The Effects of Adaptive Control on Learning Directed Locomotion

Diggelen, Fuda Van; Babuska, Robert; Eiben, A. E.

DOI
10.1109/SSCI47803.2020.9308557

Publication date
2020

Document Version
Final published version

Published in
Proceedings of the IEEE Symposium Series on Computational Intelligence, SSCI 2020

The Effects of Adaptive Control on Learning Directed Locomotion

Fuda van Diggelen
Technische Universiteit Delft,
The Netherlands
Email: f.vandiggelen@student.tudelft.nl

Robert Babuska
Technische Universiteit Delft,
The Netherlands
Email: r.babuska@tudelft.nl

A.E. Eiben
Vrije Universiteit Amsterdam,
The Netherlands
Email: a.e.eiben@vu.nl

***Abstract*—This study is motivated by evolutionary robot systems where robot bodies and brains evolve simultaneously. In such systems robot 'birth' must be followed by 'infant learning' by a learning method that works for various morphologies evolution may produce. Here we address the task of directed locomotion in modular robots with controllers based on Central Pattern Generators. We present a bio-inspired adaptive feedback mechanism that uses a forward model and an inverse model that can be learned on-the-fly. We compare two versions (a simple and a sophisticated one) of this concept to a traditional (open-loop) controller using Bayesian Optimization as a learning algorithm. The experimental results show that the sophisticated version outperforms the simple one and the traditional controller. It leads to a better performance and more robust controllers that better cope with noise.**

***Keywords*—Adaptive Control; Evolutionary Robotics; Reality Gap; Directed Locomotion**

I. Introduction

The behaviour of a robot is determined by its morphology ('body') and its controller ('brain') and the field of Evolutionary Robotics (ER) successfully demonstrated that good controllers can be evolved for various tasks [1], [2]. To maximize the potential of the evolutionary approach morphologies and controllers should be evolved together, but this implies particular challenges. Specifically, since the inherited brain of a newborn robot may not match its inherited body, evolution can end up with premature convergence of the robot population [3]. It has therefore been argued that a fully evolving robot system must contain a learning stage, directly following the 'birth' of offspring robots [4], [5]. In such a system, the design and optimization of good robots take place through two processes: 1) the evolutionary process, where both the body and the brain evolve to obtain a higher fitness, 2) a lifetime learning loop, where the brain is optimized to control a given body (produced by the evolutionary process) to obtain high task performance. This idea motivates the development of controllers that can be optimized quickly for a broad range of morphologies.

Currently, most controllers in ER are evolved on using an open-loop control architecture in simulation [6]. For real-world applications, this is clearly limiting, since in practice feedback control is often preferred. Unfortunately, feedback control can be hard to implement, especially when the environment and morphologies are not known beforehand. Many feedback mechanisms require a lot of engineering per robot, which is infeasible if robots are (re)produced automatically by an evolutionary process. Furthermore, PID controllers will likely fail in accommodating complex morphologies, as they are hard to design optimally and not well-suited for nonlinear systems [7]. In this paper, we address this problem by introducing an adaptive bio-inspired feedback controller called Internal Model Control (IMC), which can be used as an extension to already available designs. This allows for simple real robot implementation, while minimally affecting the way evolution improves the original controllers. To this end, we add our IMC as an extension to an existing CPG controller ([8], [9]) used for learning directed locomotion (see Figure 1).

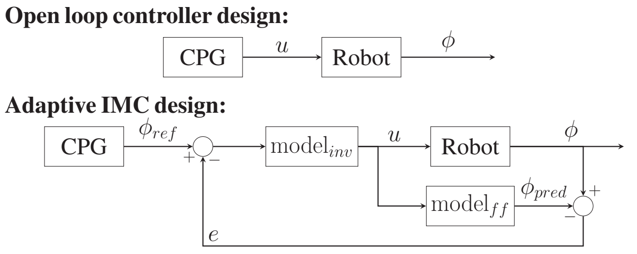


Figure 1: A schematic overview of open-loop CPG controller with and without our adaptive IMC. For the IMC the *inverse model* calculates the required motor input ($\mathrm{model}_{inv} \rightarrow u_i$) to follow a reference ($\phi_{ref,i}$), while the *feedforward model* predicts the robots state based on its input ($\mathrm{model}_{ff} \rightarrow \phi_{pred}$). Differences between the predictied and actual state (ϕ) are sent back (e).

The idea behind the IMC design is grounded both in industry and in neuroscience [10] as a way for (human) controllers to compensate for errors in (movement) control. As an extension to an open-loop controller (in Figure 1, the CPG) we add two models that work together for proper movement execution. The inverse model calculates the required control input u to get the robot from the current state to the reference state, $\mathrm{model}_{inv} : \phi_{ref} \rightarrow u$. The feedforward model predicts the result of that command, $\mathrm{model}_{ff} : u \rightarrow \phi_{pred}$. A perfect model will allow the IMC to subtract any error in the movement. For applications in ER, we will need to learn these models since we do not know the morphologies beforehand. We propose the use of Deep Neural Network (DNN) to learn

978-1-7281-2547-3/20/$31.00 ©2020 IEEE

the models of the IMC scheme. To this end, we will evaluate two different DNN types for our IMC: *IMC vanilla*, in which the two DNN internal models learn on all their layers; and *IMC reservoir*, in which the DNNs only learn on the output layers, much like in reservoir computing [11].

With the IMC controller we can accommodate two important user needs in ER, 1) applicability to a wide range of morphologies, 2) easy implementation as an extension of already existing controller designs. However, it is possible that the adaptability might influence the lifetime learning process. Thus, the overarching research question behind this paper is:

What is the effect of adding the IMC on learning control for directed locomotion in a variety of different morphologies?

II. Related Work

Most of the work in ER in simulation does not address feedback control. ER experiments in which evolution is solely done in real robots do implement feedback control, but not learning, see e.g. the publications by Brodbeck *et al.* [12], Vujovic *et al.* [13], and Nygaard *et al.* [14]. In these papers, feedback control is implemented using P(I)D controllers on simple robots. As PID control is limited in its capability, we believe that it can greatly influence the resulting morphologies. The idea behind our IMC controller is to be minimally invasive to the evolutionary process with the controllers that are currently used in ER.

The benefits of adaptive feedback control have been shown before by Bongard *et al.* [15]. They showed that their four-legged robot was able to learn locomotion rapidly through efficient sampling and continuous self-modelling. Furthermore, the adaptive self-modelling made their machine able to reconfigure its self-representation and control policy after physical damage. Other forms of adaptive feedback have been studied in the form of coupling sensory information back to the controller ([16], [17]). In the case of CPGs, this means modulation of frequency or amplitude of the signals to the motors. For other types of neural networks like HyperNEAT and recurrent neural networks ([18], [19], [20]), feedback allowed switching on and off certain parts of the network based on sensory information. Such an adaptive feedback control changes the overall robot behavior based on the sensory feedback, which is different from our aim to follow a reference trajectory as closely as possible. In other words, our feedback control is much more focused on the correct execution of movements at the local joint level while the other studies change the combined motor behaviours as a whole.

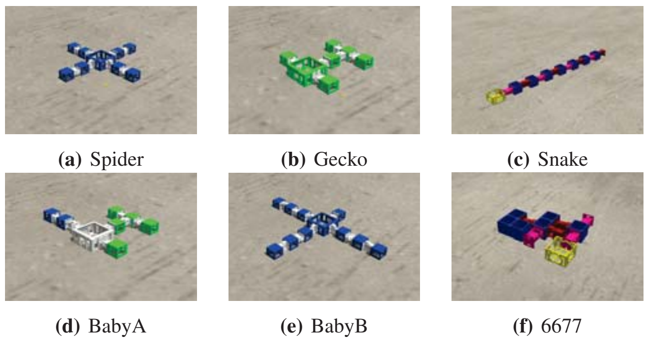
Similarly to our feedback design, Kawato *et al.* [21] implemented an inverse model controller that learned to follow a reference signal with a three-link manipulator using a neural network. In a study by Miyamoto *et al.* [22], two neural networks were used in a similar IMC structure with feedback error learning to control a simulated manipulator as well. Hunt and Sbarbaro [10] implemented IMC with two neural networks that learned their models by example (a pre-existing feedback controller), before testing it on a simulated plant. Li and Deng [23] implemented an IMC design in which a single neural network was used to compensates for model mismatches in both internal models. More recently, [24] compared two adaptive feedback controllers (a PID with and without a convolutional neural network) that used evolution to learn locomotion in a real quadruped robot. These studies show that the IMC structure can be used to obtain high-performance nonlinear control. To our knowledge, we are the first to test an adaptive IMC controller within ER.

Our work positions itself in a rare intersection of adaptive feedback control and ER. Many (adaptive) feedback control studies focus primarily on the dynamics of the errors when following a reference and not on the robot behaviour itself. However, in the context of our paper, the robot's behaviour is also influenced by the changing feedforward CPG. Conversely, much research in ER is solely focused on the behaviour of the controllers, while overlooking the influence of feedback control. We suspect a strong coupling between both within ER, thus playing a vital role in transferring from simulation to the real world. From this perspective, we position ourselves uniquely in comparison to other (bio-inspired) controllers, in that we implement both learning directed locomotion as well as adaptive feedback control. On top of that, we require learning to occur quickly, on a broad range of different robot morphologies.

III. Experimental Setup

A. Test suite

We will use 6 different robots ($N = 6$) that are made of off-the-shelf components based on the modular RoboGen framework, see Figure 2. Spider, Gecko, and Snake (top row) are designed beforehand, while BabyA and BabyB are first generation children from the Gecko and Spider. Finally, the 6677 is a product of the evolution of multiple generations.



(a) Spider **(b)** Gecko **(c)** Snake **(d)** BabyA **(e)** BabyB **(f)** 6677

Figure 2: Test suite with custom (top) and evolved (bottom) designs.

B. Test Procedure

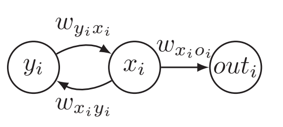
Each robot is tested in 10 different experiments with three different controllers. The original open-loop controller [9], and the two IMCs (vanilla and reservoir). Each experiment consists of 300 learning trials in which the controller improves on the task of directed locomotion. This makes the total of $3 \times 6 \times 10 \times 300 = 5400$ learning trials. At the start of each learning trial, we place the robot in the center of a flat horizontal plane with a gravitational pull of $9.81\,\mathrm{m/s^2}$ downward. Each trial takes 60s in simulation. In total we simulate $5400 \times 60 = 3.24 \times 10^6\,\mathrm{s}$, which equals $900\,\mathrm{h}$. Simulation is done in Gazebo using the ODE physics engine with Runge-Kutta 4 integration ($\mathrm{dt} = 0.05\,\mathrm{s}$). The sampling period of the controller is $0.125\,\mathrm{s}$.

IV. Controller design

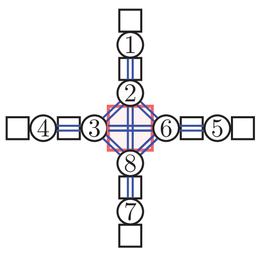
We start with explaining the open-loop CPG controller based on the work by Lan *et al.* [9]. The added IMC controller will use the same CPG controller extended with the two DNN as internal models. The two flavors of IMC (IMC vanilla or IMC reservoir) will be explained in detail later on. Learning locomotion is done by updating the CPG weights in between learning trials while the weights of the DNN models in the IMC are updated during each trial.

A. Learning Locomotion with CPG

The CPGs are based on the design by Ijspeert *et al.* [25]. A single CPG is represented by a neuron pair (x_i,y_i) that reciprocally inhibit and excite each other to produce oscillatory behaviour. Here, i denotes the specific joint that is associated with this CPG. The dynamics of a single CPG is defined by the current state of its neuron pair. The change of every neuron state is calculated by multiplying the current state of the opposite neuron with a weight (w), as shown in Figure 3a. The weights $w_{x_iy_i}$ and $w_{y_ix_i}$ represent a connection strength between each neuron pair within the CPG, while $w_{x_io_i}$ represents the coupling strength from the x-neuron to an output neuron that controls a joint (*i.e.* servo). By changing the weights of the CPGs, we can learn locomotion.



(a) CPG



(b) CPG network

Figure 3: a: A single CPG **b**: The CPG network for our Spider, containing 8 CPGs (numbers) with 10 connections (blue lines).

To enable complex output patterns, we allow CPG connection to exist between neighbouring joints, see Figure 3b. The set of neighbouring joints ($\mathcal{N}_i$) at joint i, consists of all the joints j positioned within a distance of 2 modules. The ordinary differential equations of the resulting CPG network are described in Equation 1. Connections between neighboring joints j are denoted as $w_{x_jx_i}$.

$$\dot{x}_i = w_{y_ix_i}y_i + \sum_{j\in\mathcal{N}_i} x_j w_{x_jx_i} \qquad \dot{y}_i = w_{x_iy_i}x_i \tag{1}$$

For the output neuron a tangent hyperbolic activation function is used (see Equation 2). To simplify the search space we implement the following ralations between the weights: $w_{x_iy_i} = -w_{y_ix_i}; w_{x_jx_i} = w_{x_ix_j}; w_{x_io_i} = 1$. At the start of each learning trial all neuron states will be set to a predefined value $(x, y) = \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$.

$$out_i(x_i) = \frac{2}{1 + e^{-2x_iw_{x_io_i}}} - 1 \tag{2}$$

To improve the weights of the CPG network, we define a performance measure ($\mathcal{F}$). During a learning trial the CPG network commands the robot to move through space, which can be described by a trajectory $p(t)$, see Figure 4a. In the case of directed locomotion, we want this movement to 1) be as far as possible in the target direction (β_T), and 2) have as minimum possible deviations from this direction (δ). To this end, we formulate $\mathcal{F}$ as follows:

1) First obtain δ:

$$\delta(\beta_0, \beta_1) = \begin{cases} 2\pi - |\beta_1 - \beta_0| & (|\beta_1 - \beta_0| > \pi) \\ |\beta_1 - \beta_0| & (|\beta_1 - \beta_0| \leq \pi) \end{cases} \tag{3}$$

Note that δ ranges from $[-\pi, \pi)$, with the range $\left(-\frac{1}{2}\pi, \frac{1}{2}\pi\right)$ being (partly) towards the target direction.

2) Calculate the distance travelled in the target direction at the end of a trial, $\mathcal{D}_{dir}$:

$$\mathcal{D}_{dir} = |P_{end} - P_0| \cos \delta \tag{4}$$

Here $|P_{end} - P_0|$ denotes the Euclidean distance between P_{end} and P_0.

3) Calculate the total distance deviated from the target direction at the end of the trial, $\mathcal{D}_{dev}$:

$$\mathcal{D}_{dev} = |P_{end} - P_0| \sin \delta \tag{5}$$

4) Now we formulate $\mathcal{F}$ as follows:

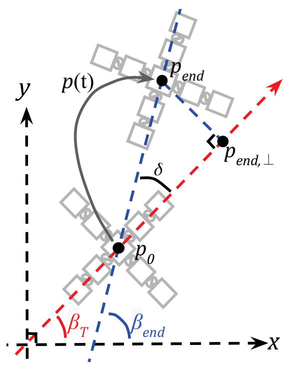
$$\mathcal{F}(\mathcal{D}_{dir}, \mathcal{D}_{dev}) = |\mathcal{D}_{dir}|\mathcal{D}_{dir} - \mathcal{D}_{dev}^2 \tag{6}$$

$\mathcal{D}_{dev}$ is squared to penalize strong deviation from the target direction, $\mathcal{D}_{dir}$ is multiplied by its absolute value to encourage movement towards the target direction. The units of our performance measure are $[\mathcal{F}] = m^2$. A contour plot of $\mathcal{F}(\mathcal{D}_{dir}, \mathcal{D}_{dev})$ is shown in Figure 4b.

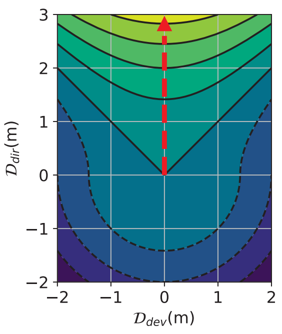
The optimization of the weight will be done using a Bayesian Optimization (BO) algorithm, which models the fitness function using Gaussian Processes ($\mathcal{GP}$) [26]. With $\mathcal{GP}$ BO efficiently selects promising samples for subsequent trials. The hyperparameters of the BO are derived from [9], which shown in Table I. We initialize the BO algorithm by pseudo-randomly evaluating 50 evenly distributed samples in search-space using Latin Hypercube Sampling (LHS).

B. Adaptive IMC Control

for the implementation of optimal feedback control, we extend the open-loop CPG controller with our IMC controller. For the internal models we will use two different DNN. The design of the DNN for the inverse model consists of an input



(a) Performance variables



(b) Contour plot

Figure 4: (a): Visualization of the variables necessary for the performance measure. β_T shows the target direction, while β_{end} the actual direction, P_0 start, P_{end} end positions. The difference in direction is denoted by angle δ and the projection of P_{end} onto the target direction by $P_{end,\perp}$. **(b)**: contour plot of the performance measure. The black isolines show monotonically increasing performance values ranging [-6,8] with steps of 2, from the bottom to top in target direction (red arrow).

Parameters	Value	Description
Initial sampling	LHS	Sampling method
Initial samples	50	Number of samples
Learning iterations	250	Number of evaluations
Kernel type	Matérn 5/2	Approximation kernel
Kernel variance	1.0	
Kernel length	0.2	
UCB alpha	3.0	Acquisition function weight

Table I: Hyperparameters of the BO algorithm, from [9].

layer containing $4k$ neurons (input layer model$_{inv}$ = $[\phi_{1:k}, \dot{\phi}_{1:k}, \phi_{ref,1:k}, \dot{\phi}_{ref,1:k}]^T$), with k being the total number of servomotors; two hidden layers of the same size ($4k$) with Rectified Linear activation units (ReLU); and an output layer of k neurons (output layer model$_{inv} = [u_{1:k}]^T$), which sends the control signals to the servomotors. For the output layer, we choose the same tangent hyperbolic neuron as in Equation 2.

The DNN for the feedforward model has an input layer of $3k$ neurons (input layer model$_{ff} = [\phi_{1:k}, \dot{\phi}_{1:k}, u_{1:k}]^T$); two hidden layers of the same size ($3k$) with ReLU; and an output layer of $2k$ neurons (output layer model$_{ff}$ = $[\phi_{pred,1:k}, \dot{\phi}_{pred,1:k}]^T$). For the output neurons we also use the tangent hyperbolic function.

The implementation of the DNN is done in C++ using libtorch v1.4.0. For optimization, we update both biases and weights. Only at the beginning of an experiment (*i.e.* at the start of the first trial) we randomly initialize these parameters by sampling from a Gaussian distribution with zero mean and variance of one. To train the models in the IMC vanilla we continuously update all the weights and biasses for the hidden and output layers. For the IMC reservoir, we freeze the weights and biases of the hidden layers after initialization and only optimize the output layer. Updating of the weights and biases is done whenever a new reference signal is sent to the IMC. For the IMC vanilla controller the total number of parameters to be optimized in the model$_{inv}$ is $4(4+4+1)2k = 72k$, and for the model$_{ff}$ $3(3+3+2)2k = 48k$. While for the IMC reservoir the total number of parameters in the model$_{inv}$ is defined as $4(0+1)2k = 8k$, and for the model$_{ff}$ $3(0+2)2k = 12k$.

The goal of the IMC is to minimize the error between the reference state and the actual state. This is achieved by 1) the model$_{inv}$ providing correct motor inputs to follow the desired state, and 2) the model$_{ff}$ correctly predicting the next robot state. To improve our DNN, we will learn these tasks separately for each model by updating the weights and biases of the DNN with the well-known ADAM optimizer not re-initializing the parameters in each subsequent trial [27]. The ADAM optimizer requires a loss function, which will be different for each model based on the error it makes. For model$_{inv}$ the desired output is the reference state, with the error being the difference between the reference and the actual robot state ($E_{inv} = \phi - \phi_{ref}$). For model$_{ff}$, the desired model output is the actual robot state, with the error being the difference between the actual and the predicted robot state ($E_{ff} = e = \phi_{pred} - \phi$). The loss function is defined as the summed squared error ($\mathcal{L} = E\frac{1}{2}E^T$). With $\mathcal{L}$ we calculate the gradients for ADAM. For regularization, we implement L2 weight decay [27]. The hyperparameters for the ADAM optimizer are shown in Table II.

Parameters	Value	Description
α	0.005	Learning rate
β_1	0.9	First moment decay
β_2	0.99	Second moment decay
ε	$1\,\mathrm{e}^{-6}$	Division constant
L2	0.001	Weight decay

Table II: Hyperparameters of the ADAM optimizer.

Below we have summarized the most important differences between our controller conditions (see Table III). For all conditions, we learn directed locomotion in a CPG network using BO to adjust the weights of the CPG network. The number of CPG weights is dependent on the number of servomotors (k) and neighbour pairs ($|\mathcal{N}|$) in a specific morphology. In addition, the IMC controllers also learn feedback control with their two DNN models using the ADAM optimizer. The IMC vanilla and IMC reservoir differ in the number of parameters that are adapted during feedback learning (based on the number of servomotors as well). The full code is publicly available on github here.

To test the validity of the feedback control, we will select the final open-loop controller and the best performing IMC controller (either vanilla or reservoir) and re-test them in a noisy environment for all experiments (in total, 120 re-evaluations). The noise (a Gaussian with mean 0 and STD 0.05) will be applied on the servomotor input at every iteration. We will compare the mean difference (Δ) between the open-loop and best IMC.

	Parameters	Open-loop	IMC vanilla	IMC reservoir
Directed	Controller	CPG	CPG	CPG
Locomotion	Optimizer	BO	BO	BO
	nr. param.	$k + \|\mathcal{N}\|$	$k + \|\mathcal{N}\|$	$k + \|\mathcal{N}\|$
IMC	Models	-	2 DNN	2 DNN
Conrol	Optimizer	-	ADAM	ADAM
	nr. param.	-	$(72 + 48)k$	$(8 + 12)k$

Table III: Summary of the different controller conditions. The open-loop controller is also implemented in the two IMC controllers, which differ in the number of parameters to optimize.

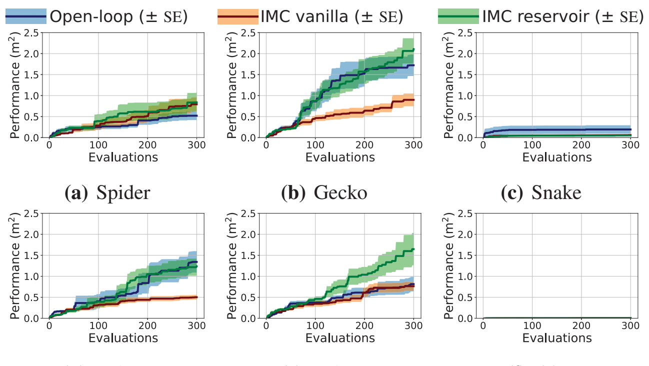
C. Statistical analysis

For the data analysis, we compare the progression of the performance measure as a function of evaluations for all controllers (open-loop, IMC vanilla, IMC reservoir) per robot. Additionally, we compare differences between the performance end-values of each controller within a certain robot morphology, as well as a grouped comparison. Similarly, in the noisy condition, we compare the mean difference in performance (Δ) between the open-loop and the best IMC per robot morphology, and aggregated in a group. Per robot morphology comparison is done using an independent samples *t-test* ($N = 10$), while for the grouped comparison we use a paired samples *t-test* (N=6). Assumptions on normality and equal variance are checked by performing a Shapiro–Wilk test (normality for $p > 0.95$) and an F-test (normal ratio between variances $\frac{1}{2}$ and 2). Additional report on the effects size are done using Cohens-d.

V. Results

In total conducting all 180 experiments took approximately 45 h in real-time, which is an average of 15 min per experiment. In comparison to the 900 h of simulated time this equates to a 20× speedup. In Figure 5 we plotted the progression of the mean performance (±SE confidence interval) over the number of evaluations for all six robots. Four of these plots show an usual improvement over time, but the Snake and 6677 do not seem to learn much. The reason is the orientation of the robots at the start of the learning period. Unfortunately, both the snake and 6677 were placed orthogonal to the required direction to follow. This made the learning task extremely hard as they should have discovered the strategy of turning 90 degrees first and then walking into the right direction (video[1]). Occasionally this occurred for the Snake, but not for 6677.

The learning curves show that the IMC reservoir performed better or just as good as the other controllers for half of the morphologies. Furthermore the open-loop performed similarly to the IMC reservoir except for the Spider and BabyB in which it performed worse. Lastly, the IMC vanilla performed worse or just as good as the open-loop for all morphologies except for the Spider. The rate of learning is similar for all controllers at the start of the learning task (evaluations < 100), except for the Gecko in which the IMC vanilla learns slower than the others. After about 100 evaluations learning diverges for the Spider, BabyA, and BabyB.



(a) Spider (b) Gecko (c) Snake
(d) BabyA (e) BabyB (f) 6677

Figure 5: Mean learning curves (±SE) of the task of directed locomotion for the Spider, Gecko, Snake, BabyA, BabyB, and 6677. The blue lines denote the open-loop controller, red lines the IMC vanilla controller, and the green lines the IMC reservoir.

Checks on the performance end-values met the requirements on normality (Shapiro-Wilk $p > 0.95$), and homogeneity of variance ($\frac{1}{2}$ <F-test< 2) between the different groups that were being compared. In Figure 6 we can see that there was no significant difference found between the end-values of the open-loop controller (blue bars) and the IMC reservoir (green bars) for all morphologies. Spider $p = 0.24$, Gecko $p = 0.36$, Snake $p = 0.20$, BabyA $p = 0.77$, and 6677 $p = 0.90$. BabyB was closest to a statically significant difference $p = 0.08$ with MEAN±STD for the OL: 0.81 ± 0.55 and IMC reservoir: 1.65 ± 1.21, effect size= 0.89.

When comparing the performance of the open-loop control and the IMC vanilla (red bars), no significant difference was found in the end-values for the Spider ($p = 0.88$), Snake ($p = 0.57$), BabyB ($p = 0.63$) and 6677 ($p = 0.32$) morphologies. Significant difference in learning locomotion performance was found between the open-loop control and the IMC vanilla for the Gecko ($p = 0.02$, OL: 1.72 ± 0.81 and IMC vanilla: 0.90 ± 0.48, effect size= 1.15), and BabyA ($p < 0.007$, OL: 1.34 ± 0.81 and IMC vanilla: 0.50 ± 0.18, effect size= 1.43).

In the final comparison, no significant difference was found between the IMC vanilla and the IMC reservoir, for the Spider ($p = 0.82$), Snake ($p = 0.23$), and 6677 ($p = 0.25$). Here, a statistically significant difference in performance end-values was found for the morphology Gecko ($p = 0.003$, IMC vanilla: 0.90 ± 0.48, and IMC reservoir: 2.11 ± 0.93, effect size= 1.63), BabyA ($p = 0.003$, IMC vanilla: 0.50 ± 0.18, and IMC reservoir: 1.24 ± 0.64, effect size= 1.57), and BabyB ($p = 0.05$, IMC vanilla: 0.76 ± 0.32, and IMC reservoir: 1.65 ± 1.21, effect size= 0.99).

The grouped comparison ($N = 6$) between the controllers did not show any statistical significant difference between mean performance end-values over all morphologies (OL $= 0.77 \pm 0.61$; IMC vanilla $= 0.50 \pm 0.35$; IMC reservoir

[1]https://youtu.be/TgC0gHII7mg

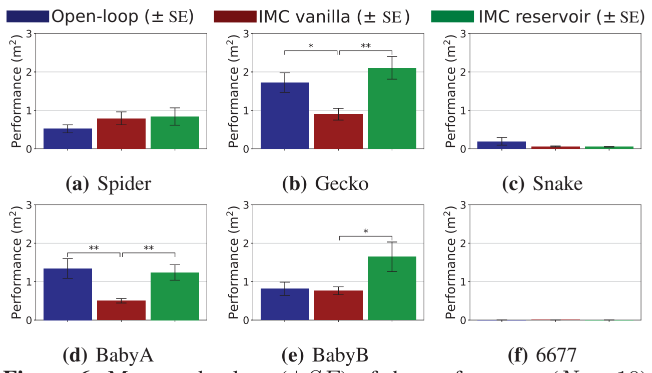


Figure 6: Mean end-values ($\pm SE$) of the performance ($N = 10$) per robot (Spider, Gecko, Snake, BabyA, BabyB, and 6677) for the open-loop controller (blue), IMC vanilla (red), and IMC reservoir (green). Here, * denotes $p < 0.05$, and ** $p < 0.01$.

$= 0.98 \pm 0.78$). OL vs. IMC vanilla controller $= p = 0.19$; OL vs. IMC reservoir $= p = 0.23$; and IMC reservoir vs. IMC vanilla $p = 0.08$ (with $d = 0.79$).

Based on the results from Figure 6, we choose to compare the open-loop against the IMC reservoir controller in the noisy condition experiment. The results of the statistical analysis on the feedback validation experiment with added noise are shown in Table IV. It should be noted that a negative difference means that the performance decreased after adding noise. Here we can see that for most morphologies the average performance decreased after adding noise (except for the Δopen-loop in the 6677, and ΔIMC reservoir in the BabyB morphology). Furthermore, we found that there was a statistically significant difference between the Δopen-loop and the ΔIMC reservoir controller for the Gecko ($p = 0.008$, Δopen-loop: -0.66$\pm$0.42, and ΔIMC reservoir: -0.18$\pm$0.25, $d = 1.40$), Snake ($p = 0.03$, IMC vanilla: -0.19 $\pm$ 0.23, and IMC reservoir: -0.46 $\pm$ 3.62 $\cdot\, e^{-2}$, $d = 1.10$), and BabyB ($p = 2.56 \cdot e^{-6}$, IMC vanilla: -0.51$\pm$0.20, and IMC reservoir $= 4.14 \pm 0.11 \cdot e^{-4}$, $d = 3.10$).

For the grouped comparison we did not find any statistical difference between the mean difference of the open-loop control and the IMC reservoir when adding noise ($p = 0.08$).

Controller	Δ open-loop	Δ IMC reservoir	p-value	d
Spider	-0.24 $\pm$ 0.29	-0.03 $\pm$ 0.12	0.07	0.92
Gecko*	-0.66 $\pm$ 0.42	-0.18 $\pm$ 0.25	<0.01	1.40
Snake*	-0.19 $\pm$ 0.23	-0.46 $\pm$ 3.62 $\cdot\, e^{-2}$	0.03	1.10
BabyA	-0.18 $\pm$ 0.38	-0.27 $\pm$ 0.39	0.62	0.24
BabyB*	-0.51 $\pm$ 0.20	4.14 $\pm$ 0.11 $\cdot\, e^{-4}$	<0.001	3.10
6677	0.46 $\pm$ 3.39 $\cdot\, e^{-3}$	-1.60 $\pm$ 2.87 $\cdot\, e^{-3}$	0.18	0.65
Grouped	-0.30 $\pm$ 0.22	-0.08 $\pm$ 0.10	0.08	1.24

Table IV: Mean difference ($\pm$STD) in performance after adding noise to the best controllers of each experiment ($N = 10$). Negative differences indicate that the performance decreased.

VI. Discussion

The issue we address here is rooted in evolutionary robot systems, where robot bodies and brains evolve simultaneously. In such systems robot 'birth' must be followed by 'infant learning' with a robust learning method that works for all possible morphologies [4], [5]. Here, we consider a specific type of evolvable robots (modular morphologies) with specific controllers (based on CPGs) learning a specific task (directed locomotion). The key idea of the paper is to extend the usual open-loop controller used frequently in ER by an adaptive feedback mechanism that uses a forward model and an inverse model that can be learned on-the-fly.

The straightforward way to test this concept would be to perform evolutionary runs with and without it. However, such runs would take enormous amounts of time. Therefore, we choose to use a test suite of six robots with various morphologies – akin to a single generation in an ER experiment. The actual experiments compare the open-loop controller and two versions the extended controller concept on the robots of our test suite. The results show that using the IMC reservoir approach can have a positive effect on the achieved speed and the robustness of the controller compared to an open-loop controller.

At the start of learning, we see similar learning curves for all controllers (Figure 5), which is to be expected since sampling was done pseudo-randomly for the first 50 samples with LHS. After this initial sampling, divergence starts around 100 evaluations in the Spider, Gecko, BabyA and BabyB, where the less performing controllers begin to plateau. The plateauing indicates to us that the end-values truly represents differences in task-performance between controllers, and were not caused by ending the experiments too soon. In support of our findings, it seems that the differences in end-values could have been more pronounced since performance in the IMC reservoir was still increasing in most morphologies.

Overall, we did not find any significant differences between the grouped mean performance end-values. We suspect that this is mainly due to the low number of robots in our study (6 morphologies). Especially in the case of the IMC reservoir vs. IMC vanilla where there is a clear trend visible ($p = 0.08$, $d = 0.79$). In hindsight, this probably means that we could have increased the number of morphologies while still retaining a medium to large effect size. Additionally, the low performance of the Snake and the 6677 may have reduced the mean end-values considerately. In retrospect rotating the morphologies might have been sufficient to find a statistically significant difference between the controllers. A rerun of these morphologies in the 'correct' direction did result in end-values that were similar to the Gecko and BabyB (not shown).

The use of adaptive IMC has some interesting consequences for learning directed locomotion with the BO algorithm, due to the additional learning of the internal models. As the internal models change over time, we can see that the IMC controllers might behave inconsistently. This means that with retesting the same CPG weights a second time, after a period of DNN adaptations, we would likely see a slightly different behaviour and thus a difference in performance. As a consequence, instead of a crisp representation of the performance, we should consider each data point more as a probability. Luckily, the BO algorithm is well equipped to do this as it evaluates samples with a certain uncertainty. Nevertheless, we did not put any extra effort into fine-tuning the BO algorithm for this, as we wanted to keep the locomotion learning equal between the controllers.

Learning two internal models at the same time might be another cause for inconsistency. changing two interdependent systems can increase the risk of unstable adaptations that are unpredictable [21]. A way to show this is by imagining a special case in which one DNN internal model is perfect. Here it could happen that the other imperfect model causes an error that will change the weights of the perfect DNN. For future work, it might be better to learn each model differently, for example, switch which model learns for a period of time, store different sets of data in a buffer (odd vs. even samples), or decrease the learning rate of one of the models making the other model adapt to it.

Additionally, adaptations of the IMC in combination with the adaptation of the CPG weights for the locomotion task can lead to over-fitting of the DNNs. Namely, as the CPG network keeps improving its weights we will likely see that behaviour converges to a (local) optimum. This may lead to the internal models recognizing which output is desired based on the robots state instead of the reference signal presented by the CPG. Resulting in bad behaviour when the robot revisits the same state with a different reference signal. As a consequence, badly functioning feedback control causes bad performance during exploratory samples, which in return reinforces the convergence of the CPGs leading to early plateauing of the fitness. We addressed over-fitting in the IMC with L2 regularization on the weights. Other possible solutions may be to change the BO accordingly, implementing dropout in the DNN [27] or to reduce the size of the models.

Inconsistency of control can cause the resulting performance to be badly represented by the $\mathcal{GP}$ [26]. The sample efficient nature of the BO algorithm becomes counterproductive if bad samples are regarded higher and/or good samples lower, as it will negatively affect the likelihood of samples being picked in the neighbourhood of those points. The sensitivity to this bad sampling is different for each morphology, as a morphology with a narrow region of high performance in search space would be more sensitive than a morphology with a wider optimal region. Additionally, inconsistency in control leads to high variability in neighbouring points, which can cause a very erratic and/or uncertain $\mathcal{GP}$ approximation of the fitness function. This is highly dependent on the amount of data and the type of kernel being used [26]. Inconsistency in control will also occur in other types of feedback controller besides our IMC. For future work, we can take this into account when learning locomotion in real robots using BO. An interesting option might be to incorporate the total amount of loss by the DNNs per evaluation in the covariance matrix of the BO to directly indicate its uncertainty.

How the adaptive IMC influenced the learning locomotion task is difficult to tell. It is interesting to see that for the IMC vanilla the added feedback during learning seemed detrimental to its performance, while the IMC reservoir seemed to benefit. The differences in the number of parameters to be learned (Table III) result in a reduced amount of learning capabilities and an increased rate of learning for the DNNs in the IMC reservoir [11]. The difference in performance indicates that there might be an optimal amount of learning for the IMC. For example, to prevent over-fitting one would like to have a DNN with a slow learning rate and low learning capacity, but, taking this into the extreme might lead to bad adaptations towards new behaviors and limitations in the complexity of the model. Additionally, slow adaptations increases the robustness of the feedback controller, while fast adaptation can cause it to adapt better to changes in control.

The fact that the IMC reservoir outperforms the open-loop controller also indicates that some inconsistencies in control can be beneficial. It can be argued that the aforementioned argument, *bad samples are regarded higher and/or good samples lower* is conversely beneficial, but we disagree. Even though higher valued good samples might attract more samples in their neighbourhood, bad samples would already be regarded as bad and ignored by the BO, thus not affecting the outcome at all. In the end, this would cause the net influence of inconsistencies to be more harmful than good. We, believe the source of improvement to lie in an increase in the amount of uncertainty of the $\mathcal{GP}$ for the IMC reservoir. This would encourage more exploration, which explains the extended periods of learning.

We tested the validity of our feedback controller in the added noise experiments. The results show that the IMC reservoir controller is able to reduce the effects of the perturbation more than the open-loop in the Gecko, Snake, and BabyB morphology. Differences between Δfitness was most apparent in the BabyB with a huge effect size ($d = 3.1$). Overall, we found that the IMC reservoir tends to perform better ($p = 0.08$, with very large effect size $d = 1.24$). This difference was mainly caused by a deviation from the target direction (decrease in $\mathcal{D}_{dir}$ and increase in $\mathcal{D}_{dev}$, not shown). For the open-loop controller in BabyA the addition of noise does not seem to affect it that much. Visual inspection of these controllers with and without noise (same video) did not reveal anything noticeable. Noteworthy, the reduction in performance of the best movement behaviour in the open-loop Snake (the one that rotated 90 degrees first) was due to a failure in rotation before rolling. This strengthens our belief that learning with open-loop control can lead to exploitation of behaviours that are very sensitive to the reality gap.

Many controllers that were evolved in ER literature seem to have a problem with bridging the reality gap [28]. The results of our experiments revealed that small changes in controllers can lead to very different results in performance, which will eventually affect the whole ER process. In this paper, we addressed this issue in two ways simultaneously with our IMC implementation. First of all, we learned directed locomotion on a more realistic robot that uses feedback control instead of open-loop control. Secondly, the added feedback allows for more robust control in the sense that the effects of perturbations and noise on the robots are being actively reduced. Several other solutions to the reality gap have been proposed which we did not cover [28]. As a consequence, we may have overestimated the effect of the IMC reservoir on its ability to reduce the reality gap. We ignored these techniques as we were only interested to see the capabilities of the IMC controller and its effect on learning directed locomotion.

The necessity of a learning loop in ER is based on the difficulties that arise with simultaneous body and brain evolution [3], [4]. Learning prevents potentially good morphologies with initially bad controllers to be discarded right away, which allows for 'hard to control' morphologies to be more present. Another approach that has been proposed is protecting new morphologies that have recently been mutated during selection [29]. Here, controller adaptation to occur within the evolutionary process rather than an additional learning loop. For simple morphologies this method seems to work, but [29] noted that for larger creatures novelty protection is only beneficial with a minimum mutation threshold (which was obtained by a parameter sweep). The advantage of learning over novelty protection is the speed and scalability to more complex morphologies (as such tuning is not required beforehand). For further research we would like to test lifetime learning within an fully evolving robot system with robots that use our IMC design.

VII. Conclusion

The advances of 3D-printing, rapid prototyping and automated assembly make Evolutionary Robotics with physical robots increasingly feasible. This implies that using realistic feedback control in the robots is becoming more important. To this end, we developed a bio-inspired IMC feedback system based on two internal models that can be learned on-the-fly. Our experiments showed that adding this system to existing controllers helps learn proper movement execution without hampering the learning of a locomotion task. The system also can reduce the learning effort and increase the robustness of the learned controllers.

References

[1] J. C. Bongard, "Evolutionary robotics," *Communications of the ACM*, vol. 56, no. 8, pp. 74–83, 2013.

[2] S. Doncieux, N. Bredeche, J.-B. Mouret, and A. E. Eiben, "Evolutionary robotics: what, why, and where to," *Frontiers in Robotics and AI*, vol. 2, p. 4, 2015.

[3] N. Cheney, J. Bongard, V. Sunspiral, and H. Lipson, "On the difficulty of co-optimizing morphology and control in evolved virtual creatures," in *Artificial Life Conference Proceedings 13*. MIT Press, 2016, pp. 226–233.

[4] A. E. Eiben, N. Bredeche, M. Hoogendoorn, J. Stradner, J. Timmis, A. M. Tyrrell, and A. Winfield, "The triangle of life: Evolving robots in real-time and real-space," in *Artificial Life Conference Proceedings 13*. MIT Press, 2013, pp. 1056–1063.

[5] A. Eiben and E. Hart, "If it evolves it needs to learn," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 2020, pp. 1383–1384.

[6] A. F. Winfield and J. Timmis, "Evolvable robot hardware," in *Evolvable Hardware*. Springer, 2015, pp. 331–348.

[7] R. Babuška, *Fuzzy modeling for control*. Springer Science & Business Media, 2012, vol. 12.

[8] G. Lan, M. Jelisavcic, D. M. Roijers, E. Haasdijk, and A. E. Eiben, "Directed locomotion for modular robots with evolvable morphologies," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2018, pp. 476–487.

[9] G. Lan, M. De Carlo, F. van Diggelen, J. M. Tomczak, D. M. Roijers, and A. E. Eiben, "Learning directed locomotion in modular robots with evolvable morphologies," *arXiv preprint arXiv:2001.07804*, 2020.

[10] K. Hunt and D. Sbarbaro, "Neural networks for nonlinear internal model control," in *IEE Proceedings D (Control Theory and Applications)*, vol. 138, no. 5. IET, 1991, pp. 431–438.

[11] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[12] L. Brodbeck, S. Hauser, and F. Iida, "Morphological evolution of physical robots through model-free phenotype development," *PloS one*, vol. 10, no. 6, p. e0128444, 2015.

[13] V. Vujovic, A. Rosendo, L. Brodbeck, and F. Iida, "Evolutionary developmental robotics: Improving morphology and control of physical robots," *Artificial life*, vol. 23, no. 2, pp. 169–185, 2017.

[14] T. F. Nygaard, C. P. Martin, D. Howard, J. Torresen, and K. Glette, "Environmental adaptation of robot morphology and control through real-world evolution," *arXiv preprint arXiv:2003.13254*, 2020.

[15] J. Bongard, V. Zykov, and H. Lipson, "Resilient machines through continuous self-modeling," *Science*, vol. 314, no. 5802, pp. 1118–1121, 2006.

[16] J.-K. Ryu, N. Y. Chong, B. J. You, and H. I. Christensen, "Locomotion of snake-like robots using adaptive neural oscillators," *Intelligent Service Robotics*, vol. 3, no. 1, p. 1, 2010.

[17] K. Inoue, T. Sumi, and S. Ma, "Cpg-based control of a simulated snake-like robot adaptable to changing ground friction," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2007, pp. 1957–1962.

[18] J.-Q. Huang and F. L. Lewis, "Neural-network predictive control for nonlinear dynamic systems with time-delay," *IEEE Transactions on Neural Networks*, vol. 14, no. 2, pp. 377–389, 2003.

[19] J. Drchal, J. Koutník, and M. Snorek, "Hyperneat controlled robots learn how to drive on roads in simulated environment," in *2009 iEEE congress on evolutionary computation*. IEEE, 2009, pp. 1087–1092.

[20] J. Nordmoen, T. F. Nygaard, K. O. Ellefsen, and K. Glette, "Evolved embodied phase coordination enables robust quadruped robot locomotion," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 133–141.

[21] M. Kawato, K. Furukawa, and R. Suzuki, "A hierarchical neural-network model for control and learning of voluntary movement," *Biological cybernetics*, vol. 57, no. 3, pp. 169–185, 1987.

[22] H. Miyamoto, M. Kawato, T. Setoyama, and R. Suzuki, "Feedback-error-learning neural network for trajectory control of a robotic manipulator," *Neural Networks*, vol. 1, no. 3, pp. 251–265, 1988.

[23] H.-X. Li and H. Deng, "An approximate internal model-based neural control for unknown nonlinear discrete processes," *IEEE transactions on neural networks*, vol. 17, no. 3, pp. 659–670, 2006.

[24] E. Massi, L. Vannucci, U. Albanese, M. C. Capolei, A. Vandesompele, G. Urbain, A. M. Sabatini, J. Dambre, C. Laschi, S. Tolu *et al.*, "Combining evolutionary and adaptive control strategies for quadruped robotic locomotion," *Frontiers in Neurorobotics*, vol. 13, p. 71, 2019.

[25] 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.

[26] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'12, USA, 2012, pp. 2951–2959.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[28] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *European Conference on Artificial Life*. Springer, 1995, pp. 704–720.

[29] N. Cheney, J. Bongard, V. SunSpiral, and H. Lipson, "Scalable co-optimization of morphology and control in embodied machines," *Journal of The Royal Society Interface*, vol. 15, no. 143, p. 20170937, 2018.