theta)\hat{A}(s,a), \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}(s,a)\}] \quad (5)$$

where \hat{A} is an estimate of the advantage function. PPO imposes its constraint by enforcing a small interval around 1, $[1 - \epsilon, 1 + \epsilon]$ to be exact, where ϵ is a hyperparameter. The function $\text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon)$ clips the ratio to be no more than $1 + \epsilon$ and no less than $1 - \epsilon$. The objective function of PPO takes the minimum of the original value and the clipped version, and thus we lose the motivation for increasing policy updates to extremes for better rewards. We use Generalized Advantage Estimation (GAE) [40] to estimate the advantage function \hat{A} . We adopt an open-source implementation of PPO [41] for our research.

C. Frameworks: control architecture + learning method

We consider three combinations (frameworks) of control architectures and learning methods. The set-up of these three frameworks is shown in Figure 1.

First, we use CPG-based controllers trained by a derivative-free method RevDE (see Figure 1-a). Second, we consider an MLP-based ANN controller trained by the same learner RevDE (see Figure 1-b). Lastly, we use a DRL-policy based controller trained by PPO (see Figure 1-c).

CPGs are not combined with PPO because our CPGs do not receive inputs (states). ANN and DRL are somewhat equivalent because they both are ANN-based controllers, however, DRL has one extra Critic NN therefore more parameters to be optimized. The outputs of the actor network in these two controllers are different too.

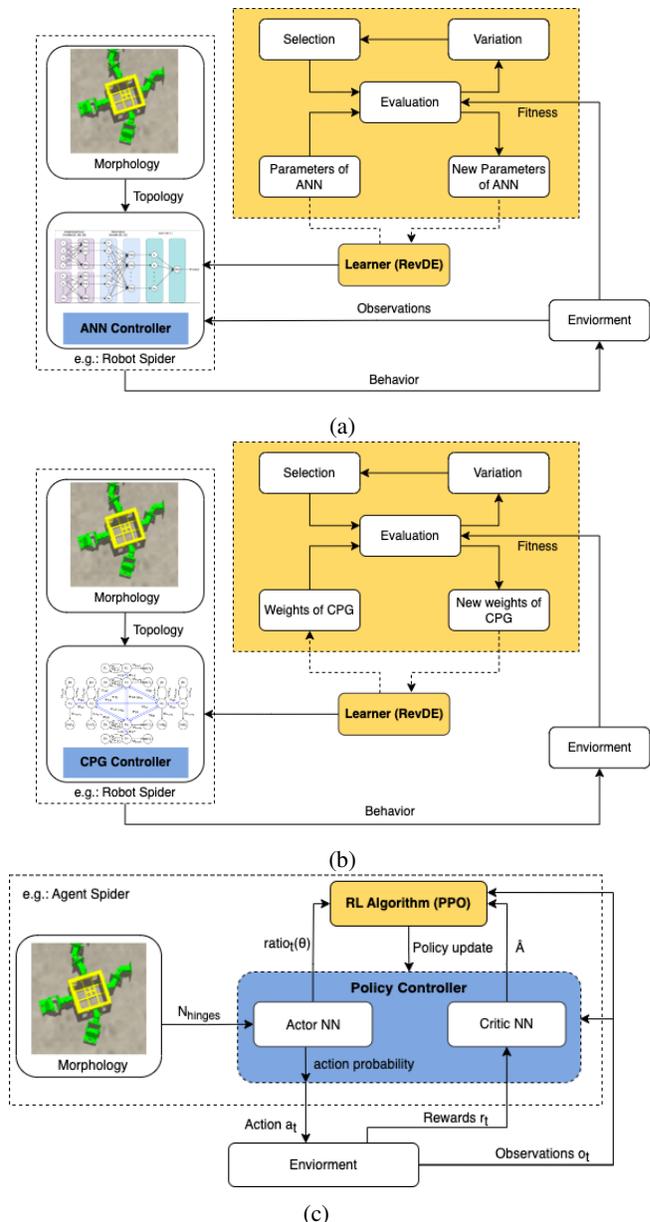


Fig. 1: Schematic representations of three learning controller frameworks. The blue boxes show the controllers and the yellow boxes show the learners. In 1-(a), we show a spider as an example of robot morphology. The topology of the morphology determines the topology of the controller. The learner RevDE optimizes the weights of the CPG controller. In 1-(b), the learner RevDE optimizes the parameters of the ANN controller. 1-(c) is a DRL framework using PPO as the learning algorithm to change the parameters of two deep NNs to improve the policy controller for the tasks.

D. Test suite of robot morphologies

Given a set of robot modules and ways to attach them to functional robots, the number of possible configurations (thus, the number of possible robot morphologies) is, in general, infinite. For practically feasible empirical research, we need a limited set of robots to serve as test cases. In this paper, we use a test suite of twenty robots made of two parts: a set of viable and diverse robots produced by an evolutionary process, and a set of hand-picked robots [33]. Regarding the first part, the key is to apply task-based fitness (viability) together with novelty search (diversity). The second part of the test suite can be filled by robots added manually by the experimenter. This option is entirely optional, it is to accommodate subjective preferences and interest in particular robot designs. The test suite we use here was generated by evolving a population of 500 robots for speed and novelty w.r.t. the *k-nearest neighbours* in the morphological space [42]. After termination, 15 out of the 500 robots were selected by maximizing the pairwise Euclidean distance in the morphology space. The other five robots (Gecko, Snake, Spider, BabyA, BabyB) were added manually. The robots are shown as inserts in Figure 5.

IV. EXPERIMENTAL SETUP

1) *Simulator*: We use a Mujoco simulator-based wrapper called Revolve2 to run the experiments. To have a fair comparison, we set the number of evaluations to be the same for each learner: 1000 learning evaluations. This number is based on the evaluations from RevDE for running 10 initial samples with 34 iterations. The first iteration contains 10 samples, and from the second iteration onwards each iteration creates 30 new candidates, resulting in a total of $10 + 30 \cdot (34 - 1) = 1000$ evaluations. Then with the same evaluation number 1000, we set PPO with 10 agents per iteration and 100 episodes. For the task of gait learning, we define the robot’s fitness as its average speed in 30s, i.e. absolute distance in centimetres per second (cm/s).

2) *Setups and Code*: The code for carrying out the experiments is available online: <https://shorturl.at/gozS3>. A video showing examples of robots from the experiments can be found in <https://shorturl.at/gGHR3>. Table I shows the set-up of the experiments. The specific values of the hyperparameters are presented in Table II.

TABLE I: Experiments

Experiment	Control Architecture	Learner
CPG+RevDE	CPG	RevDE
ANN+RevDE	ANN	RevDE
DRL+PPO	DRL-Policy	PPO

V. RESULTS

To compare the different frameworks, we consider three key performance indicators: efficiency, efficacy, and robustness to different morphologies.

TABLE II: Main experiment hyper parameters

CPG+RevDE	Value	Description
μ	10	Population size
N	30	New candidates per iteration
λ	10	Top-sample size
F	0.5	Scaling factor
CR	0.9	Crossover probability
Iterations	34	Number of iterations in RevDE
ANN+RevDE	Value	Description
μ	30	Population size
N	30	New candidates per iteration
λ	10	Top-samples size
F	0.5	Scaling factor
CR	0.9	Crossover probability
Iterations	34	Number of iterations in RevDE
DRL+PPO	Value	Description
γ	0.2	Discount gamma
ϵ	0.2	PPO clipping parameter epsilon
Entropy coefficient	0.01	Entropy coefficient
Value loss coefficient	0.5	Value loss coefficient
Episode	100	A sequence of states, actions and rewards
Agents	10	Number of agents per episode
Steps	150	Number of steps before training

1) *Efficacy*: The quality of a robot (fitness) is defined by the speed of the robot from the starting position to the stopping position within the simulation time. The efficacy of a method is defined by the mean maximum fitness, averaged over the 20 independent repetitions: First, the maximum fitness achieved at the end of the learning process (1000 evaluations) is calculated within each independent repetition. Second, these maximum values are averaged over the 20 independent repetitions.

In Figure 2, the dots indicate the maximum fitness in each evaluation (averaged over 20 runs). We can see that with the same learner (RevDE), the ANN controller outperforms the CPG controller significantly. ANN+RevDE achieves a two times higher fitness value compared to CPG+RevDE at the end of the 1000 evaluations. This could be due to CPGs producing more connected actions with fewer controller parameters, while ANNs have much more parameters to optimize and output the action probability instead of the action itself which produces actions that are very different even in subsequent time steps, eventually helping the exploration. The mean maximum fitnesses of ANN+RevDE and DRL+PPO have no significant difference initially but after evaluation 200, ANN+RevDE yields much higher fitness values than DRL+PPO.

Second, another way to measure the quality of the solution is by giving the same computational budget (number of evaluations) and measuring which method finds the best solution (highest fitness) faster. In Figure 3, it is more significantly different among these three frameworks at evaluations 600 and 1000 than 200. Given the evaluations of 200, DRL+PPO has the highest mean fitness value. While at the evaluations of 600 and 1000, ANN+RevDE surpasses DRL+PPO. With regards to CPG+PPO, the fitness increasing speed is slower than the

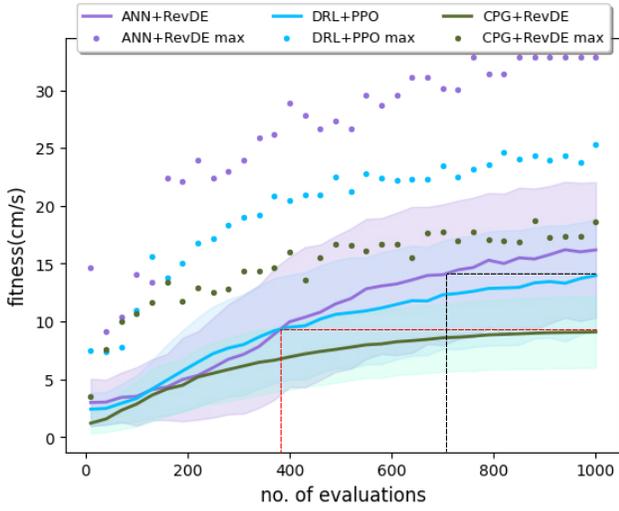


Fig. 2: Mean fitness over 1000 evaluations across morphologies (averaged over 20 runs) for 3 experiments. The dots indicate the mean maximum fitness in each evaluation (averaged over 20 runs). The shaded areas show the standard deviation.

other two methods.

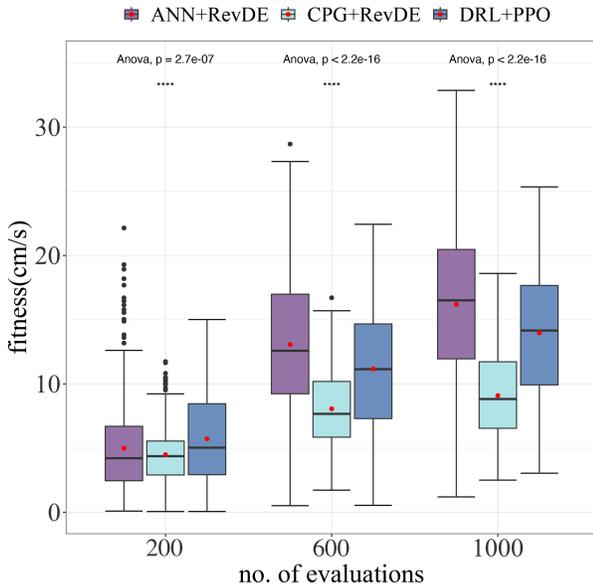
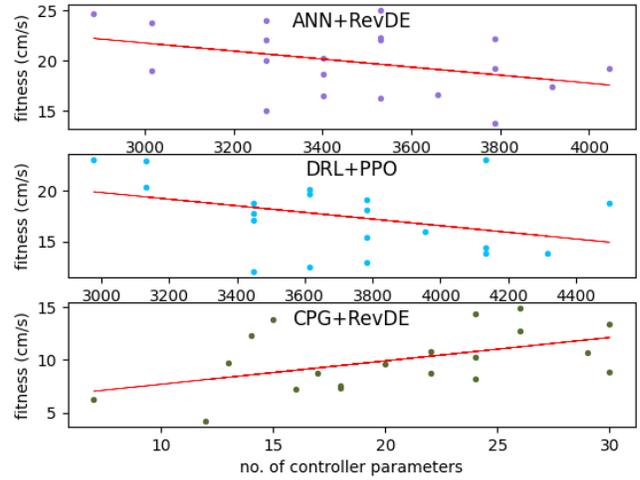


Fig. 3: Efficacy boxplot. Validation of three frameworks at three evaluations. Red dots show mean values.

2) *Efficiency*: Efficiency indicates how much effort is needed to reach a given quality threshold (the fitness level): it is measured as the average number of evaluations to ‘find a solution’. Figure 2 displays the usual quality-versus-effort plots, specifically the mean fitness over the number of evaluations. Looking at the solid curves reveals that ANN+RevDE is more efficient than the other methods. As marked by the red dotted lines, it takes only 370 evaluations for ANN+RevDE (purple curve) to reach the level of fitness that the CPG+RevDE method

Robot ID	Np_ANN	ANN+RevDE	Np_DRL	DRL+PPO	Np_CPG	CPG+RevDE	p-value
a	3272	19.99	3449	17.80	17	8.75	=3.7e-08
b	3530	22.29	3783	15.46	22	10.76	=2.2e-07
c	2885	24.65	2978	23.05	7	6.24	<2.2e-16
d	3530	16.24	3783	12.98	24	8.16	=4.9e-08
e	3014	23.73	3131	22.87	13	9.65	=3e-12
f	3401	20.25	3614	20.12	22	8.68	=8.2e-13
g	3788	19.21	4133	13.90	29	10.63	=3.2e-08
h	3788	13.79	4133	14.40	30	13.39	=0.52
i	3788	22.24	4133	22.98	26	14.90	=0.00015
j	3530	22.13	3783	18.08	24	14.33	=5.6e-11
k	3401	16.50	3614	12.53	16	7.23	=3.6e-07
l	4046	19.25	4499	18.76	24	10.24	=2.7e-06
m	3272	15.07	3449	12.04	15	13.83	=0.0062
n	3272	22.03	3449	17.13	18	7.30	=1.3e-12
o	3014	18.98	3131	20.37	12	4.22	<2.2e-16
p	3401	18.65	3614	19.63	18	7.59	=3.8e-18
q	3272	23.98	3449	18.75	14	12.35	=1.1e-15
r	3917	17.39	4314	13.90	30	8.86	=4.2e-05
s	3530	25.03	3783	19.08	20	9.56	=1.2e-14
t	3659	16.67	3956	16.03	26	12.75	=0.00017
	3466	19.90	3709	17.49	20	9.97	=2.3e-14

(a)



(b)

Fig. 4: (a) Mean maximum fitness per morphology for each framework. For each robot morphology, there’re columns of controller parameters numbers, namely Np_ANN in purple, Np_DRL in blue and Np_CPG in green and the best result which is indicated with boldface while underline indicates significantly better performance compared to the other frameworks. The last row shows the aggregated result for each framework over all morphologies. (b) the correlation between the number of controller parameters and the mean maximum fitness per framework. The red lines are the linear regression lines.

(green curve) achieves at the end of the learning period, 1000 evaluations. Similarly, the black dotted lines mark the number of evaluations at 730 when ANN+RevDE achieved the levels of fitness that DRL+PPO reached after 1000 evaluations.

3) *Robustness*: The robustness of a framework is defined by the variance in different robot morphologies. We can measure this by the variance of a framework’s mean maximum fitness over the robot zoo and the mean fitness per robot over the number of evaluations.

Figure 5 shows the mean and maximum fitness of three

frameworks over the number of evaluations per robot from Robot Zoo. The bands indicate the 95% confidence intervals ($\pm 1.96 \times SE$, Standard Error). CPG+RevDE has a narrower band than the other two frameworks which indicates lower uncertainty and is more stable. In Figure 4-a, we present a numerical summary of the results of the mean maximum fitness per robot per framework. It shows ANN+RevDE outperform CPG+RevDE or DRL+PPO on 16 robots significantly: a, b, c, d, e, f, g, j, k, l, m, n, q, r, s, t. DRL+PPO wins on robot i, o, p significantly and on robot h non-significantly.

Figure 4-b exhibits the correlation between the number of controller parameters and mean maximum fitness per framework. The dots in each plot represent 20 robot zoo. The results indicate that the deep neural network-based frameworks (ANN+RevDE and DRL+PPO) show a negative linear relationship between the number of controller parameters and the fitness value while the CPG-based framework shows a positive linear relationship. Among the frameworks, it shows that the significant difference in the number of controller parameters between the Deep NN-based and the CPG-based controllers does reflect on their fitnesses (different y-scales).

VI. CONCLUSIONS AND FUTURE WORK

This work investigated different combinations of control architectures with learning algorithms applied to a diverse set of robot morphologies.

Regarding efficacy and efficiency, the ANN+RevDE framework achieved levels of quality that the other two frameworks managed to achieve only at later stages of the learning period. As for robustness, all three frameworks successfully optimized all robots. However, ANN+RevDE outperformed DRL+PPO or CPG+RevDE significantly on 16 robots, while DRL+PPO outperformed on only 3 robots. Therefore ANN+RevDE is the best-performing learning controller framework in all three measures.

Interestingly, with the same learning algorithm (RevDE), the CPG controller performs more steadily with a lower standard deviation while the ANN controller takes longer to explore at the beginning, then it increases steeply with a much higher standard deviation (Figure 2 and 3). This can be due to the significant difference in the number of parameters in different controllers, but future research is needed to investigate this phenomenon.

REFERENCES

- [1] A. Eiben, N. Bredeche, M. Hoogendoorn, J. Stradner, J. Timmis, A. Tyrrell, and A. Winfield, "The triangle of life: Evolving robots in real-time and real-space," *Advances in Artificial Life, ECAL 2013*, 09 2013.
- [2] A. E. Eiben and E. Hart, "If it evolves it needs to learn," in *GECCO 2020 Companion - Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 2020, pp. 1383–1384.
- [3] F. van Diggelen, E. Ferrante, and A. E. Eiben, "Comparing lifetime learning methods for morphologically evolving robots," in *GECCO '21: Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 07 2021, pp. 93–94.
- [4] L. K. Le Goff, E. Buchanan, E. Hart, A. E. Eiben, W. Li, M. de Carlo, M. F. Hale, M. Angus, R. Woolley, J. Timmis, A. Winfield, and A. M. Tyrrell, "Sample and time efficient policy learning with CMA-ES and Bayesian Optimisation," in *The 2020 Conference on Artificial Life*, no. January, 2020, p. 2020.
- [5] E. Weglarz-Tomczak, J. M. Tomczak, A. E. Eiben, and S. Brul, "Population-based parameter identification for dynamical models of biological networks with an application to *saccharomyces cerevisiae*," *Processes*, vol. 9, no. 1, p. 98, 2021.
- [6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
- [7] W.-P. Lee, "Evolving robot brains and bodies together: An experimental investigation," *Journal of the Chinese Institute of Engineers*, vol. 26, no. 2, pp. 125–132, 2003.
- [8] J. B. Pollack and H. Lipson, "The golem project: Evolving hardware bodies and brains," in *Proceedings. The Second NASA/DoD Workshop on Evolvable Hardware*. IEEE, 2000, pp. 37–42.
- [9] J. B. Pollack, G. S. Hornby, H. Lipson, and P. Funes, "Computer creativity in the automatic design of robots," *Leonardo*, vol. 36, no. 2, pp. 115–121, 2003.
- [10] H. Lipson and J. Pollack, "Evolving physical creatures," in *Artificial Life VII: Proceedings of the seventh international conference on artificial life*, 2006, pp. 282–287.
- [11] H. H. Lund, "Co-evolving control and morphology with LEGO robots," in *Morpho-functional machines: the new species*. Springer, 2003, pp. 59–79.
- [12] K. Poikselkä, I. Vallivaara, and J. Röning, "Evolutionary robotics on LEGO NXT platform," in *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2015, pp. 1137–1144.
- [13] A. Ranganath, J. Gonzalez-Gomez, and L. M. Lorente, "A distributed neural controller for locomotion in linear modular robotic configurations," in *Proceedings of the 8th Workshop of RoboCity2030*, 2011, pp. 129–144.
- [14] J. S. Bhatia, H. Jackson, Y. Tian, J. Xu, and W. Matusik, "Evolution gym: A large-scale benchmark for evolving soft robots," 2022.
- [15] M. D'Angelo, B. Weel, and A. Eiben, "Online gait learning for modular robots with arbitrary shapes and sizes," 12 2013.
- [16] H. Shen, J. Yosinski, P. Kormushev, D. Caldwell, and H. Lipson, "Learning fast quadruped robot gaits with the rl power spline parameterization," *International Journal of Cybernetics and Information Technologies*, vol. 12, 09 2012.
- [17] J. C. Bongard, V. Zykov, and H. Lipson, "Resilient machines through continuous self-modeling," *Science*, vol. 314, pp. 1118 – 1121, 2006.
- [18] M. Yim, D. Duff, and K. Roufas, "Polybot: a modular reconfigurable robot," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 1, 2000, pp. 514–520 vol.1.
- [19] J. Gonzalez-Gomez and E. Boemo, "Motion of minimal configurations of a modular robot: sinusoidal, lateral rolling and lateral shift," in *Climbing and Walking Robots*. Springer, 2006, pp. 667–674.
- [20] J. Bruce, K. Caluwaerts, A. Iscen, A. P. Sabelhaus, and V. SunSpiral, "Design and evolution of a modular tensegrity robot platform," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 3483–3489.
- [21] A. Fafiã, F. Orjales, F. Bellas, R. Duro *et al.*, "First steps towards a heterogeneous modular robotic architecture for intelligent industrial operation," in *Workshop on Reconfigurable Modular Robotics at the IROS*, 2011.
- [22] J. Milan, D. C. Matteo, H. Elte, E. Panagiotis, O. Jakub, H. Evert, A. J. E., and E. A. E., "Real-world evolution of robot morphologies: a proof of concept," *Artificial life*, vol. 23, no. 2, pp. 206–235, 2017.
- [23] J. Kober and J. R. Peters, "Policy search for motor primitives in robotics," in *Advances in neural information processing systems*, 2009, pp. 849–856.
- [24] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: A review," *Neural Networks*, vol. 21, no. 4, pp. 642–653, 2008, robotics and Neuroscience.
- [25] D. Christensen, J. Larsen, and K. Stoy, "Fault-tolerant gait learning and morphology optimization of a polymorphic walking robot," *Evolving Systems*, vol. 5, 03 2013.
- [26] A. Sprowitz, R. Moeckel, J. Maye, and A. J. Ijspeert, "Learning to move in modular robots using central pattern generators and online optimization," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 423–443, 2008.

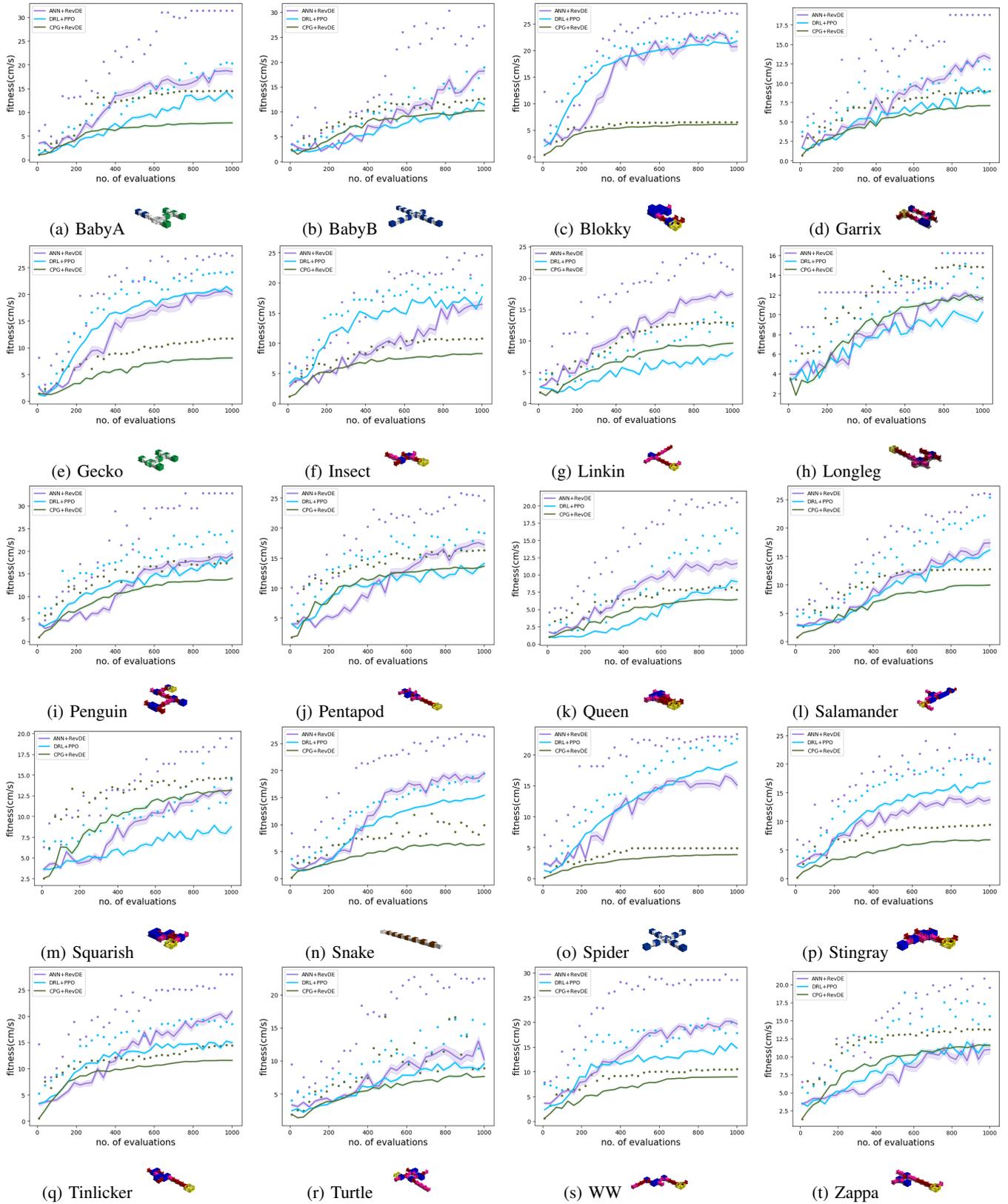


Fig. 5: Mean speed over time (the number of evaluations) during gait-learning, averaged over 20 independent repetitions per robot. The purple lines show the ANN+RevDE, the blue lines indicate DRL+PPO and the green lines indicate CPG+RevDE. The bands indicate the 95% confidence intervals ($\pm 1.96 \times SE$, Standard Error).

- [27] S. Pouya, J. Kieboom, A. Badri-Spröwitz, and A. Ijspeert, “Automatic gait generation in modular robots: to oscillate or to rotate? that is the question,” 11 2010, pp. 514 – 520.
- [28] J. Luo, A. Stuurman, J. M. Tomczak, J. Ellers, and A. E. Eiben, “The effects of learning in morphologically evolving robot systems,” *Frontiers in Robotics and AI*, vol. 5, 2022.
- [29] D. Bucher, G. Haspel, J. Golowasch, and F. Nadim, “Central Pattern Generators,” in *eLS*, 11 2015, pp. 1–12.
- [30] C. Audet and W. Hare, *Derivative-free and Blackbox Optimization*. Springer, 2017.
- [31] D. R. Jones, M. Schonlau, and W. J. Welch, “Efficient global optimization of expensive black-box functions,” *Journal of Global optimization*, vol. 13, no. 4, pp. 455–492, 1998.
- [32] B. Weel, M. D’Angelo, E. Haasdijk, and A. E. Eiben, “Online Gait Learning for Modular Robots with Arbitrary Shapes and Sizes,” *Artificial Life*, vol. 23, no. 1, pp. 80–104, 02 2017. [Online]. Available: https://doi.org/10.1162/ARTL_a_00223
- [33] F. van Diggelen, E. Ferrante, and A. E. Eiben, “Comparing lifetime learning methods for morphologically evolving robots,” <https://arxiv.org/abs/2203.03967>, 2021.
- [34] G. Lan, J. M. Tomczak, D. M. Roijers, and A. E. Eiben, “Time Efficiency in Optimization with a Bayesian-Evolutionary Algorithm,” in *Swarm and Evolutionary Computation*, 2 2020, p. 100970.
- [35] J. M. Tomczak, E. Weglarz-Tomczak, and A. E. Eiben, “Differential Evolution with Reversible Linear Transformations,” in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 2020, pp. 205–206.
- [36] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau *et al.*, “An introduction to deep reinforcement learning,” *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.
- [37] A. Gupta, S. Savarese, S. Ganguli, and L. Fei-Fei, “Embodied Intelligence via Learning and Evolution,” *Nature Communications*, vol. 12, 2021.
- [38] R. M. Storn, “Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces,” in *Journal of Global Optimization*, 1997, pp. 131–141.
- [39] M. Pedersen, “Good Parameters for Differential Evolution,” *Evolution*, pp. 1–10, 2010.
- [40] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, “High-dimensional continuous control using generalized advantage estimation,” 2015.
- [41] I. Kostrikov, “Pytorch implementations of reinforcement learning algorithms,” 2018.
- [42] K. Miras, E. Haasdijk, K. Glette, and A. Eiben, “Effects of selection preferences on evolved robot morphologies and behaviors,” in *Artificial Life Conference Proceedings*, MIT Press. Cambridge, MA: MIT Press, 2018, pp. 224–231.