Adaptive Control for Evolutionary Robotics

And its effect on learning directed locomotion

Master Thesis
October 12, 2020
Fuda van Diggelen
4737539

Adaptive Control for Evolutionary Robotics

And its effect on learning directed locomotion

Author:
Fuda van Diggelen, *4737539*
Technische Universiteit Delft, The Netherlands
Dept: *Mechanical Engineering*:
`f.vandiggelen@student.tudelft.nl`

Supervisors:

prof. dr. Robert Babuska
Technische Universiteit Delft, The Netherlands
Dept: *Mechanical Engineering*:
`r.babuska@tudelft.nl`

prof. dr. Guszti Eiben
Vrije Universiteit Amsterdam, The Netherlands
Dept: *Computer Sciences*:
`a.e.eiben@vu.nl`

Abstract

It has been argued that for the real-world application of an Evolutionary Robotics (ER) system, in which both body (morphology) and brain (controller) are being evolved, a learning stage must occur directly following the 'birth' of a robot. This additional learning stage is deemed necessary because the inherited brain of a newborn robot may not match its inherited body after recombination of the parents' morphologies. In a system that both evolves and learns, the design and optimization of robots take place through two processes: 1) the evolutionary process, where both robot body and brain evolve to obtain a higher fitness, 2) a lifetime learning loop, where the brain is optimized to control a fixed body (produced by the evolutionary process) to obtain a high performance in a certain task. Here, we will only focus on the development of controllers in the lifetime learning loop for the task of directed locomotion, meaning we do not conduct evolution.

The ER perspective with lifetime learning challenges us to develop a controller that can learn rapidly on a broad range of morphologies while being eligible for evolution. This is because lifetime learning is seen as a sub-task that precedes the accomplishment of the user-defined goal in ER. Currently, most simulated robots that are being optimized in ER use an open-loop controller design. For real-world applications, this is clearly limiting, since in practice feedback control is often the norm. The implementation of feedback control in ER can be tricky as the robot morphologies and environments are unknown beforehand. Furthermore, static feedback control limits the range of morphologies on which the controller is applicable, and will likely influence the performance during lifetime learning. To enable learning on a broad range of morphologies we propose the use of adaptive feedback.

In this study, we present an adaptive Internal Model Control (IMC) design for feedback control, that can be used as an extension to an open-loop controller. The core idea is to add a forward model and an inverse model for feedback control, which are both being learned on-the-fly. We will apply two different versions of feedback learning in our IMC design, 1) using a simple deep neural network as our internal models which learns in all layers (IMC vanilla), and 2) using reservoir computing networks that only learn at the output layer (IMC reservoir).

In the end, our controllers contains two learning systems. 1) a *lifetime learning* loop that improves on the task of directed locomotion, and 2) *Adaptive control*, which learns optimal control. To our knowledge, we are the first that conduct a study on lifetime learning with adaptive feedback control in a broad range of morphologies. It is therefore uncertain if/how lifetime learning and adaptive control influence each other. To see if such a controller is eligible for evolution within an ER framework, we will test if it can learn directed locomotion rapidly on a broad range of morphologies.

Directed locomotion is learned by changing the weights of Central Pattern Generators within the controller, using Bayesian Optimization. Feedback learning is done by updating the weights and biases of the internal models in the IMC, using Adaptive Moment estimation. A comparison on directed locomotion performance will be made between an open-loop controller and the same open-loop controller with the addition of either IMC vanilla or IMC reservoir, on a set of modular robots with different morphologies. Additionally, we also investigate the effect of adding noise at the actuator level, as a test for transferability to real robots.

The results show that for the task of directed locomotion the IMC reservoir performs just as good or better in every morphology compared to the open-loop controller, while the IMC vanilla performed worse. Additionally, transferring the IMC reservoir and the open-loop controllers to a noisy environment yielded less detrimental effects on the former. We suspect the improvement in transferability to occurs in two ways: 1) Learning directed locomotion with feedback control directs learning towards behaviours that are less sensitive to perturbations, compared to open-loop; 2) the IMC was able to learn proper feedback control which made the controller more robust.

With our study, we showed that the addition of adaptive feedback control can both affect the rate of learning directed locomotion, the performance of the final controller, and the transferability to a noisy environment. Our controller is special in the sense that it could learn proper feedback control on top of the task of learning directed locomotion in a feasible amount of time for a broad range of morphologies. For future work, we will test our controller in real robots.

Contents

Nomenclature **3**

1 Introduction **4**

1.1 Evolutionary Robotics, and Learning Locomotion 4

1.2 Simulations and the reality gap 5

1.3 Adaptive Internal Model Control 6

1.4 Research Question 8

2 Related Work **8**

3 Method **10**

3.1 Overall system architecture 10

3.1.1 Test suite 10

3.1.2 Simulator 10

3.1.3 Test procedure 10

3.1.4 Learning systems 11

3.2 Learning Directed locomotion 12

3.2.1 CPG network 12

3.2.2 Fitness function 13

3.2.3 Bayesian Optimization 14

3.3 Adaptive Internal Model Control 15

3.3.1 DNNs as internal models 16

3.3.2 Learning optimal feedback control 16

3.4 Statistical analysis 18

3.4.1 Testing the effects of IMC control 18

3.4.2 Validating the feedback controller 18

4 Results **19**

5 Discussion **22**

6 Conclusion **25**

A Triangle of Life Background **31**

B The Bio-inspired IMC **32**

C Revolve, Robots, and Other Technicalities **33**

D Building a CPG Network **34**

E Learning Locomotion **35**

F Learning the Internal Models **36**

G Overview Experimental Setup **37**

Nomenclature

General

$1 \ldots i \ldots k$	Servomotor i up to total number of k servomotors
$\dot{\phi}_i$	Angular velocity of servomotor i
ϕ_i	Angle of servomotor i
ϕ_{ref}	Reference angle
EA	Evolutionary Algorithm
ER	Evolutionary Robotics
ToL	Triangle of Life

Learning Locomotion

CPG	Central Pattern Generators
$1 \ldots j$	Neighbouring joints
$\mathcal{N}_i^j$	Set of neighbouring CPGs j at servomotor i
w	CPG weight (direction is often denoted as $x_i y_i = x_i \rightarrow y_i$)
$\mathcal{D}_{dev}$	Distance travelled perpendicular to the target direction
$\mathcal{D}_{dir}$	Distance travelled in target direction
$\vec{\boldsymbol{w}}$	Vector of CPG weights of a single run/sample
$\boldsymbol{Y}$	Vector containing the fitness values of the corresponding samples of $\boldsymbol{X}$: $[\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_n]$
$\mathcal{F}$	Fitness function
$\boldsymbol{X}$	Matrix consisting of all sample vectors: $[\vec{\boldsymbol{w}}_1, \vec{\boldsymbol{w}}_2, \ldots, \vec{\boldsymbol{w}}_n]$
BO	Bayesian Optimization, method used to learn directed locomotion
$\mathcal{GP}$	Gaussian Process
$f(\vec{\boldsymbol{w}})$	Function approximator
$a\left(f(\vec{\boldsymbol{w}}) \vert \boldsymbol{X}, \boldsymbol{Y}\right)$	Acquisition function
LHS	Latin Hypercube Sampling, pseudo random initial sampling method
UCB	Upper Confidence Bound, sampling approach used for $a\left(f(\vec{\boldsymbol{w}}) \vert \boldsymbol{X}, \boldsymbol{Y}\right)$

Adaptive Control

IMC	Internal Model Control
model$_{inv}$	Inverse model
model$_{ff}$	Feedforward model
DNN	Deep Neual Networks
ReLU	Rectified Linear Units
$\mathcal{L}$	Loss function
ADAM	Adaptive Moment estimation, optimizer used for updating the internal models

1 Introduction

1.1 Evolutionary Robotics, and Learning Locomotion

Developing a robot can be a difficult task that requires a lot of time and iterations in the design process. An often employed strategy is to reduce the complexity of this task by creating a static environment for the robot. Unfortunately, this cannot be done for robots that are being developed to operate outside of such controlled settings. Especially, when the environment is (partly) unknown beforehand like the deep sea, volcanoes, or other planets [37], the traditional designing method can fall short to account for every obstacle possible. If we look at how nature resolves this issue, we can see a solution to this problem through evolution. The evolutionary process has shown to be able to find designs that can coop with a wide variety of the most extreme environments. The field of Evolutionary Robotics (ER) attempts to recreate the circumstances by which such optimal designs could emerge for robotic applications. An advantage of this evolutionary approach is that the overall designing process can be autonomous, cheaper, and generally applicable in a wide range of different situations. Additionally, it has been shown before that an evolutionary process can present novel 'out of the box' design ideas that outperform human design [8].

The field of ER "aims to apply evolutionary computation techniques to evolve the overall design (morphology) or controllers, or both, for real and simulated autonomous robots" [80]. In more detail, an ER system consists of a population of robots that are stationed in an uncontrolled (unknown) environment and subjected to survival. Each robot tries to accomplish a predefined task *e.g* gathering as many resources as possible in the environment. An algorithm selects and mates specific parent robots that performed well on this predefined task, which is expressed in their fitness (for example, in the gathering resources task this can be the total amount acquired till the robot breaks). New offspring designs are created by recombination of the parent robot designs (*i.e.* cross-over and mutation of their genetic code), much like in natural evolution [18]. A major advantage of this approach is that designing can take place on the spot autonomously without the need for prior knowledge about the environment. Evaluation is only done on the fitness for the user-defined task, which circumvents the need for in-depth knowledge about the environment itself. Optimal robot designs emerge on their own and the robot population is adaptive to unforeseen challenges or changes in the environment. The only decision a human has to make is formulating the predefined task via an objective function that is suitable for the evolutionary algorithm (EA).

Early ER research dates back to 1990 [5; 72] and was then primarily seen as a way to study models of cognition with minimal prior assumptions [24; 62]. ER provided a core methodology for research into *artificial life*, using simple integrated sensorimotor control systems for robots. This has led the main focus of ER to be on the development of EA for designing mobile robots with the predefined task of moving through space [59]. Moving through space has been formulated in different types of objectives: 1) *gait learning* moving as far away as possible from a starting position; 2) *targeted locomotion* moving from starting position A to an end position B; 3) *directed locomotion* move as far as possible in a certain direction. In this thesis, we are mostly concerned with the latter, which can be formulated by an objective function that increases with the amount of traveled distance in a certain direction and decreases with the amount of deviation from this direction.

Most ER experiments are focused on improving the design of a controller in robots with a fixed morphology (this was estimated to be the case for over 95% of the ER literature in 2015 [61; 83]). This is because

In a system that both evolves and learns, the design and optimization of robots take place through two processes: 1) the evolutionary process, where both robot body and brain evolve to obtain a higher fitness, 2) a learning loop, where only the brain is optimized to control a fixed body (produced by the evolutionary process) for high performance. This idea is conceptualized in a framework called the Triangle of Life (ToL [16], see Figure 1), which distinguishes three important events and stages in the life of a robot: 1) when a design is formulated and subsequently built during morphogenesis, 2) after the robot is delivered it will enter an infancy stage where the additional learning occurs (represented by the blue loop in Figure 1), and lastly 3) a mature stage where the robot attempts to complete the predefined tasks and can become eligible for mating by the EA. The selection of the parents takes place during this last stage, which will eventually lead to the conceptualization of a new robot design to start a new life. The learning stage in the ToL serves as a way to improve basic motor behaviors like locomotion before attempting to accomplish the more complex tasks, much like a child learning to crawl and walk [35]. The idea is that this additional learning can reduce the chance that a potentially good morphology is discarded when it is paired with an initially bad controller. Additionally, such an extra optimization step can be used to test for early viability to the predefined tasks (*e.g.* did a robot learn locomotion sufficiently before we try to gather resources). For a more detailed introduction to the ToL, we refer to Appendix A. The additional learning loop in the ToL is also referred to as *lifetime learning* [36], and will be the main focus of our research for the task of directed locomotion.

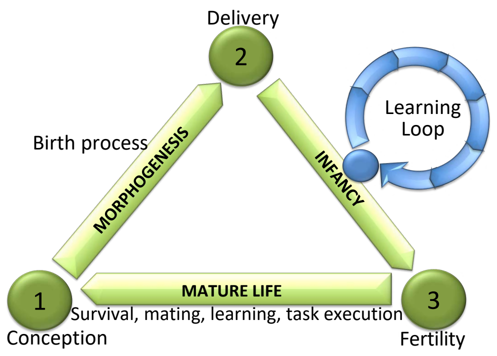


Figure 1: Triangle of Life, adapted from [16]. Here the lifecycle of a robot starts at conceptualizing its design. Subsequently, the robot is born either by hand or in an automated process. To properly assess the robot performance, we first optimize its controller in a learning loop (similar to an infant learning to walk). At last we can consider the robot to be eligible for selection by evolution in its mature life

It might seem to the attentive reader that the ToL re-framed the optimization done with evolution, to a machine learning optimization problem that learns directed locomotion in a single robot with fixed morphology. Although there are clearly some similarities between machine learning in general and lifetime learning, we would like to emphasize that the ER perspective challenges us to design a controller that can learn rapidly on a broad range of morphologies while being eligible for evolution. Often machine learning is narrowly focused on a certain task for a specific machine, while lifetime learning is intended for learning multiple sub-tasks in a robot for which the morphology and environment are not known beforehand. Additionally, the intended real-world application in evolving robots reduces the number of computational resources available for learning as each robot learns on its own and presents a unique test case which prevents the use of parallelization. Furthermore, the main goal for the robot is not to learn but to accomplish user-defined tasks after the initial learning phase. In the end, this motivates the development of controllers that can be optimized efficiently on a broad range of morphologies.

1.2 Simulations and the reality gap

It has been estimated that the majority of the experiments conducted in ER are solely done in simulation [67]. Most of the studies that did test their designs on real robots only constructed the final product of their EA, as a means of validations [48]. This bias towards simulations is because real-world ER experiments are often very time-consuming, which ultimately limits the number of evaluations and robots tested. On the other hand, simulation run much faster, but often fail to capture the rich complexity of the outside world. This has lead to a bad transferability of the controllers developed from a simulated robot to their real counterpart. Meaning that the most optimal controllers that were developed inside the simulation often perform significantly worse in the real world. This phenomenon is also called the *reality gap* [33], which refers to differences that emerge from the simplifications in simulators.

The reality gap calls for ER research to step out of the simulator, but the resources and time that are needed for multiple generations of different robots that evolve seems problematic. It has therefore been argued that a real-world application of a full-fledged ER system (evolving both body and brain) must contain an additional learning loop as proposed by the ToL [17] (for efficient use of both resources and time). Although the development of controllers within the ToL framework has shown its feasibility in simulation [14; 34; 46; 36], we still lack the time efficiency to conduct a full-fledged experiment in the real world. A recent study by Gongjin *et. al* [45] developed a lifetime learning algorithm that was able to reliably learn directed locomotion in simulation within 5 h, which makes real-world experiments seem possible soon. Unfortunately, real-world testing of the final controllers did show a reality gap as well. We believe that this was caused by the use of open-loop control in simulation during the learning process.

ER studies that use feedback often implemented sensory information as a means for adjusting the overall motor behavior of a robot controller [26; 13; 63]. For example, touch sensors can turn certain parts of a neural network controller on/off when sensing the ground. Changing the overall motor behavior is different from closed-loop feedback control in which the robot tries to follow a reference trajectory. Here, the state of the robot is monitored to compensate for disturbances acting on the robot. When the actual movement deviates from the intended movement, the robot attempts to reduce the movement error accordingly. To our surprise, we found that much of the research in ER has been focused on the development of control strategies without such a closed-loop design. For real-world applications, this is clearly limiting, since in practice closed-loop feedback control is often the norm. Disturbances acting on the robot, measurement noise, and model-robot mismatch can all be a source for movement errors. These deviations from the intended movement are intrinsic to the real world (not necessarily due to a reality gap) and require some form of feedback to compensate for correct movement execution. If we transfer an evolved open-loop controller from simulation to a real-world robot we can, therefore, already see a reality gap emerging by the fact that the controllers are different. Hence, the controllers obtained by the EA in the simulator will most likely show a difference in behavior in the real world. Adding to that, the absence of feedback control can also impair the transferability of (already developed) optimization algorithms in ER. Meaning, the learning performance of developed EA may also depend on the types of controllers they optimized (open-loop vs. feedback). It is therefore uncertain if we can transfer these optimizers directly to a real-world ER system without losing in learning performance. If we want to step out of the simulator we should investigate the influence of feedback control on our algorithms.

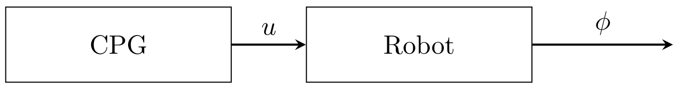
There have been multiple solutions proposed for reducing the reality gap, *e.g.* adding noise, randomly changing parameters, and switching between simulator environments [42; 78]. Besides being more realistic, closed-loop controllers themselves can also reduce the reality gap by stabilizing for proper movement execution in the case of perturbations. We should consider the effects of feedback control on our learning algorithms, especially now that we are approaching the feasibility of real-life experiments. In this thesis, we will investigate the influence of adding feedback control on the task of learning directed locomotion using the lifetime learning algorithm from [45]. In general, this subject is of interest as there seems to be a significant gap in the literature about ER learning with feedback control. It would be interesting to see if the addition of feedback hurts the learning algorithm that was used by [45]. In the next section, we will introduce the type of feedback control that we will be using.

1.3 Adaptive Internal Model Control

Correct implementation of feedback control in ER can be tricky as the robot morphologies and environments are unknown beforehand. As for now, when an open-loop ER controller is transferred from simulation to a real robot, then the latter often employs a PID feedback control system. The feedback signal from a PID-controller is based on the error signal between the intended movement and the actual movement of the robot. PID control takes the dynamics of this error signal (its current, integral, and derivative value) and alters the motor input signals via gains (respectively: Proportional, Integral, and Derivative gains) [2]. PID control is the most used type of feedback control in the industry since it is easy to implement and works sufficiently well for most applications. A drawback to PID control is its limitation in the amount of control complexity it can accommodate. For complex systems, there is often significant coupling among the effects of the three PID gains and thus correct tuning can become very difficult [3]. Subsequently, bad design can lead to oscillations and instability of a PID controller. The use of PID, therefore, might influence the complexity of the controllable morphologies it can accommodate.

We propose the use of a bio-inspired feedback controller schema named internal model control (IMC) [40; 27]. The IMC scheme was both proposed in industry [21; 15], and in neuroscience [40; 53] as a way for (human) controllers to compensate for errors in (movement) control. We present more in-depth information on the bio-inspired background of IMC in Appendix B. A major advantage of our controller design is its easy implementation as an extension to an already existing open-loop controller. Figure 2, shows the addition of the IMC control scheme to the open-loop controller used in [45]. The original open-loop controller used to learn directed locomotion by changing the weight of a network of Central Pattern Generators (CPG) that sends an input signal u to the servomotor(s) in the robot (more on this in the method, Section 3.2). In response, the robot tries to move the corresponding servomotor to angle ϕ. The IMC design is an extension to the (CPG-based) open-loop controller and ads two internal models for proper movement execution with additional feedback Figure 2.

Open-loop controller design:



Closed loop IMC design:

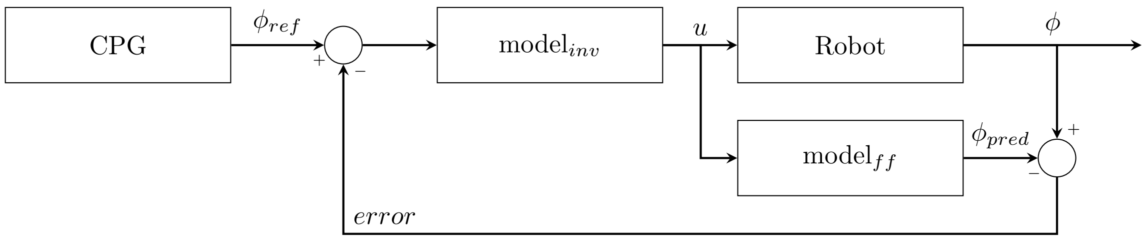


Figure 2: Schematics of the open-loop controller (top) and the closed-loop IMC controller (bottom). Signals denoted between processes can be either single-valued (in the case of 1 actuator) or vectors (for multiple actuators). In the open-loop case, a Central Pattern Generator based controller (denoted as CPG) sends an input u to the servomotor(s) in the robot, which directly dictates the movement of the robot by actuating the corresponding servomotor(s) to angle ϕ. Here, the CPG is uninformed about the current state or actual execution of the movement. In the case of the IMC, the CPG controller produces a pattern that defines a preferred state *i.e.* reference state ϕ_{ref} for the servomotor(s). The inverse model (model$_{inv}$) calculates the required servomotor input u to get to this state. Based on u the feedforward model (model$_{ff}$) predicts the state of the robot ϕ_{pred} in parallel to the robot moving. The difference between the state prediction and the actual state ϕ is fed back as an error signal.

The two internal models in the IMC are called the *inverse* and *feedforward* model in the open-loop control scheme. The inverse model calculates the required control input u to get the robot from the current state to the reference state, model$_{inv}$: $\phi_{ref} \rightarrow u$. The feedforward model predicts the expected sensor output of that movement based on the control input, model$_{ff}$: $u \rightarrow \phi_{pred}$. Differences between the predicted sensor output and the actual sensor feedback are relayed back to the inverse model thereby closing the loop. The idea behind the added feedforward model working in parallel with the robot is to subtract the expected effect of the motor input on the servomotor from the actual angle. When the difference between the predicted movement and actual movement is zero then the *error* becomes zero. With a perfect model of the robot, the IMC scheme allows for filtering and canceling out disturbances via the error signal. Additionally, some studies apply a filter on this error signal to limit the effects of measurement noise and to provide stabilization for higher frequencies [3]. We did not deem this necessary as we bounded the possible output behavior of our controller and expected the adaptive control scheme to learn to accommodate for destabilization.

For robots that live in uncontrolled dynamic environments, internal models with fixed models will likely fall short. For the use in ER, a static feedback controller may limit the range of morphologies on which the controller is applicable, and will likely influence the performance during lifetime learning. Adaptive control is an approach where the parameters of the controller are (continuously) changed online to improve and/or maintain the required performance, even when (unforeseen) changes in the environment or robot take place. In IMC we can apply adaptive control by changing the internal models to cope with the mismatch between the robot and the model. The amount of change can be based on an error signal that represents how well the inverse model controls the robot, and how well the feedforward model predicts the outcome of that control. How the models are adapted depends on the type of model that is being used as an internal representation. The names inverse and feedforward refer to the type of dynamical model necessary when using a mechanical approach [60]. Unfortunately, in ER the robot's morphology and environment are not known beforehand, which denies the use of such a mechanical representation. To implement adaptive control in our robot, we will use two Deep Neural Networks (DNN) to learn the functions of both internal models. DNNs has shown great potential for IMC because of their broad applicability and strong function approximation capabilities [39; 32; 1]. The DNNs will allow optimal feedback control within the IMC structure, by learning the mapping of the internal models model$_{inv}$: $\phi_{ref} \rightarrow u$, and the model$_{ff}$: $u \rightarrow \phi_{pred}$. We refer to learning the internal models as *feedback learning*, which runs in parallel to the *lifetime learning* task of directed locomotion.

We will test two different implementation of DNN for our internal models. As a start we will begin with a naive approach, using a simple of the shelf DNN which we will call IMC vanilla. Secondly, we add a bit more sophistication in our internal models by implementing a similar DNN on which we will only learn on the output layer (based on reservoir computing models in neuroscience [51]), which we will refer to as IMC reservoir. More information on the distinction between these two will follow in the method.

To summarize, lifetime learning in ER challenges us to develop a controller that can learn rapidly on a broad range of morphologies while being eligible for evolution. Current controllers suffer from a reality gap when transferred to real robots with feedback control. We propose an adaptive IMC design to increase the transferability of existing open-loop controllers. Here, we test how such a feedback controller influences lifetime learning performance. Taken together, we implement lifetime learning for the task of directed locomotion on an open-loop controller with the addition of adaptive IMC using two different implementations (vanilla and reservoir).

1.4 Research Question

In this paper, we address the transferability of ER controllers from simulation to real robots by learning directed locomotion with feedback control. For our test case we will use the open-loop controller that was used by [45], and two different implementations of our IMC controller on a set of robots. Due to the novelty of learning locomotion with adaptive feedback control in ER, we are interested in answering the following question:

> *What effect does adding the IMC have on learning directed locomotion on a broad range of robot morphologies?*

We will answer this question in terms of

1. Speed of learning, the rate at which the controllers improve on the task.
2. Task performance, the end-values of the fitness for each morphology.
3. Movement strategies, the eventual best-performing robot behaviors.

Subsequently, to validate if the IMC does increase the transferability of the learned controller we will answer the following sub-question:

> *Is the IMC able to retain a higher level of fitness in a noisy environment compared to open-loop control?*

2 Related Work

As mentioned before, many ER experiments stay inside the simulator where their open-loop controller is never transferred to a real robot. In the experiments where the controllers are tested on their physical twins, real-world performance is often worse than what would be expected from simulation [65; 69; 44]. Our work investigates the role of feedback control on learning directed locomotion which is overlooked by most of the work in ER. The experiments that do address feedback control are the ones in which evolution is solely done in real robots, *e.g.* by Brodbeck *et al.* [5], Vujovic *et al.* [81], and Nygaard *et al.* [64]. In these papers, feedback control is implemented using a P(I)D controller on simple robot morphologies. As PID 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. We attempt to achieve this by adapting the internal models for learning optimal feedback control, which makes them suitable for a wide range of morphologies. An additional advantage of our setup is the fact that real ER experiments are often limited in population size, the number of parameters that are being optimized for in the search space, and/or the number of generations, due to constraints in time and/or resources. Nevertheless, they do show that EAs can be used to learn locomotion in real robots.

The benefits of adaptive feedback control have been shown before in a famous paper by Bongard *et al.* [4]. They showed that their four-legged robot was able to learn locomotion rapidly through efficient sampling and continuous self-modeling. Here, self-modeling was done by white-box system identification with predefined parameterized shapes. Based on the motor input data and feedback from the joint angles and IMU sensors multiple explanatory self-models could be produced by the white-box algorithm. Efficient sampling was done by collecting new data that could discriminate the most between these hypothesized models. The adaptive self-modeling made their machines resilient meaning it was able to reconfigure its self-representation and control policy after physical damage. Other forms of adaptive control have been studied by implementing different types of neural networks [68; 31]. In the case of CPGs, sensory feedback has been used to modulate the frequency and/or amplitude of the output signals to the motors. For other types of neural networks like HyperNEAT and recurrent neural networks [26; 13; 63], feedback allowed switching on/of certain parts of the network to change overall behavior. As mentioned before, such adaptive feedback control aims to change the overall robot behavior based on the sensory feedback, which is different from our controller that aims to follow a reference trajectory. In other words, our feedback control is much more focused on the correct execution of movements at the local joint level while these studies change the combined motor behaviors as a whole.

The idea of bio-inspired control is not new and has already been studied in the field of neurorobotics [19]. The mathematical formulation of the oscillatory behavior in groups of neurons in the spinal cord of mammals [20; 11; 38] has aspired the development of CPG controllers that successfully learned locomotion in different types of robots like salamanders [30], fish [43], and insects [49]. In these studies, open-loop control was used to learn locomotion. In previous studies, our IMC scheme has been used for (simulated) robotic applications as a means to reverse-engineer certain brain structures like the cerebellum [84]. Kawato *et al.* [40] implemented an inverse model controller that learned to follow a reference signal with a three linked manipulator using a neural network. In a study by Miyamoto *et al.* [58], two neural networks were used in a similar IMC structure with feedback error learning to control a simulated manipulator as well. Hunt and Sbarbaro [27] implemented IMC with two neural networks that learned their models first by example (a pre-existing feedback controller) before implementing it on a simulated plant. Li and Deng [47] implemented an IMC design in which a single neural network was used to compensates for model mismatches in both internal models. Much more recently [52] compared two adaptive feedback controllers that used evolution to learn locomotion in a quadruped robot. One PID based and the other PID + a convolutional neural network. These studies show that the IMC structure can be used to obtain high performing non-linear control. Most of the aforementioned studies focused on a single type of (robotic) application. We are more interested in testing the general applicability of our IMC design to learn optimal feedback control for a wide variety of morphologies, which would be beneficial in an ER context. To our knowledge, we are the first to test an adaptive IMC controller in an ER context.

Our work positions itself in a rare intersection of adaptive feedback control and ER. Many (adaptive) feedback control studies focus primarily on the performance of the feedback signals i.e. the dynamics of the errors, not on the actual movements itself. However, here behavior is also (mainly) given by the feedforward CPG. On the other hand, much ER research is solely focused on the learned performance of the robot behavior, while forgetting the difficulties arising from implementing feedback control. We think that the interplay between both can play a vital role in transferring ER from simulation to the real world. Lastly, in comparison to other (bio-inspired) controllers, we position ourselves uniquely from an ER perspective which requires the IMC to be broadly applicable on multiple morphologically different robots.

3 Method

Our code with the presented IMC implementation is publicly available here.

3.1 Overall system architecture

3.1.1 Test suite

To test the general applicability of our algorithm we use a test suite of 6 morphologically different robots ($N_r = 6$). The robots are made of off-the-shelf components that are based on the modular RoboGen[1] framework, see Figure 3 below. Three robots are designed beforehand which are Spider, Gecko, and Snake (top row). The other three are a product of an evolutionary process of which BabyA and BabyB are first generation children from the Gecko and Spider, while the 6677 is a product of the evolution of multiple generations starting from random shapes.

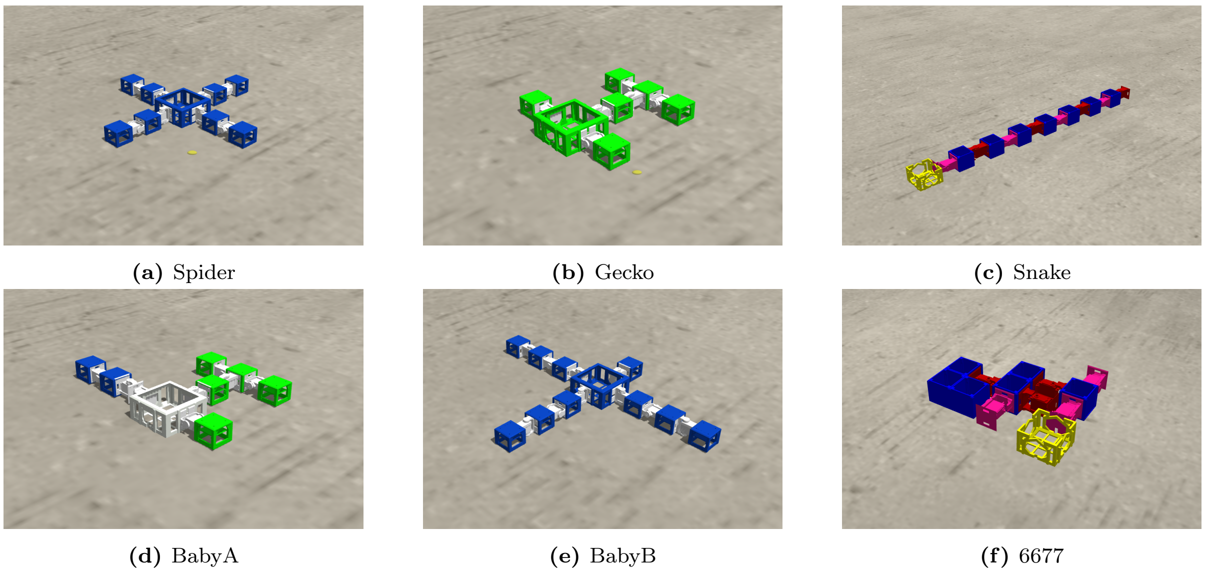


Figure 3: The six robot subjects Spider, Gecko, Snake, BabyA, BabyB, and 6677. Hand-designed robots are shown in the top row while the robots in the bottom row are a product of an evolutionary process.

3.1.2 Simulator

We use the software kit Revolve[2] that is designed as a wrapper for ER experiments around Gazebo[3] which is an open-source 3D robotic simulator [28]. Revolve facilitates ER experiments by calculating fitnesses, selecting parents, recombining/designing new robots, and loading and starting new experiments/trials. The simulations of the robot behavior are done in Gazebo, from which the results are subsequently used to assess the fitnesses for learning locomotion. With Revolve, we are able to design and place robots efficiently in Gazebo while designing/improving our own controllers during a simulation and in-between trials. In Gazebo we use the ODE physics engine for Runge–Kutta 4 integration with a fixed step size of 0.05 s. The sampling period of the whole controller is set to 0.125 s in which the reference signal, sensory input, internal models, and controller outputs are updated. The output of the controller is converted to a continuous signal using a zero-order hold. A more technical in-depth description of the robots and simulator can be found in Appendix C.

3.1.3 Test procedure

At the start of each learning trial, we place a single robot in the center of a flat horizontal plane environment with a gravitational pull of 9.81 $\mathrm{m/s^2}$ downward. Each joint i of the robot receives a signal

[1]`http://robogen.org`
[2]`https://github.com/ci-group/revolve`
[3]`http://gazebosim.org`

generated by the controller. We test each robot in 10 different experiments ($N_e = 10$) with three different controllers (OL, IMC vanilla, IMC reservoir). Each experiment consists of 300 learning trials in which the weights of the CPG controller are updated to learn directed locomotion. In total, we conduct $3 \times 6 \times 10 = 180$ robot experiments that each contain 300 learning trials. In the end, this will add up to a total of $180 \times 300 = 54000$ learning trials. Each learning trial takes 60s in simulation, which totals to $54000 \times 60 = 3.24 \times 10^6$ s of simulated time (equal to 900 h).

3.1.4 Learning systems

It is important to note that we end up with two different learning systems: 1) *lifetime learning* for directed locomotion through updating CPG weights with Bayesian Optimisation (BO, [73]), and 2) *feedback learning* for IMC that changes internal models by adapting the DNN weights and biases with Adaptive Moment estimation (ADAM, [41]). Updating the CPG weights for directed locomotion is done in between trials while adapting the DNN models is done during each trial every 0.125 s (see Figure 4). First, we explain our implementation of lifetime learning with the open-loop CPG controller in Section 3.2. Followed by the implementation of feedback learning with the closed-loop IMC design in Section 3.3. Here, the two flavors of IMC (IMC vanilla or IMC reservoir) will also be explained in detail. The complete implementation of the IMC in this paper is the combination of the same open-loop CPG controller with two DNN for internal models as an extension (as shown in Figure 2).

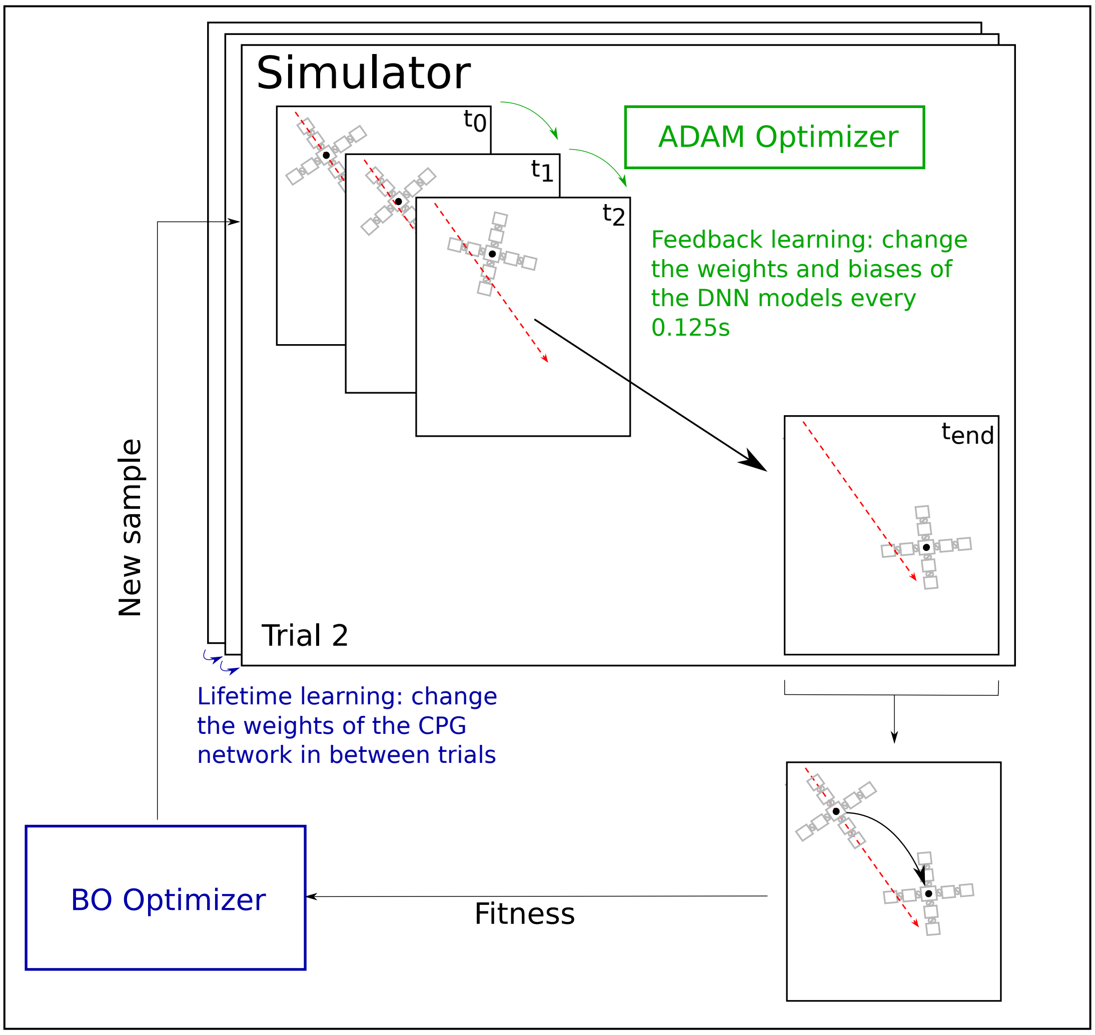

Simulator
t0
t1
t2
ADAM Optimizer
Feedback learning: change the weights and biases of the DNN models every 0.125s
tend
Trial 2
New sample
Lifetime learning: change the weights of the CPG network in between trials
BO Optimizer
Fitness

Figure 4: Overview of the two learning systems. For lifetime learning we evaluate the fitness of the CPG controller after each simulation. Subsequently, the BO algorithm updates the CPG weights as a next sample in between the trials (shown in blue). For feedback learning, we assess the amount of error the robot makes when following a reference state in simulation during each trial. Here, the DNN weights and biases are adapted every 0.125 s by the ADAM optimizer (shown in green). Feedback learning is not present in the open-loop controller.

3.2 Learning Directed locomotion

In the directed locomotion task, the goal is to learn how to maximize the distance traveled in a certain direction within 60 s. This distance measure defines the fitness of the controller and will be determined by a fitness function $\mathcal{F}(\vec{\boldsymbol{w}})$. We use a network of CPGs as our controller, encoded by a vector of weights $\vec{\boldsymbol{w}}$, which determines the behavior of the robot. The BO algorithm attempts to maximize the fitness of the controller by changing the weights in the CPG network. For learning locomotion, the architecture of the CPG network, the BO algorithm, and $\mathcal{F}$ always remain the same.

3.2.1 CPG network

CPGs are used to model the behavior of groups of neurons that were found in the spinal cord of different types of mammals [20; 11]. For controller applications, CPGs are represented by a neuron pair (x_i,y_i) that reciprocally inhibit and excite each other, which results in oscillatory behavior. Here i denotes the specific servomotor that is associated with this CPG (ranging from 1 to the k^{th} servomotor within a single robot). Connected to the x_i neuron is an output neuron (out_i) that computes the input signal ($\phi_{ref,i}$) for a specific servomotor at joint (see Figure 5).

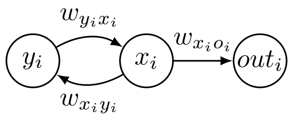


Figure 5: Overview of a single CPG with its neuron pair (x_i, y_i) and output neuron out_i. The connections between each neuron is multiplied by a weight $w_{x_iy_i}$, $w_{y_ix_i}$ and $w_{x_io_i}$ respectively.

The dynamics of a single CPG is solely defined by the current state of its neuron pair. Meaning, that the change in the state of every neuron is coupled by multiplying the current state of the opposite neuron with a weight (w). The weights $w_{x_iy_i}$ and $w_{y_ix_i}$ represent a connection strength between each neuron pair within a CPG, and are responsible for its oscillatory behavior. The weight $w_{x_io_i}$ represents the coupling strength from the x-neuron to the output neuron and defines the CPG output behavior. Changing the motor inputs can thus solely be realized by changing the values of these CPG weights. To enable more complex output patterns we will implement a network of multiple interconnected CPGs to control our robots (see Figure 6).

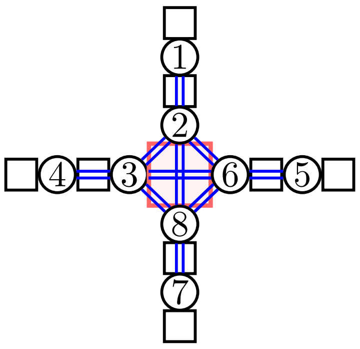


Figure 6: The CPG network that is being used for our Spider. CPGs in the network also influence neighboring CPGs up to 2 blocks distance. This network contains 8 CPGs (numbers) with 10 neighboring pair connections (blue lines).

Earlier work in ER often assigned a single CPG per joint [29; 56; 7], where each CPG would act independently of one another. To increase the complexity of possible motor behaviors we also allow CPG connections to be made between the x-neurons of nearby joints, thereby creating an interconnected CPG network. This means that the dynamics of joint i are also influenced by the state of the CPGs at neighboring joints j through additional coupling denoted as $w_{x_jx_i}$. In this study, the set of neighboring joints ($\mathcal{N}_i$) at joint i, consist of all the joints j positioned within a distance of two-modules (see Figure 6).

From this, the dynamic behavior of (x_i,y_i) can be described with the following two ordinary differential equations.

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

Here we use Newton dot notation to denote derivatives with respect to time. The first part of $\dot{x}_i$ shows the connection of the neuron pair within the CPG. Where, the value of the y-neuron (y_i) is influencing the derivative of x_i with a specific connection strength $w_{y_i x_i}$. *Vice versa* the same holds for calculating $\dot{y}_i$. The second part of $\dot{x}_i$ represents the summed effect of the connections with the x-neurons of neighboring CPGs. $\mathcal{N}_i$ refers to the set of CPG neighbors for joint i, x_j the state value of the neighboring x-neuron, and $w_{x_j x_i}$ their connection strength. The output neuron is a tangent hyperbolic activation function that is only dependent on the x-neuron of its own CPG (see Equation 2).

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

For a given network containing N_C CPGs and $|\mathcal{N}|$ neighboring pairs, the total number of weights to be optimized would equate to $N_w = 3N_C + 2|\mathcal{N}|$. In the case of our Spider (Figure 6) this would result in 44 weights. To simplify the learning task for our search algorithm we define the following symmetries to reduce the total number of weights to be optimized to $N_w = N_C + |\mathcal{N}| = 18$:

$$w_{x_i y_i} = -w_{y_i x_i} \qquad\qquad w_{x_j x_i} = w_{x_i x_j} \qquad\qquad w_{x_i o_i} = 1 \tag{3}$$

At the start of every learning trial, all neuron states are reset to a predefined value $(x, y) = \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$. Subsequently, every 0.125 s each neuron state is updated by Heun integration using the following computation:

$$\begin{bmatrix} \vec{x}_{t+1} \\ \vec{y}_{t+1} \\ \vec{o}_{t+1} \end{bmatrix} = (I + \mathbf{W} \cdot dt) \begin{bmatrix} \vec{x}_t \\ \vec{y}_t \\ \vec{o}_t \end{bmatrix} \tag{4}$$

Here $\vec{x}_t$, $\vec{y}_t$, and $\vec{o}_t$ denotes the vector of all the neuron values in the CPG network at time t, I denotes the identity matrix, dt the integration time step and $\mathbf{W}$ a sparse matrix that contains the coupling weights of all the CPG connections w (in accordance with to the ordinary differential equations mentioned before Equation 1). For a short example of $\mathbf{W}$ and a full derivation of Equation 4 we refer to the Appendix D.

3.2.2 Fitness function

To improve on the task of directed locomotion we need to formulate a fitness function $\mathcal{F}$ that defines the performance of the CPG network. 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 7). With directed locomotion we want the end value of $p(t)$ to be as far as possible in the target direction β_T. The fitness function we apply combines two objectives for directed locomotion: *1)* we want to minimize the amount of deviation from the target direction δ; *2)* maximize the distance travelled in the target direction $p_{end,\perp}$. Fitness can thus be based on the starting position P_0, end position P_{end}, target direction β_T, and actual direction of locomotion β_{end}. We formulate $\mathcal{F}$ as follows:

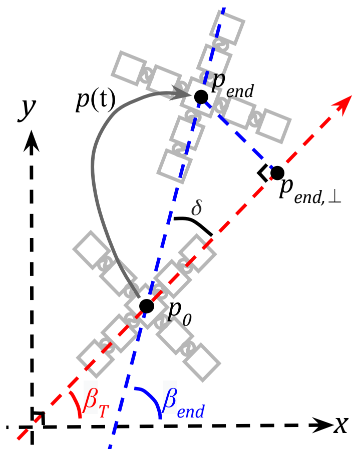


Figure 7: Visualization of the variables necessary for the fitness function. The robot has a target direction β_T (red arrow), and moves from position P_0 to P_{end} in direction β_{end} (blue line). The difference in direction is denoted by angle δ and the projection of P_{end} onto the target direction by $P_{end,\perp}$.

1) Obtain the amount of angular deviation between the actual and target direction, δ:

$$\delta(\beta_0, \beta_1) = \begin{cases} 2\pi - |\beta_1 - \beta_0| & (|\beta_1 - \beta_0| > \pi) \\ |\beta_1 - \beta_0| & (|\beta_1 - \beta_0| \leq \pi) \end{cases} \tag{5}$$

Note that by this definition 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 total distance travelled towards the target direction, $\mathcal{D}_{dir}$:

$$\mathcal{D}_{dir} = ||P_{end} - P_0||_2 \cos\delta \tag{6}$$

$||P_{end} - P_0||_2$ denotes the Euclidean distance between P_{end} and P_0. Since we define our starting point P_0 at the origin, $\mathcal{D}_{dir}$ is equal to the projection of the the end position on the target direction ($P_{end,\perp}$).

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

$$\mathcal{D}_{dev} = ||P_{end} - P_0||_2 \sin\delta \tag{7}$$

4) The fitness function $\mathcal{F}$ is determined as follows:

$$\mathcal{F}(\mathcal{D}_{dir}, \mathcal{D}_{dev}) = |\mathcal{D}_{dir}|\mathcal{D}_{dir} - \mathcal{D}_{dev}^2 \tag{8}$$

Here, the total distance deviated from the target direction ($\mathcal{D}_{dev}$) is squared to punish strong deviations from the target direction, while distance traveled toward the target direction ($\mathcal{D}_{dir}$) is multiplied by its absolute value to encourage movement towards the target direction. The units of our resulting fitness function is $[\mathcal{F}]= m^2$. The value of $\mathcal{F}$ as a function of $\mathcal{D}_{dev}$ and $\mathcal{D}_{dir}$ is shown in the contour plot below (see Figure 8).

3.2.3 Bayesian Optimization

We implement a BO algorithm for the optimization of the CPG weights, which is a state-of-the-art black-box optimizer that has been deployed in both industry science and engineering [73]. The BO algorithm is known for finding optimal solutions with a small number of samples. It does this by having two functionalities: 1) a function approximator $f(\vec{\boldsymbol{w}})$, that models $\mathcal{F}$ to predict the fitness values as a

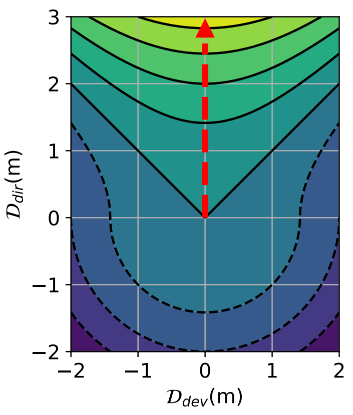


Figure 8: Contour plot of the fitness function values. The black isolines show monotonically increasing fitness values ranging between -6 to 8 with steps of 2, starting from the bottom to top in target direction (*i.e.*, red arrow).

function of the optimization parameters (in our case the CPG weights $\vec{\boldsymbol{w}}$), and 2) an acquisition function $a(f(\vec{\boldsymbol{w}})|\boldsymbol{X},\boldsymbol{Y})$, that uses $f(\vec{\boldsymbol{w}})$ to select a new sample $\vec{\boldsymbol{w}}$ to test given the current data ($\boldsymbol{X}$, $\boldsymbol{Y}$). To prevent the BO algorithm from converging too fast at a local optimum, we first obtain 50 fitness values by Latin Hypercube Sampling (LHS). LHS chooses samples pseudo-randomly to make sure they are evenly distributed in search-space. An overview of the whole optimization process is shown in Appendix E.

In BO, modelling $\mathcal{F}$ is realized by constructing a posterior distribution of functions that best describes the values of $\mathcal{F}$ found so far. In short, we can construct a prior distribution of non-linear functions in search space ($\vec{\boldsymbol{w}}$) by using a kernel. In our BO algorithm we implement the Matérn 5/2 kernel that describes a similarity between all the samples taken so far ($\boldsymbol{X} = [\vec{\boldsymbol{w}}_1, \vec{\boldsymbol{w}}_2, \ldots, \vec{\boldsymbol{w}}_n]$, with $\vec{\boldsymbol{w}}$ denoting the CPG weights of a single sample). Here we can describe the value of $f(\vec{\boldsymbol{w}})$ as a collection of (infinite) random variable, which is called a Gaussian Process $(f(\vec{\boldsymbol{w}}) \sim \mathcal{GP}\left(\mu(\vec{\boldsymbol{w}}), \sigma^2(\vec{\boldsymbol{w}})\right)$. Using the prior and the corresponding measured fitness function values ($\boldsymbol{Y} = [\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_n]$, where $\mathcal{F}$ denotes fitness values of a single sample) a posterior can be described by Bayesian inference. The posterior induces limitations on the non-linear function distribution from the prior and can be used formulate a predictive distribution on the approximated fitness function $f(\vec{\boldsymbol{w}})$ at any point in weight-space, denoted as $p(f(\vec{\boldsymbol{w}})|\boldsymbol{Y}, \boldsymbol{X}, \vec{\boldsymbol{w}}) \sim N\left(\mu\left(f\left(\vec{\boldsymbol{w}}\right)\right), \sigma^2\left(f\left(\vec{\boldsymbol{w}}\right)\right)\right)$. Here, function p describes the predicted distribution at $\vec{\boldsymbol{w}}$ given the current data. Now we have a probabilistic approximation of $\mathcal{F}$ as a function of $\vec{\boldsymbol{w}}$, which has the nice property to inherently define an uncertainty over the predictions that it made. For our kernel the approximated value of the fitness and its corresponding uncertainty is directly dependent on the distance between the sample chosen for prediction and $\boldsymbol{Y}, \boldsymbol{X}$. We refer to [82] for additional reading into Gaussian Processes, different types of kernels and how to use them for optimization.

With the predictions, we can select a new sample from search space to test in our next learning trial. This is done with the second part of the BO algorithm the acquisition function, which bases its selection of the next sample using the $\mathcal{GP}\left(\mu(\vec{\boldsymbol{w}}), \sigma^2(\vec{\boldsymbol{w}})\right)$. As the acquisition function, we choose an Upper Confidence Bound approach (UCB) which will be maximizing the expectation of the posterior distribution given the current observations. In the end, this results in the BO algorithm choosing a sample that either yield a high reward in the approximated objective function or reduces uncertainty in the $\mathcal{GP}$. The pseudo-code for the BO algorithm is presented below. This approach in the BO method is based on previous work (Lan et al., 2020). Below is a description of the hyperparameters that we used (see Table 1).

3.3 Adaptive Internal Model Control

For learning optimal feedback control we want the IMC controller to reduce the number of errors that the DNNs make. We define an error as a deviation of the actual robot state from the reference state, formulated by a loss function $\mathcal{L}$. Here the internal models are adapted by changing a set of weights and biases, which we will refer to as DNN parameters for clarity. The ADAM optimizer [41] is used to update these parameters. Learned DNN parameters are transferred in-between subsequent learning trials, but not between experiments within the same morphology.

Algorithm 1 Bayesian optimization for learning the weights of CPG controllers.

```
1: Initiate $n$ initial samples: $\boldsymbol{X} = [\vec{\boldsymbol{w}}_1, \vec{\boldsymbol{w}}_2, \dots, \vec{\boldsymbol{w}}_n]$    ▷ with $n = 50$ pseudo-random samples by LHS
2: Obtain $n$ initial fitnesses: $\boldsymbol{Y} = [\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_n]$    ▷ using Equation 8
3: Approximate the objective function: $f(\vec{\boldsymbol{w}}) \sim \mathcal{GP}\left(\mu(\boldsymbol{X}), \sigma^2(\boldsymbol{X})\right)$
4: for $k = n+1, n+2, ...$ do    ▷ for evaluation $k$
5:     Select new sample $\vec{\boldsymbol{w}}_k$ based on approximation $f(\vec{\boldsymbol{w}})$:
           $\vec{\boldsymbol{w}}_k = \arg\max_{\vec{\boldsymbol{w}}_k} a\left(p(f(\vec{\boldsymbol{w}})\,|\boldsymbol{Y}, \boldsymbol{X}, \vec{\boldsymbol{w}})\right)$
6:     Obtain the new sample fitness $\mathcal{F}_k(\vec{\boldsymbol{w}}_k)$
7:     Append the data $\boldsymbol{X} = [\boldsymbol{X}, \vec{\boldsymbol{w}}_k]$,   $\boldsymbol{Y} = [\boldsymbol{Y}, \mathcal{F}_k]$
8:     Update the approximation: $f(\vec{\boldsymbol{w}}) \sim \mathcal{GP}\left(\mu(\boldsymbol{X}), \sigma^2(\boldsymbol{X})\right)$
9: end for
10: return data $\boldsymbol{X}, \boldsymbol{Y}, f(\vec{\boldsymbol{w}})$
```

Table 1: Hyperparameters of the BO algorithm

Parameters	Value	Description
Initial samples	50	Number of initial samples.
learning iterations	250	Number of evaluations, excluding initial samples.
Kernel variance	1.0	Kernel variance in Matérn 5/2 kernel.
Kernel length	0.2	Characteristic length-scale in Matérn 5/2 kernel.
UCB alpha	3.0	Weight in the acquisition function.
Initial sampling	LHS	Method used to generate initial sampling.

3.3.1 DNNs as internal models

For the internal models we will use two different DNN. The DNN for the inverse model consists of an input 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. Furthermore, we use two hidden layers of the same size ($4k$) with Rectified Linear activation units (ReLU). The output layer contains k neurons (output layer model$_{inv}$ = $[u_{1:k}]^T$), that sends control signals to the servomotors. For the output layer we choose the same tangent hyperbolic neuron as in Equation 2, which makes u_i range between $[-1, 1]$ (as a normalized range for the servomotor torque).

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 neurons, and an output layer of $2k$ neurons (output layer model$_{ff}$ = $[\phi_{pred,1:k}, \dot{\phi}_{pred,1:k}]^T$). For the output layer we also use the tangent hyperbolic function as our neurons. The model$_{ff}$ tries to predict the next robot state normalised ($\phi_{pred,1:k}$ and $\dot{\phi}_{pred,1:k}$).

3.3.2 Learning optimal feedback control

The implementation of the DNN is realized using libtorch v1.4.0 (downloaded April 2020), which is a C++ distribution of the popular PyTorch library[4]. Only at the start of an experiment (*i.e.* the beginning of the first trial) we randomly initialize the DNN parameters by sampling from a Gaussian distribution with zero mean and variance one. The models in the IMC vanilla are continuously updated by changing the DNN parameters in the hidden and output layers. For the IMC reservoir, the DNN parameters of the hidden layers are frozen after initialization, and adaptations are only done on the output layer. The DNN parameters are updated whenever a new reference signal is sent to the IMC (every 0.125 s). 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, and 2) the model$_{ff}$ correctly predicting the next robot state. The DNNs learn optimal feedback control by minimizing the sum of errors made

[4]`https://pytorch.org`

by the internal models, measured by a loss function $\mathcal{L}$ (see Equation 9). Each internal model has its own error function based as shown in Equation 10. For the model$_{inv}$ the desired output is the reference state, with the error being the difference between the reference and the actual robot state ($\phi - \phi_{ref}$). For the 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 ($\phi_{pred} - \phi$). The loss function for each model is their respective summed square error.

$$\mathcal{L} = error \frac{1}{2} error^T \tag{9}$$

$$\begin{aligned} error_{inv} &= \left[\phi_1 - \phi_{ref,1}, \dots, \phi_k - \phi_{ref,k}, \dot{\phi}_1 - \dot{\phi}_{ref,1}, \dots, \dot{\phi}_k - \dot{\phi}_{ref,k}\right] \\ error_{ff} &= \left[\phi_{pred,1} - \phi_1, \dots, \phi_{pred,k} - \phi_k, \dot{\phi}_{pred,1} - \dot{\phi}_1, \dots, \dot{\phi}_{pred,k} - \dot{\phi}_k\right] \end{aligned} \tag{10}$$

To update the DNN parameters we use the ADAM optimizer [41]. By changing the weights we can define the gradient of $\mathcal{L}$ with respect to each weight. The ADAM algorithm uses this gradient to estimate the running averages of the gradient and its moments (for a full derivation of all the ADAM functions see [22]). The estimated running average of the gradient and moments are subsequently used to update the weights. For regularization, we implemented L2 weight decay. A pseudo-code of the IMC optimization is shown below. The hyperparameters for the ADAM optimizer are given in Table 2. A more detailed overview of the IMC adaptation scheme and the ADAM optimizer is presented in the Appendix F.

Algorithm 2 IMC update rule for timestep $t = 0, 1, 2, \dots, t_{end}$

```
1:  for $t = 0, 1, 2, \dots, t_{end}$ do
2:      Obtain the reference state vector: $\vec{\boldsymbol{y}}_{ref,t} = \text{CPG}(t)$        ▷ with $\vec{\boldsymbol{y}} = [\phi_{1:k}, \dot{\phi}_{1:k}]$
3:      Obtain the current state vector: $\vec{\boldsymbol{y}}_t = proprioception(t)$
4:      Calculate the prediction error: $error_{ff,t} = \vec{\boldsymbol{y}}_t - \vec{\boldsymbol{y}}_{ref,t}$
5:      Calculate the movement error: $error_{inv,t} = \vec{\boldsymbol{y}}_{pred,t} - \vec{\boldsymbol{y}}_t$
6:      Update the models: ADAM(model$_{inv}$, $error_{inv,t}$); ADAM(model$_{ff}$, $error_{ff,t}$)
    Calculate for next iteration step
7:      Calculate the motor input: $\vec{\boldsymbol{u}}_t = \text{model}_{inv}(\vec{\boldsymbol{y}}_{ref,t}, \vec{\boldsymbol{y}}_t, error_{move,t})$
8:      Predict the next state: $\vec{\boldsymbol{y}}_{pred,t+1} = \text{model}_{ff}(\vec{\boldsymbol{y}}_t, \vec{\boldsymbol{u}}_t)$
9:      Obtain the next state $\vec{\boldsymbol{y}}_{t+1} = \text{Robot}(\vec{\boldsymbol{y}}_t, \vec{\boldsymbol{u}})$
10: end for
```

Table 2: Hyper parameters of the ADAM optimizer

Parameters		Value	Description
ADAM:	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

A summary of the most important distinctions between our controller conditions is presented in Table 3. For all conditions, directed locomotion in a CPG network is learned using BO to adjust the weights of the CPG network. The number of CPG weights are dependent on the total number of servomotors (k) and neighbor pairs ($|\mathcal{N}|$). In addition, the IMC controllers also learn feedback control with their two DNN models using the ADAM optimizer. The distinction between the IMC vanilla and the IMC reservoir shows in the number of parameters that are adapted during feedback learning.

Table 3: Summary of the different controller conditions. The open-loop controller based on [45] is also implemented in the two IMC controllers, which differ in the number of parameters to optimize.

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

3.4 Statistical analysis

In addition to the learning task, we also want to validate our feedback controller. This will be done by retesting the best controllers in a noisy environment (more on this in Section 3.4.2). In the end, we conduct two types of experiments. First, testing the open-loop, IMC vanilla, and IMC reservoir. Second, validating the best feedback control in a noisy environment. A clear overview of the difference between the first IMC test and this feedback validation test is provided in Appendix G.

3.4.1 Testing the effects of IMC control

For the data analysis in the directed locomotion task, we compare the progression of the fitness as a function of evaluations, between all controllers (OL, IMC vanilla, IMC reservoir) per robot. per robot morphology, all 10 experiments are aggregated to allow for a robust comparison of the results. The progression of the fitnesses as a function of learning trials will be plotted to see differences in learning curves. For statistical analysis, differences between the end-values of the objective function will be compared for each controller within a certain robot morphology. Additionally, we group these statistics to find general differences in learning between the OL, IMC vanilla, and IMC reservoir controller. Per robot morphology statistical differences are tested using an independent samples *t-test* ($N_e = 10$), while for the grouped comparison statistical differences are determined by a paired-samples *t-test* ($N_r = 6$). Assumptions on normality and equal variance will be verified by performing a Shapiro–Wilk test (with normality assumed for $p > 0.95$) and an F-test (requiring a normal ratio between variances $\frac{1}{2}$ and 2). Additional report on the effects size will be made by using Cohens-d, for which magnitude of mean differences are as described by [70], *small*: $d = 0.20$, *medium*: $d = 0.50$, *large*: $d = 0.80$, *very large*: $d = 1.20$, *huge*: $d > 2.0$.

3.4.2 Validating the feedback controller

To validate our feedback controller, we want to compare the performance of the best IMC controller (either vanilla or reservoir) in a noisy environment with the performance of the open-loop controller in a noisy environment. We do this by selecting the best controller per run for each morphology and retesting it. In total we will re-evaluate 120 controllers in this noisy environment, *i.e.* the best controllers of 10 runs for each of the 6 subjects in the open-loop and the best IMC controller.

As additional noise, we will add a Gaussian distributed perturbation to the servomotor input signal every time the control signal gets updated. The noise signal will have a mean of zero and a standard deviation of 0.05, which equates to 5% of the maximum torque. The control signal+noise will be sent to the motor input signal where it will drive the servomotor.

To gain insight into the changes in fitness due to the added noise we will provide a box-plot of the fitness values per morphology before and after adding noise. Additionally, we will plot the amount of distance traveled towards ($\mathcal{D}_{dir}$), and deviated from the target direction ($\mathcal{D}_{dev}$), which are the variables that directly influence the fitness as formulated in Equation 8.

For statistical analysis, we compare the new fitnesses with noise to the original ones without calculating the difference ($\Delta\text{fitness} = \text{fitness}_{original}\text{-fitness}_{noise}$). A comparison between the mean Δfitness of the open-loop controller and the best IMC within each morphology will be done using independent samples *t-test* ($N_e = 10$). To see if the IMC controller better controls for disturbances, we will compare the aggregated mean Δfitness for all morphologies in a paired samples *t-test* ($N_r = 6$). Additionally, we will report on the effect size using Cohens-d.

4 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 off simulated time this equates to a 20× speedup. In Figure 9 below we plotted the progression of the mean fitness (±95% confidence interval) of all six robots as a function of learning evaluations. Here we can see that learning occurs in all morphologies for all controllers, open-loop in blue, IMC vanilla in red, and IMC reservoir in green. The highest mean fitness value at the end of the evaluations is obtained by the gecko morphology with the IMC reservoir controller, while the lowest mean fitness end-value is found in the 6677 with open-loop control.

When looking at the overall trend of each morphology we can see that the IMC reservoir performed better or just as good as the other controllers. Furthermore, the open-loop performed similarly to the IMC reservoir in most morphologies 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 about the same for all controllers at the start of the learning task (evaluations < 100), except for the Gecko in which the open-loop and IMC reservoir learn much faster than the IMC vanilla. After about 100 evaluations learning diverges for the Spider, BabyA, and BabyB.

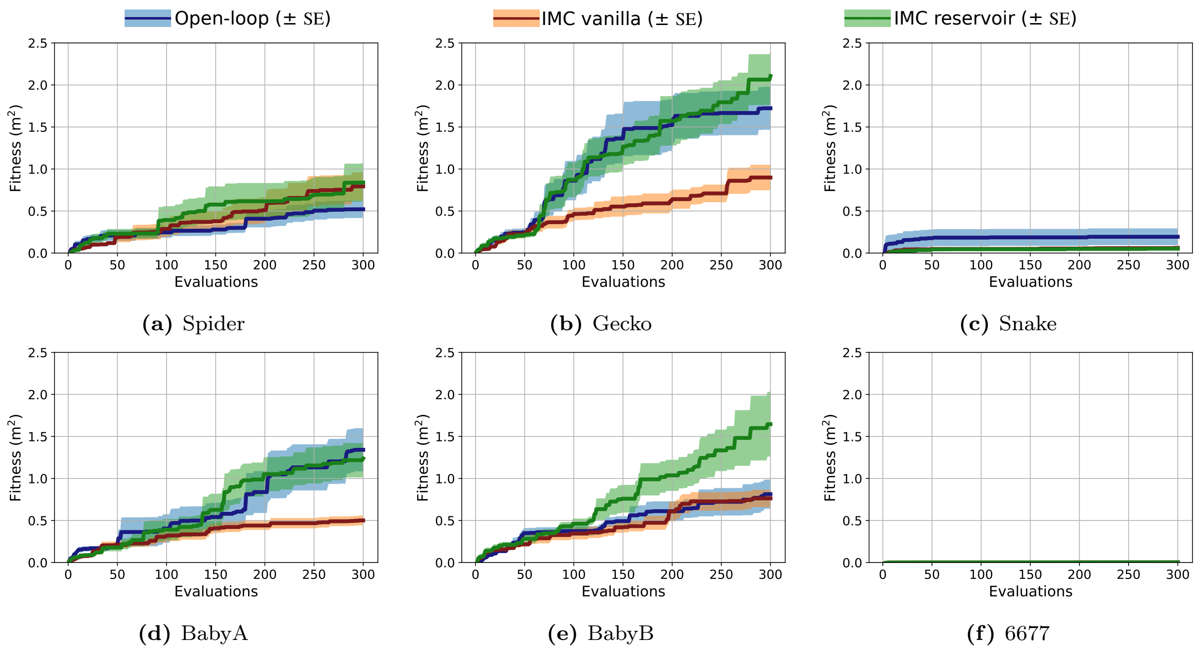


Figure 9: Results of the locomotion task for the Spider, Gecko, Snake, BabyA, BabyB, and 6677. The blue lines denote the mean fitness ($N_e = 10$) of each robot for the open-loop controller as a function of samples, while the red line denotes the mean fitness of the IMC vanilla controller, the green lines show the results for the IMC reservoir. The similarly colored areas indicate the SE at for the aggregated data.

From the results of Figure 9 we took the end-values of all the fitnesses (so after the last evaluation) to do our data analysis. Group analysis on the end-values met the requirements for the assumption of normality (Shapiro-Wilk $p > 0.95$), and homogeneity of variance ($\frac{1}{2} < F$-test< 2) between the different groups that were being compared. First, we performed an independent samples *t-test* between the different controller for each robot morphology (see Figure 10). Statistical significance was set at $p < 0.05$ (with d.f. $= 18$) which is denoted with a star *. Additionally, effect sizes were calculated using Cohens-d.

In Figure 10 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).

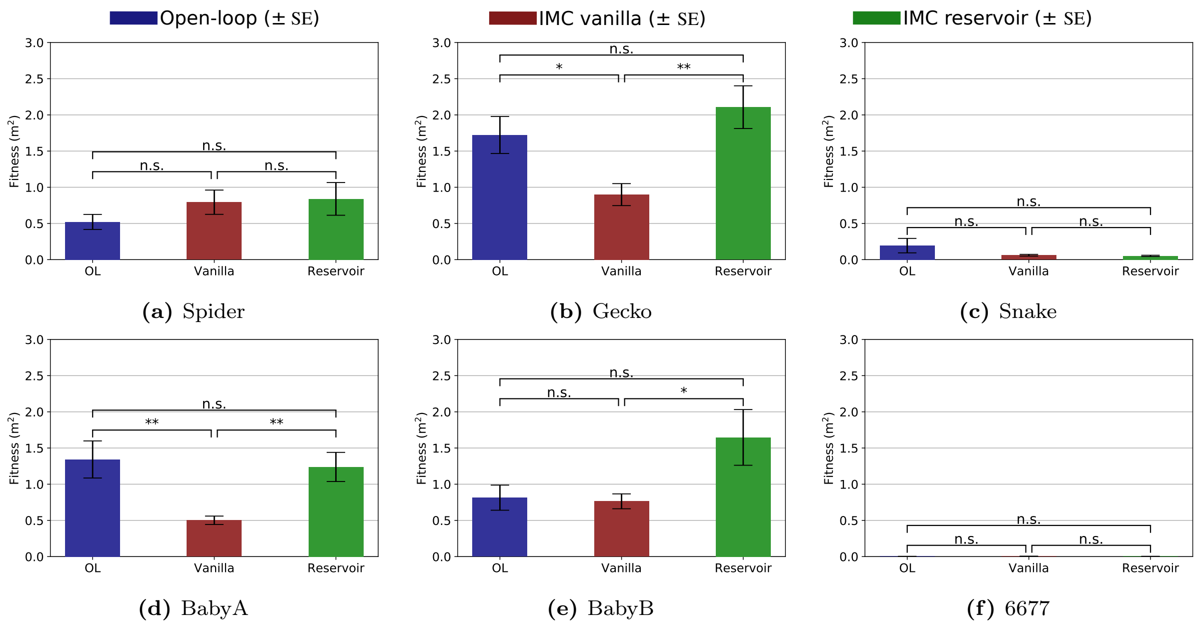


Figure 10: Bar plot of the mean end-values of the fitnesses ($N_e = 10$) of each robot: Spider, Gecko, Snake, BabyA, BabyB, and 6677. The blue bars denote the open-loop controller, red bars denote the IMC vanilla, and green bars denote the IMC reservoir. Error bars are plotted as well and significant differences are denoted by * for $p < 0.05$, and ** for $p < 0.01$.

To test for general differences between the controllers we calculated mean end-values of the fitnesses (±STD) over all morphologies (OL: 0.77 ± 0.61; IMC vanilla: 0.50 ± 0.35; IMC reservoir: 0.98 ± 0.78). Subsequently, a paired samples *t-test* was performed between these groups to check for statistically significant differences ($N_r = 6$). A summary of the grouped controller statistics of the fitness end-values is shown below in Table 4. In the end, we did not find any significant difference between any of the aggregated controller groups: the OL vs. IMC vanilla controller: $p = 0.19$; OL vs. IMC reservoir: $p = 0.23$; and IMC reservoir vs. IMC vanilla $p = 0.08$.

Table 4: Grouped comparison of the mean end-values of the fitnesses (±STD) for each controller (OL vs. IMC vanilla vs. IMC reservoir). For statistical analysis we performed a paired samples *t-test* between each group. The corresponding p-values (d.f. = 10) are shown underneath. We define significant difference at $p < 0.05$ (denoted with *). Effect size was calculated using Cohens-d.

Controllers	OL	IMC vanilla	OL	IMC reservoir	IMC reservoir	IMC vanilla
	0.77 ± 0.61	0.50 ± 0.35	0.77 ± 0.61	0.98 ± 0.78	0.98 ± 0.78	0.50 ± 0.36
p-value	0.22		0.22		0.08	
Cohens-d	0.52		0.31		0.79	

Based on the results from the end-values of the fitnesses we choose to compare the open-loop to the IMC reservoir controller in the additional experiment with added noise. The boxplot of the fitness values, the amount of distance traveled towards ($\mathcal{D}_{dir}$), and the amount of distance deviated from the target direction ($\mathcal{D}_{dev}$) are plotted below (see Figure 11). Here we see can that for most morphologies the fitness and $\mathcal{D}_{dir}$ decreases, while $\mathcal{D}_{dev}$ increases, when adding noise. We can also see that the decrease in fitness is more apparent for the open-loop control than the IMC reservoir in most morphologies. As an exception to the rule, we see that BabyA does not seem to be affected that much by the added noise.

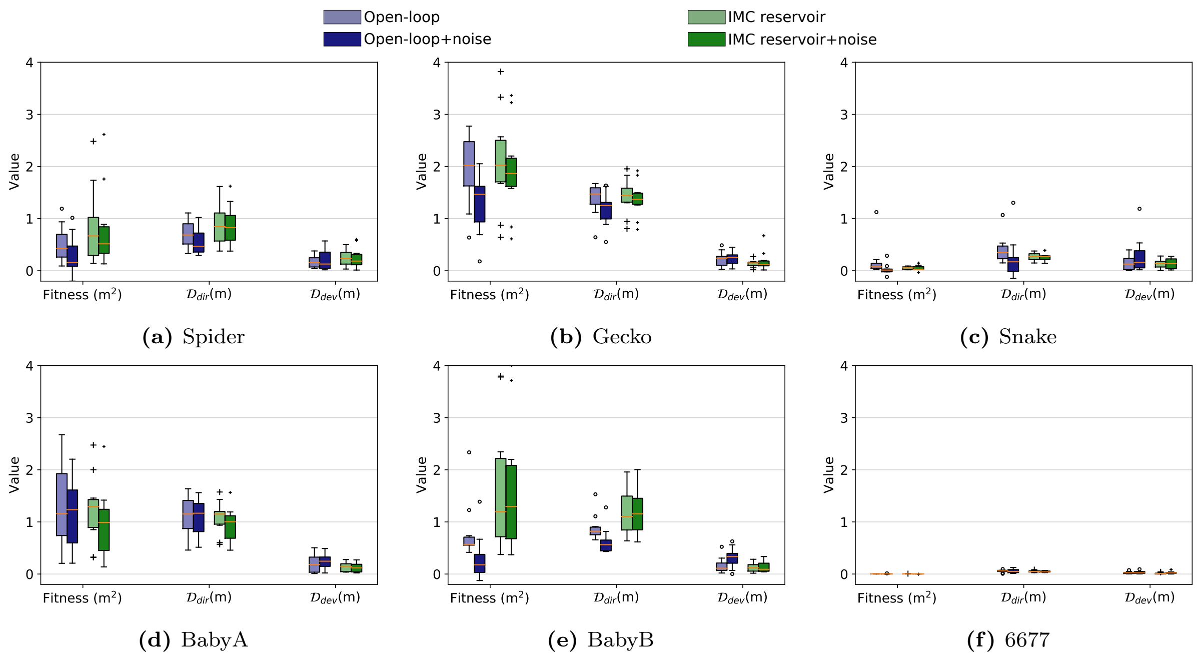


Figure 11: Results from retesting the best controllers in each run for the open-loop (blue) and the IMC reservoir (green) with added noise. For comparison, we also included the corresponding noiseless results (open-loop: light blue, and IMC reservoir: light green). Boxplots of the fitness values, the amount of distance traveled towards ($\mathcal{D}_{dir}$), and the amount of distance deviated from the target direction ($\mathcal{D}_{dev}$) for each morphology. The orange line denotes the median value, the box indicates the bounds of 50% of the population, and the whiskers 100% bounds when omitting outliers.

The results of the statistical analysis of the added noise experiments are shown in Table 5. It should be noted that a negative difference indicates a decrease in fitness after adding noise. Here we can see that for most morphologies the average fitness decreased after adding noise (except for the 6677 Δopen-loop, and BabyB Δ IMC reservoir for which the fitness slightly increased). Furthermore, we found that there was a statistically significant difference in the mean Δfitness between the open-loop and the IMC reservoir controller for the Gecko ($p = 0.008$, Δopen-loop: -0.66 $\pm$ 0.42, and ΔIMC reservoir: -0.18 $\pm$ 0.25, effect size= 1.40), Snake ($p = 0.03$, IMC vanilla: -0.19 $\pm$ 0.23, and IMC reservoir: -0.46 $\pm 3.62 \cdot e^{-2}$, effect size= 1.10), and BabyB ($p = 2.56 \cdot e^{-6}$, IMC vanilla: -0.51 $\pm$ 0.20, and IMC reservoir: $4.14 \pm 0.11 \cdot e^{-4}$, effect size= 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 ($p = 0.08$).

Table 5: Per robot mean difference (±STD) in performance after adding noise to the best controllers of each experiment ($N_e = 10$). We compare the differences in fitness values for the original and added noise condition between the open-loop (Δopen-loop) controller and the IMC reservoir (ΔIMC reservoir). It should be noted that a negative difference means that the fitness decreased after adding noise. For statistical analysis we implement an independent samples *t-test* for the 10 runs within a specific robot morphology (d.f. = 18), while for the grouped statistics we implement a paired samples *t-test* (d.f. = 10) is conducted to for all morphologies aggregated. We define significant difference at $p < 0.05$ (denoted with *). Effect size was calculated using Cohens-d.

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

5 Discussion

Many controllers that are evolved in ER literature seem to have a problem with bridging the reality gap [33]. In this paper, we addressed this issue in two ways simultaneously with our IMC implementation. First of all, we learned directed locomotion on more realistic robots that use feedback control instead of open-loop control. Secondly, the added feedback control might result in more robust movement execution in the sense that the effects of perturbations and noise on the robots are actively being reduced. This led to the formulation of the following two research questions: 1) How does the addition of the IMC controllers affect learning? 2) Is the IMC controller able to retain a higher level of fitness in a noisy environment? In short, our results do indicate that the addition of feedback control can improve the system (*i.e.* the learning performance in robots). The more sophisticated implementation of the IMC reservoir can have a positive effect on the achieved speed and the robustness of the controller w.r.t. an open-loop controller in several morphologies. Furthermore, a simple sub-optimal implementation of feedback control (IMC vanilla) can be detrimental to the learning process. These results show the importance of well-thought feedback design

At the start of learning, we see similar learning curves for all controllers (Figure 9). This is as expected because for the first 50 evaluations learning should occur randomly due to the LHS initialization that was implemented for all controller conditions. A divergence in the rate of learning is visible after this initial sampling in the Spider, Gecko, BabyA, and BabyB. This divergence starts around 100 evaluations where the less performing controllers begin to plateau in performance. The plateauing indicates that the end-values of the fitnesses truly represent differences in task-performance of each controller, and were not caused by stopping the experiments too soon. In further support of our findings, it seems that the differences in the end-values could have been more pronounced since learning in the IMC reservoir was still apparent in most morphologies at the end of the experiments.

For statistical analysis on the directed locomotion task, we compared the end-values of the fitnesses between controllers. Here we found that for the BabyA and Gecko both the open-loop and IMC reservoir controllers performed similarly and significantly better than the IMC vanilla. If we look at the effect size we can see that these differences are large to very large [70], which indicates a strong difference between these controllers. We also found a statistically significant difference between the IMC vanilla and the IMC reservoir for the BabyB with a very large effect size. This was not the case for the open-loop IMC and reservoir in the BabyB (even though open-loop performed similarly to the IMC vanilla), nevertheless it should be noted that a trend was visible $p = 0.08$ with a large effect size.

Overall, we did not find any significant difference between the grouped mean fitness end-values. We suspect that this is mainly due to the low number of subjects in our study (6 robots). 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 indicates that we could have increased the number of subjects while still retaining a medium to large effect size. Furthermore, we suspect that the differences could have been more pronounced if we had continued learning. When we look at the learning curves (Figure 9) we can see that the IMC did not plateau yet in some morphologies (Spider, Gecko, BabyB) compared to the others. Additionally, the absence of a significant effect might also be caused by the choice of the fitness function. We can always design a fitness function that would lead to a significant difference, but our choice for a quadratic $\mathcal{F}$ might have been poor. If we instead took a linear function (*e.g.* $\sqrt{\mathcal{F}}$, with unit m) than the differences the between fitnesses would be more pronounced (especially for the Snake and 6677).

The low fitnesses for the Snake and the 6677 morphologies grabbed our attention as it greatly reduced the size of the mean difference between the controllers (see Table 5). To us, this poor performance was quite surprising since these morphologies performed (very) well in previous work [55; 57]. Visual inspection of the simulated robots revealed that learning did occur in these robots, but that the direction of locomotion was sub-optimal for these models (video[5]). Both morphologies seemed to require a rotation of 90 degrees first before they could properly locomote in the target direction. Earlier work showed that one of the optimal movement strategies for the Snake was to roll sideways, while the 6677 performed well by hopping in a similar direction [56; 57]. Quite remarkably the highest fitness in the Snake showed a behavior in which the open-loop controller was able to make the snake turn 90 degrees before it started to roll sideways (resulting in an end-fitness of $1.03\,\mathrm{m}^2$). In retrospect, rotating the morphologies might have been sufficient to increase the fitnesses tremendously. Rerunning the experiments for the Snake and 6677 in the 'correct' direction resulted in similar performances as the Gecko and BabyB respectively (not shown).

The use of adaptive IMC has some interesting consequences for lifetime learning with BO, due to the additional feedback learning of the internal models. As the internal models change over time, we

[5]`https://youtu.be/TgCOgHII7mg`

can see that the IMC controllers might behave inconsistently. This means that if we would retest the same sample a second time, after a period of feedback learning, we would likely see a slight difference in behavior and thus fitness. As a consequence, instead of a crisp representation of the fitnesses in search space, we should consider each data point more as a probability when dealing with adaptive control. Luckily the BO algorithm is well equipped to do this as it in itself evaluates fitness values samples with a certain uncertainty. Nevertheless, we did not put any extra effort into fine-tuning the BO algorithm for this inconsistency of control as we wanted to keep all learning equal between the controllers.

Another property that can cause inconsistency in control is the fact that learning occurs in two internal models simultaneously. When learning is done on two systems simultaneously we create the risk that the adaptations of both systems become unpredictable and unstable [9]. Especially when the models depend on each other's output. This is because the changes that occur in one of the systems (in our case the DNNs) can also influence the adaptations in the other. A clear way to show this inter-dependency of the DNNs in our IMC is by imagining a special case in which one of the internal models is perfect. Here it can happen that when the other model is not perfect an error will emerge, which creates a gradient that directs the weights away from the perfect model. Unfortunately, we did not focus on the convergence of the two adaptive models when we designed our IMC controllers. Nevertheless, we were able to reliably learn feedback control. For future work, it might be better to learn each model differently, for example, switch which model learns for some 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).

Besides the two internal models that are being optimized for optimal feedback control, we also implemented adaptations in the weights of the CPGs for the lifetime learning task of directed locomotion. Here we recognize a problem that can lead to over-fitting. Namely, as the CPG network keeps improving its control output we will likely see that the robot behavior converges to a (local) optimum. During feedback learning, early convergence can cause the DNNs to recognize which output is desired based on the robot state instead of the reference signal presented by the CPG network. When exploring a new behavior the controller will function badly since it did not learn how to follow the reference state correctly. 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 the internal models with L2 regularization on the DNN weights [22]. Furthermore, the LHS should also induce a wide variety of movements to prevent over-fitting. Another possible solution might be to change the BO in such a way that prevents early convergence of the robot behavior (increase exploration). Additionally, we could have implemented some form of dropout in the DNN [74] or reduce the size of the models as well. The problems arising from over-fitting seem similar to those regarding continual learning in [50]. Here it has been found that models show a poor ability to adapt to new problems quicker and have difficulties in retaining acquired knowledge (also known as *catastrophic forgetting*). As a solution, it has been proposed to memorize and learn on specific samples (evenly distributed in input space) rather than continuously learning on all samples, which may cause convergence if samples become increasingly similar [71]. For our internal models, we could implement a similar strategy by memorizing less frequent states to train our DNN.

In the end, the inconsistency of control can cause the resulting fitnesses to be badly represented by the $\mathcal{GP}$ [73]. When bad samples are regarded higher and/or good samples lower we find that the sample efficient nature of the BO algorithm becomes counterproductive, as it will negatively affect the likelihood of samples being picked in the neighborhood of those points. The sensitivity to this bad sampling is different for each morphology as the optimal region of samples can be big or small. A morphology that has a narrow region with high fitnesses in search space would be more sensitive than one with a wider region of high performance. Another reason why inconsistency in control can lead to bad approximation by the $\mathcal{GP}$ is that high variability in neighboring points can cause a very erratic and/or uncertain models. This is highly dependent on the amount of data and the type of kernel that is being used (kernels differ in the amount of smoothing of the fitness data [82]). Inconsistency in control is not a unique property for our IMC controller but will also be apparent in any type of feedback controller and for any robot in a noisy environment as well. For future work, it might be better to take this into account when learning locomotion in real robots using BO. An interesting option might be to relate the total amount of loss (feedback error of the internal models, $\mathcal{L}$) during a trial in the uncertainty of that sample for the BO algorithm.

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 learning seemed detrimental to its performance in directed locomotion, while for the IMC reservoir it seemed to have a positive effect. The main difference between the two DNN is that the reservoir computing technique has been ascribed to have a reduced amount of learning capabilities and an increased rate of learning [76]. The above-mentioned problems regarding inconsistency in control and over-fitting of the DNN may explain why there seems to be an optimal amount of learning for the internal models. For example, to prevent over-fitting, we would like to have a slow rate of learning on a model with low learning capacity. On the other hand, to counteract bad adaptations towards new behaviors and limitations in the complexity of the model we would like to have a higher learning rate and model capacity. Furthermore, the effects of the inconsistency of control between evaluations of the BO can be reduced by a slower learning rate of the DNNs, but this will also mean that the inconsistencies take longer to fade away.

The fact that the IMC reservoir seems to outperform 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* conversely is beneficial, but we disagree. Even though higher valued good samples might attract more samples in their neighborhood, it can also lead to converging faster on a local optimum. Furthermore, bad samples that are evaluated worse would have been regarded as bad and ignored by the BO anyway, 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, therefore, suspect the source of improvement to lie somewhere else. A possible consequence of the inconsistency in control might be an increase in the amount of uncertainty of the $\mathcal{GP}$. This could cause exploration to be encouraged, which would result in a longer period of learning and less convergence on a specific behavior. If we look at the learning curves we can see that the IMC reservoir seems to plateau much later than the rest, which would also be as expected when exploration is encouraged more. It would also be possible to look at the spread of the samples in search space where we would expect a higher variance for the controllers with more exploration.

From the added noise experiments we can observe that the IMC reservoir controller was able to reduce the effects of the perturbation more than the open-loop in the Gecko, Snake, and BabyB morphology. Here, the added noise condition served as a way to validate the feedback controller but can also be seen as a proxy for the reality gap. Overall a trend in better performing IMC was visible for $p = 0.08$. For the individual comparison, two results stand out the most: 1) the effect of the noise was most apparent in the BabyB with a huge effect size ($d = 3.1$), and 2) the open-loop retained a high performance on the BabyA. We suspect that these findings are caused by the robustness of the learned behavior. Visual inspection of the BabyA controllers with and without noise (video is the same as mentioned before) did not reveal anything noticeable. For the BabyB controllers, we noticed that most controllers learned to row with both long sidearms while steering with the long arm at the front. This behavior results in the robot balancing on two points of contact for a brief moment in time. Adding noise made the robot fall differently in the open-loop controllers which resulted in a deviation from the target direction. If we look at Figure 11 we can see that most reduction in fitness was caused by this deviation from the target direction (decrease in $\mathcal{D}_{dir}$ and increase in $\mathcal{D}_{dev}$). Similarly, for the open-loop snake, the reduction in performance of the best movement behavior in the open-loop Snake (the one that rotated 90 degrees first) was mainly caused by the fact that the rotation was not executed properly before rolling. This is in line with the findings that open-loop control can lead to exploitation of control behaviors that are very sensitive to perturbations and/or the reality gap.

Even though we did not find any significant difference between the two controllers in the grouped added noise environments, we do believe that overall the IMC reservoir feedback tends to perform better. We base this conclusion on the fact that the difference is close to significant ($p = 0.08$) and the effect size is very large ($d = 1.24$). Similarly to the grouped statistics of the mean fitness end-values, we suspect that increasing the sample size, changing the fitness function or loading the 6677 and Snake in different orientations would have been sufficient to see significant differences.

When considering real-world application of ER (whether it be transferring controllers from simulation to real and/or conducting a real-world ER experiment), one should keep in mind that most techniques are developed in a simulated environment [61]. This leads to difficulties when transferring controllers from simulation to real robots, the so-called reality gap [33]. If we place our study in the context of existing ER work we provide strong reasons for rethinking the types of controllers that are being optimized. The results presented here revealed that small differences in these controllers can already lead to very different results in the performance of current learning algorithms. Furthermore, our bio-inspired controller has shown to be a solution to the reality gap by being more robust in performance after transferring to a noisy environment and being more realistic to real-world control systems compared to the open-loop control.

ER experiments in the real world show that PID control can work for simple morphologies [65; 5; 81; 64]. Nevertheless, we believe that there is a need for adaptive feedback control like our IMC architecture. As morphologies become increasingly complex so does the control necessary to accommodate such designs. Exciting progress in ER is being made, in the additional evolution of building materials [25], non-modular shapes [23], and even biomaterials [44]. Furthermore, the use of non-static control allows for cooping with reality gaps, damage during the life of a robot, and changing environments. We could even use our internal models to improve the simulator as well. The implementation of adaptive feedback control seems, therefore, a logical step to take in ER.

Here we have shown that our IMC design can provide adaptive feedback control without affecting learning too much. Furthermore, lifetime learning in ER challenges us to develop controllers that can learn rapidly on a broad range of morphologies while being eligible for evolution. A unique selling point of our bio-inspired IMC design is that we can accommodate these demands while being easy to implement on already existing controllers. In comparison, most PID controllers are limited in their capabilities for complex morphologies and require elaborate fine-tuning [3]. Other nonlinear feedback controllers [40; 47; 43] are often highly customized and require redesigning for different morphologies. This makes the IMC design attractive for future ER experiments. In future work, it might be interesting to see how well the IMC works on a set of completely different robots outside of the Revolve framework.

A major limitation in our study is the absence of real-world testing, even though we consider our controller to be better transferable to real robots. Unfortunately, limitations in resources prevented the use of real robots. With the current experiments, we have shown that the IMC reservoir can learn with a similar rate as the open-loop controller of [45], and retain its performance better in a noisy environment. As a logical next step, we would want to test lifetime learning with our IMC reservoir in a real-world experiment, which seems feasible given the low amount of simulated time. Another limitation of our study lies in the fact that we purposely did not reduce the reality gap by implementing solutions found in other work (as mentioned in the introduction) [33; 42; 78]. As a consequence, we may have overestimated the effect of the IMC reservoir on its ability to reduce the reality gap. We choose to disregard these techniques as we were interested to see the capabilities of the IMC controller only. Using more techniques would have occluded how well our controller could reduce the reality gap in our robots. Additionally, these techniques would also affect both learning systems in a non-obvious way. Although it would be interesting to see how such techniques would affect the internal models, this would have been beyond the scope of this study.

Additionally, we may have limited the IMC performance by keeping the architecture simple and fixed. We purposely designed the DNNs simple and only compared different learning capabilities to keep the difference as clear as possible. In the future we might want to optimize our DNN designs more, or make them eligible for evolution as well through NEAT [75]. Adding to that is the possibility of placing a filter on the error signal before feeding it back in the IMC. This has been done in the industry, as error signals often need some form of smoothing and filtering to prevent instability caused by noise and high peaks [21]. Since the model$_{inv}$ output layer was bounded by Equation 2, we did not expect such high peaks (even during the added noise condition). Furthermore, we expected that our IMC would learn to adapt its control adequately to prevent instabilities.

The last shortcoming we want to address is not retesting the IMC vanilla to see if the transferability is similar to the IMC reservoir. Although it could have strengthened our position on the value of the IMC feedback control in general, we want to emphasize that if performance was worse it would have been difficult to explain the results. Both IMCs were modeled differently making it hard to identify the possible causes for a difference in performances. Therefore we tested the validity of the feedback controller to only apply to the IMC reservoir.

This study aimed to see if IMC was a viable option for feedback control during lifetime learning. The necessity of a learning loop in ER is based on the difficulties that arise with the simultaneous body and brain evolution [6; 16]. Other proposed methods often rely on preserving novelty after morphological mutations [7]. [7] noted that for larger creatures, novelty protection was only beneficial with a minimum mutation threshold (which was obtained by a parameter sweep). The advantage of lifetime learning over such methods is the scalability to more complex morphologies since such tuning is not required beforehand. Unfortunately, we could not show this advantage since we omitted the evolutionary process in this study. For future work, we should consider an experiment in which both body and brain evolve, with additional lifetime learning during a robot life using our IMC control.

6 Conclusion

Limitations of current ER research lies in bridging the reality gap for controllers that learn locomotion in simulation. With our bio-inspired IMC controller, we showed that the addition of adaptive feedback control can both affect the amount of learning as well as the transferability of the final controllers. Our controller is special in the sense that it could learn proper movement execution on top of locomotion in a feasible amount of time for a broad range of morphologies.

References

[1] Aoi, S., Manoonpong, P., Ambe, Y., Matsuno, F., and Wörgötter, F. (2017). Adaptive control strategies for interlimb coordination in legged robots: a review. *Frontiers in neurorobotics*, 11:39.

[2] Åström, K. J. and Hägglund, T. (1995). *PID controllers: theory, design, and tuning*, volume 2. Instrument society of America Research Triangle Park, NC.

[3] Babuška, R. and Kober, J. (2019). *Knowledge-Based Control Systems*. Delft University of Technology.

[4] Bongard, J., Zykov, V., and Lipson, H. (2006). Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121.

[5] Brodbeck, L., Hauser, S., and Iida, F. (2015). Morphological evolution of physical robots through model-free phenotype development. *PloS one*, 10(6):e0128444.

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

[7] Cheney, N., Bongard, J., SunSpiral, V., and Lipson, H. (2018). Scalable co-optimization of morphology and control in embodied machines. *Journal of The Royal Society Interface*, 15(143):20170937.

[8] Dasgupta, D. and Michalewicz, Z. (2013). *Evolutionary algorithms in engineering applications*. Springer Science & Business Media.

[9] Datta, A. and Ochoa, J. (1996). Adaptive internal model control: Design and stability analysis. *Automatica*, 32(2):261–266.

[10] Dietz, V. (2002). Proprioception and locomotor disorders. *Nature Reviews Neuroscience*, 3(10):781.

[11] Dietz, V. (2003). Spinal cord pattern generators for locomotion. *Clinical Neurophysiology*, 114(8):1379–1389.

[12] Doncieux, S. and Mouret, J. B. (2014). Beyond black-box optimization: A review of selective pressures for evolutionary robotics. *Evolutionary Intelligence*, 7(2):71–93.

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

[14] D'Angelo, M., Weel, B., and Eiben, A. E. (2013). Online gait learning for modular robots with arbitrary shapes and sizes. In *International Conference on Theory and Practice of Natural Computing*, pages 45–56. Springer.

[15] Economou, C. G., Morari, M., and Palsson, B. O. (1986). Internal model control: Extension to nonlinear system. *Industrial & Engineering Chemistry Process Design and Development*, 25(2):403–411.

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

[17] Eiben, A. E. and Hart, E. (2020). If it evolves it needs to learn. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, pages 1383–1384.

[18] Eiben, A. E. and Smith, J. (2015). From evolutionary computation to the evolution of things. *Nature*, 521(7553):476–482.

[19] Falotico, E., Vannucci, L., Ambrosano, A., Albanese, U., Ulbrich, S., Vasquez Tieck, J. C., Hinkel, G., Kaiser, J., Peric, I., Denninger, O., et al. (2017). Connecting artificial brains to robots in a comprehensive simulation framework: the neurorobotics platform. *Frontiers in neurorobotics*, 11:2.

[20] Forssberg, H., Grillner, S., Halbertsma, J., and Rossignol, S. (1980). The locomotion of the low spinal cat. ii. interlimb coordination. *Acta Physiologica Scandinavica*, 108(3):283–295.

[21] Garcia, C. E. and Morari, M. (1982). Internal Model Control. 1. a Unifying Review and Some New Results. *Industrial and Engineering Chemistry Process Design and Development*, 21(2):308–323.

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

[23] Hale, M. F., Buchanan, E., Winfield, A. F., Timmis, J., Hart, E., Eiben, A. E., Angus, M., Veenstra, F., Li, W., Woolley, R., et al. (2019). The are robot fabricator: How to (re) produce robots that can evolve in the real world. In *The 2018 Conference on Artificial Life: A Hybrid of the European Conference on Artificial Life (ECAL) and the International Conference on the Synthesis and Simulation of Living Systems (ALIFE)*, pages 95–102. MIT Press.

[24] Harvey, I., Di Paolo, E., Wood, R., Quinn, M., and Tuci, E. (2005). Evolutionary robotics: A new scientific tool for studying cognition. *Artificial Life*, 11(1-2):79–98.

[25] Howard, D., Eiben, A. E., Kennedy, D. F., Mouret, J.-B., Valencia, P., and Winkler, D. (2019). Evolving embodied intelligence from materials to machines. *Nature Machine Intelligence*, 1(1):12–19.

[26] Huang, J.-Q. and Lewis, F. L. (2003). Neural-network predictive control for nonlinear dynamic systems with time-delay. *IEEE Transactions on Neural Networks*, 14(2):377–389.

[27] Hunt, K. and Sbarbaro, D. (1991). Neural networks for nonlinear internal model control. In *IEE Proceedings D (Control Theory and Applications)*, volume 138(5), pages 431–438. IET.

[28] Hupkes, E., Jelisavcic, M., and Eiben, A. E. (2018). Revolve: a versatile simulator for online robot evolution. In *International Conference on the Applications of Evolutionary Computation*, pages 687–702. Springer.

[29] Ijspeert, A. J. (2008). Central pattern generators for locomotion control in animals and robots: a review. *Neural networks*, 21(4):642–653.

[30] Ijspeert, A. J., Crespi, A., Ryczko, D., and Cabelguen, J.-M. (2007). From swimming to walking with a salamander robot driven by a spinal cord model. *science*, 315(5817):1416–1420.

[31] Inoue, K., Sumi, T., and Ma, S. (2007). 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*, pages 1957–1962. IEEE.

[32] Ivanenko, Y. P., Poppele, R. E., and Lacquaniti, F. (2004). Five basic muscle activation patterns account for muscle activity during human locomotion. *The Journal of physiology*, 556(1):267–282.

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

[34] Jelisavcic, M., De Carlo, M., Haasdijk, E., and Eiben, A. (2016). Improving RL power for on-line evolution of gaits in modular robots. In *2016 IEEE symposium series on computational intelligence (SSCI)*, pages 1–8. IEEE.

[35] Jelisavcic, M., De Carlo, M., Hupkes, E., Eustratiadis, P., Orlowski, J., Haasdijk, E., Auerbach, J. E., and Eiben, A. E. (2017a). Real-world evolution of robot morphologies: a proof of concept. *Artificial life*, 23(2):206–235.

[36] Jelisavcic, M., Glette, K., Haasdijk, E., and Eiben, A. E. (2019). Lamarckian evolution of simulated modular robots. *Frontiers in Robotics and AI*, 6:9.

[37] Jelisavcic, M., Kiesel, R., Glette, K., Haasdijk, E., and Eiben, A. E. (2017b). Analysis of lamarckian evolution in morphologically evolving robots. In *Artificial Life Conference Proceedings 14*, pages 214–221. MIT Press.

[38] Katz, P. S. (2016). Evolution of central pattern generators and rhythmic behaviours. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 371(1685):20150057.

[39] Kawato, M. (1999). Internal models for motor control and trajectory planning. *Current opinion in neurobiology*, 9(6):718–727.

[40] Kawato, M., Furukawa, K., and Suzuki, R. (1987). A hierarchical neural-network model for control and learning of voluntary movement. *Biological cybernetics*, 57(3):169–185.

[41] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[42] Koos, S., Mouret, J.-B., and Doncieux, S. (2012). The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, 17(1):122–145.

[43] Korkmaz, D., Ozmen Koca, G., Li, G., Bal, C., Ay, M., and Akpolat, Z. H. (2019). Locomotion control of a biomimetic robotic fish based on closed loop sensory feedback cpg model. *Journal of Marine Engineering & Technology*, pages 1–13.

[44] Kriegman, S., Blackiston, D., Levin, M., and Bongard, J. (2020). A scalable pipeline for designing reconfigurable organisms. *Proceedings of the National Academy of Sciences*, 117(4):1853–1859.

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

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

[47] Li, H.-X. and Deng, H. (2006). An approximate internal model-based neural control for unknown nonlinear discrete processes. *IEEE transactions on neural networks*, 17(3):659–670.

[48] Lipson, H. and Pollack, J. B. (2000). Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974.

[49] Liu, C., Chen, Y., Zhang, J., and Chen, Q. (2009). Cpg driven locomotion control of quadruped robot. In *2009 IEEE International Conference on Systems, Man and Cybernetics*, pages 2368–2373. IEEE.

[50] Lopez-Paz, D. and Ranzato, M. (2017). Gradient episodic memory for continual learning. In *Advances in neural information processing systems*, pages 6467–6476.

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

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

[53] McNamee, D. and Wolpert, D. M. (2019). Internal models in biological control. *Annual review of control, robotics, and autonomous systems*, 2:339–364.

[54] Miall, R. C. and Wolpert, D. M. (1996). Forward models for physiological motor control. *Neural networks*, 9(8):1265–1279.

[55] Miras, K. and Eiben, A. (2019). Effects of environmental conditions on evolved robot morphologies and behavior. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 125–132.

[56] Miras, K., Haasdijk, E., Glette, K., and Eiben, A. (2018a). Effects of selection preferences on evolved robot morphologies and behaviors. In *Artificial Life Conference Proceedings*, pages 224–231. MIT Press.

[57] Miras, K., Haasdijk, E., Glette, K., and Eiben, A. (2018b). Search space analysis of evolvable robot morphologies. In *International Conference on the Applications of Evolutionary Computation*, pages 703–718. Springer.

[58] Miyamoto, H., Kawato, M., Setoyama, T., and Suzuki, R. (1988). Feedback-error-learning neural network for trajectory control of a robotic manipulator. *Neural Networks*, 1(3):251–265.

[59] Nelson, A. L., Barlow, G. J., and Doitsidis, L. (2009). Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345–370.

[60] Nguyen-Tuong, D. and Peters, J. (2011). Model learning for robot control: a survey. *Cognitive processing*, 12(4):319–340.

[61] Nichele, S. (2015). The coevolution of robot controllers ("brains") and morphologies ("bodies")–challenges and opportunities. Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.696.2391&rep=rep1&type=pdf`, accessed (2020/06/10).

[62] Nolfi, S., Floreano, D., and Floreano, D. D. (2000). *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines.* MIT press.

[63] Nordmoen, J., Nygaard, T. F., Ellefsen, K. O., and Glette, K. (2019). Evolved embodied phase coordination enables robust quadruped robot locomotion. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 133–141.

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

[65] Pollack, J. B. and Lipson, H. (2000). The golem project: Evolving hardware bodies and brains. In *Proceedings. The Second NASA/DoD Workshop on Evolvable Hardware*, pages 37–42. IEEE.

[66] Popa, L. S. and Ebner, T. J. (2019). Cerebellum, predictions and errors. *Frontiers in cellular neuroscience*, 12:524.

[67] Prabhu, S. G. R., Seals, R. C., Kyberd, P. J., and Wetherall, J. C. (2018). A survey on evolutionary-aided design in robotics. *Robotica*, 36(12):1804–1821.

[68] Ryu, J.-K., Chong, N. Y., You, B. J., and Christensen, H. I. (2010). Locomotion of snake-like robots using adaptive neural oscillators. *Intelligent Service Robotics*, 3(1):1.

[69] Samuelsen, E. and Glette, K. (2015). Real-world reproduction of evolved robot morphologies: Automated categorization and evaluation. In *European Conference on the Applications of Evolutionary Computation*, pages 771–782. Springer.

[70] Sawilowsky, S. S. (2009). New effect size rules of thumb. *Journal of Modern Applied Statistical Methods*, 8(2):26.

[71] Shin, H., Lee, J. K., Kim, J., and Kim, J. (2017). Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2990–2999.

[72] Sims, K. (1994). Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM.

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

[74] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

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

[76] Tanaka, G., Yamane, T., Héroux, J. B., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., and Hirose, A. (2019). Recent advances in physical reservoir computing: A review. *Neural Networks*, 115:100–123.

[77] Tin, C. and Poon, C.-S. (2005). Internal models in sensorimotor integration: perspectives from adaptive control theory. *Journal of Neural Engineering*, 2(3):S147.

[78] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE.

[79] Tuthill, J. C. and Azim, E. (2018). Proprioception. *Current Biology*, 28(5):R194–R203.

[80] Vargas, P. A., Di Paolo, E. A., Harvey, I., and Husbands, P. (2014). *The horizons of evolutionary robotics*. MIT press.

[81] Vujovic, V., Rosendo, A., Brodbeck, L., and Iida, F. (2017). Evolutionary developmental robotics: Improving morphology and control of physical robots. *Artificial life*, 23(2):169–185.

[82] Williams, C. K. and Rasmussen, C. E. (2006). *Gaussian processes for machine learning*, volume 2(3). MIT press Cambridge, MA.

[83] Winfield, A. F. and Timmis, J. (2015). Evolvable robot hardware. In *Evolvable Hardware*, pages 331–348. Springer.

[84] Wolpert, D. M., Miall, R. C., and Kawato, M. (1998). Internal models in the cerebellum. *Trends in cognitive sciences*, 2(9):338–347.

A Triangle of Life Background

Most of the research in ER only involves the evolution of controllers with fixed robot morphologies [61]. However, for a full-fledged robotic evolution system, the robot controller and morphology should be evolved together [12]. [61]. In the few studies with simultaneous controller and body evolution, we found that real-world testing only takes place at the end, as a means of validation [65]. This is because the simultaneous evolution of both body and brain is challenging, since the inherited brain of a newborn robot may not match its inherited body even if the parents perform well (a so-called body-brain mismatch). It has therefore been argued that a real-world application of a full-fledged ER system (evolving both brain and body) should also introduce an additional learning loop at the early stage of a robot life [17]. The ToL conceptualizes what it would take for a full-fledged ER to take place in the real world (see Figure 12).

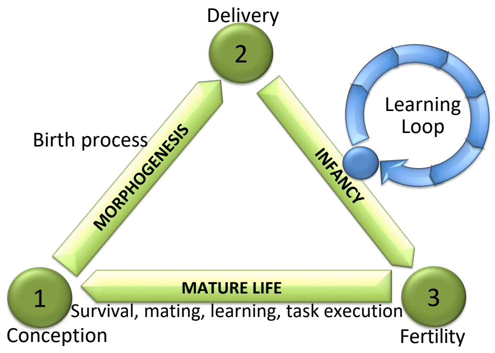


Figure 12: Triangle of Life, adapted from [16]. Here the lifecycle of a robot starts when we conceptualize its design. Subsequently, the robot is born either by hand or in an automated process. To properly assess the performance of a robot, its controller will first be optimized in an additional learning loop (similar to an infant learning to walk). At last in the mature stage, the robot will perform its predefined task and becomes eligible for selection by the evolutionary algorithm

Conception, signals the start of the ToL in which a new robot design is formulated. From here on a birthing process takes place in which the body is actualized. In current literature, this process is mostly done manually, which is very time-consuming. The ToL envisions an automated robot birthing clinic that could speed up the assembly process. The use of modules with 3D printing can enable the pre-fabrication of parts for rapid prototyping, and robotic assembly may allow for a fully automated birthing process. Currently, there are some labs that are working on developing such an automated birthing clinic [5; 81].

Delivery, denotes the birth of a newborn baby robot and is followed by an infancy stage. Our study investigates the learning loop that takes place during this stage. The ToL suggests that learning should take place in a safe environment, a so-called nursery. In a nursery, the robot will learn some fundamental motor tasks like locomotion and grasping under direct supervision in a shielded environment. The nursery has two important advantages for real-world applications. 1) learning reduces the effect of an initial body-brain mismatch, and 2) very weak individuals (ones that do not perform well on the fundamental motor tasks) can be filtered out and recycled early on in the evolutionary process. In the end, this will result in more efficient use of time and resources.

Fertility, starts at the moment a robot is deemed good enough in the fundamental motor tasks. From here on, the mature robot is released into the world to accomplish its predefined task (*e.g.* collecting resources) with the skills it learned during infancy. Additionally, the robot is now eligible for mating as well, thereby starting a new loop when it is selected for reproduction.

B The Bio-inspired IMC

In biological systems, information about joint angles, velocities, and torques are provided by different sensors of the proprioceptive system [79]. The proprioceptive system relays its sensory information back to the central nervous system where it is used for movement control [10]. A long-lasting hypothesis on how proprioceptive feedback is used by humans states that the nervous system constructs predictive models of the body in the physical world to guide behavior [53]. These predictive models are internal representations of the body in the real world based on proprioceptive feedback. To guide behavior two predictive models are necessary for the movement of the limbs. The first model (called the inverse model) computes the correct motor-input to the muscles in order for the body to move as intended. The second model (called the feedforward) computes the expected proprioceptive feedback based on the motor-input of the inverse model. The expected proprioceptive output is sent as an efferent copy to the motor neurons where it is used as a way to detect deviations in the actual movement. This flow of information is similar to the IMC in our robots [77].

The internal model hypothesis in neuroscience has driven research into finding a specific brain region that could serve as body representations. For the inverse model, it has been found that the cerebellum plays an important role in the computation of motor inputs for the muscles [40; 52]. Namely, sensory information about the current state is transferred into the cerebellum via mossy fibers and encode specific commands for certain joints through granular cells and parallel fibers. Subsequently, Purkinje cells have been proposed to encode for the dynamic signals to produce motor input commands.

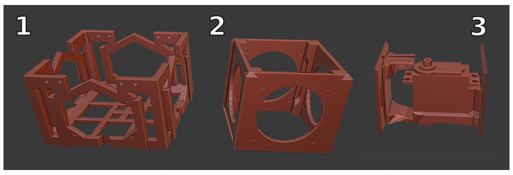
Evidence for the existence of a forward model in the cerebellum has been less direct [54]. Data from previous studies showed that the cerebellum is concerned with processing sensory re-afference [84]. More recent work suggests that the output of the Purkinje cells can both function as a motor input and a prediction of proprioceptive feedback [66]. Furthermore, climbing fibers are assumed to carry information about movement errors back to the Purkinje cells to close a feedback loop.

Unfortunately, the similarity between our IMC design and the brain stops here. In real biological systems, the sensory output is defined by large groups of neurons that encode information by firing rates. Our implementation of ReLU units falls short to accommodate such an encoding mechanism. Furthermore, in the cerebellum, one can see that computations occur by deploying many more neurons at the input level (mossy fibers), often with much more redundancy between them, converging with less connectivity to a small number of output neurons (*i.e.* a network with a sparsely connected funnel shape). Lastly, learning itself occurs differently in a biological brain (through Hebbian learning) in contrast to the back-propagation of an error signal in artificial neural networks like our DNNs.

C Revolve, Robots, and Other Technicalities

The robot evolution simulator Revolve is developed in earlier work as a way to enable ER experiments within the ToL framework in simulation [28]. A major part of the ToL is that besides the evolution of both body and brain, additional learning takes place after the birth of a robot. In the case of a population of learning robots, this would mean that learning is also influenced by interactions between robots. After a quick assessment, it was found that pre-existing evolution simulators could not accommodate such requirements, which launched the development of Revolve[6]. The Revolve library consists of tools to create robots throughout an evolutionary experiment. This library includes ready to go experimental setups, as well as setups that require more in-dept experimental specification. For example, the robot genome (for both body and/or brain), its phenotype (into construct-able robots), which evolutionary operators, different types of robotic frameworks, fitness functions, and robot environment can all specified. Additionally, Revolve allows developed controllers to be used onto real robots.

In our experiment, we used Revolve to create the robots that were subjected to learning in simulation. These robots consist of modular parts that make them applicable to evolution. The modules are based on the RoboGen framework and are designed in such a way that they can be bought/3d printed in the real world. In the end, we only implemented a subset of RoboGen modules, see Figure 13a: 1. The main core, consisting of a battery, micro-controller with an IMU; 2. Fixed blocks; and 3. Servomotor Towerpro MG996R (with a range between -45, and 45 degrees). For the servomotor module, the axis of rotation is positioned in the middle. We redesigned the RoboGen modules in such a way to accommodate for the Towerpro servomotor, as we want to implement our IMC in real robots in the future (see Figure 13b, video here[7]). The physical properties of these new blocks are listed below in Table 6. We assume symmetry in each dimension and homogeneity for all modules.



(a) Modules

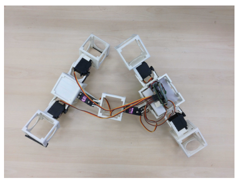

(b) Gecko

Figure 13: **(a)** Robot modules: 1. main core component; 2. fixed brick; 3. servomotor. **(b)** A real version of the Gecko robot.

Table 6: Physical properties of the modules used in the robots

Parameters		Core Component	Fixed Brick	Servomotor
Dimensions w×d×h	(mm)	89×89×60.3	63×63×60.3	103.5×53×53
Mass	(gr)	250	30	69
Ang range	(rad)	-	-	$\pm 0.25\pi$
Max. Ang vel	(deg/s)	-	-	5.24
Max. torque	(Nm)	-	-	0.92

Building a robot in Revolve is simple. A robot architecture is defined in blocks on a 2D lattice that can differ in orientation. Only one module can be placed in the position of a block on the lattice. This block can be connected at 4 sites (north, east, south, west). These sites correspond with one of the four faces of a module. Different rotations between modules are in steps of 90 degrees.

After the creation of our robot, Revolve will place it in the Gazebo[8] simulator. In Gazebo we signal the robot servomotors using our IMC controller. The update rate (0.125 s) of the controller was set in such a way that it is feasible in a Raspberry PI micro-controller extended with a custom HAT. In Gazebo we use the Open Dynamics Physics Engine[9] in which each module is described as a physical body with a bounding box. The Revolve code with the modules and robot designs is publicly available here.

[6]https://github.com/ci-group/revolve
[7]https://youtu.be/iS1pEOF4Ur4
[8]http://gazebosim.org
[9]https://www.ode.org/

D Building a CPG Network

The dynamics of a single CPG as described by Ijspeert [29] is shown in Equation 1 without the summed term for the x-neuron. Because these weights have static values, we can describe the dynamics as a system of linear equations in the following form:

$$\begin{aligned} v &= \begin{bmatrix} x \\ y \\ o \end{bmatrix} \\ \frac{dv}{dt} &= \begin{bmatrix} 0 & w_{yx} & 0 \\ w_{xy} & 0 & 0 \\ w_{xo} & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ o \end{bmatrix} = \boldsymbol{w}v \end{aligned} \tag{11}$$

With x and y representing the value of their respective neuron state and o the signal that is send to the output neuron. This system representation can be easily extended to encompass two CPGs:

$$\begin{aligned} \vec{v} &= \begin{bmatrix} \vec{x} \\ \vec{y} \\ \vec{o} \end{bmatrix} \qquad \text{with,} \qquad \begin{aligned} \vec{x} &= [x_1, x_2]^t \\ \vec{y} &= [y_1, y_2]^t \\ \vec{o} &= [o_1, o_2]^t \end{aligned} \\ \frac{d\vec{v}}{dt} &= \begin{bmatrix} 0 & 0 & w_{y_1x_1} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{y_2x_2} & 0 & 0 \\ w_{x_1y_1} & 0 & 0 & 0 & 0 & \\ 0 & w_{x_2y_2} & 0 & 0 & 0 & \\ w_{x_1o_1} & 0 & 0 & 0 & 0 & \\ 0 & w_{x_2o_2} & 0 & 0 & 0 & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \\ o_1 \\ o_2 \end{bmatrix} = \boldsymbol{W}\vec{v} \end{aligned} \tag{12}$$

From here on introducing more CPGs will lead to similar extension of the matrix by adding weights diagonally. Creating a connection between the x-neurons of the two CPGs can be done as well:

$$\frac{d\vec{v}}{dt} = \begin{bmatrix} 0 & w_{x_2x_1} & w_{y_1x_1} & 0 & 0 & 0 \\ w_{x_1x_2} & 0 & 0 & w_{y_2x_2} & 0 & 0 \\ w_{x_1y_1} & 0 & 0 & 0 & 0 & \\ 0 & w_{x_2y_2} & 0 & 0 & 0 & \\ w_{x_1o_1} & 0 & 0 & 0 & 0 & \\ 0 & w_{x_2o_2} & 0 & 0 & 0 & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \\ o_1 \\ o_2 \end{bmatrix} = \boldsymbol{W}\vec{v} \tag{13}$$

To calculate the values of the CPGs in the network at a next time step we can do the following:

$$\begin{aligned} \vec{v}_{t+1} &= \vec{v}_t + \frac{d\vec{v}_t}{dt} \cdot dt \\ \vec{v}_{t+1} &= \vec{v}_t + \boldsymbol{W}\vec{v}_t \cdot dt \\ \vec{v}_{t+1} &= (I + \boldsymbol{W} \cdot dt)\,\vec{v}_t \end{aligned} \tag{14}$$

With I denoting the identity matrix. Equation 14 is equal to the Equation 4 from the report.

E Learning Locomotion

Learning Locomotion is done through the optimization of the CPG weights ($\vec{w}$) using BO. First, we initialize 50 samples pseudo-randomly using LHS and test them in the simulator. Subsequently, the BO algorithm approximates the fitness function $\mathcal{F}$ by using Gaussian Processes. A schematic overview of the learning directed locomotion loop is shown below (Figure 14). For the mathematical implementation of the acquisition function (upper confidence bound, UCB) and the $\mathcal{GP}$ approximation kernel (Matérn 5/2) we refer to [82].

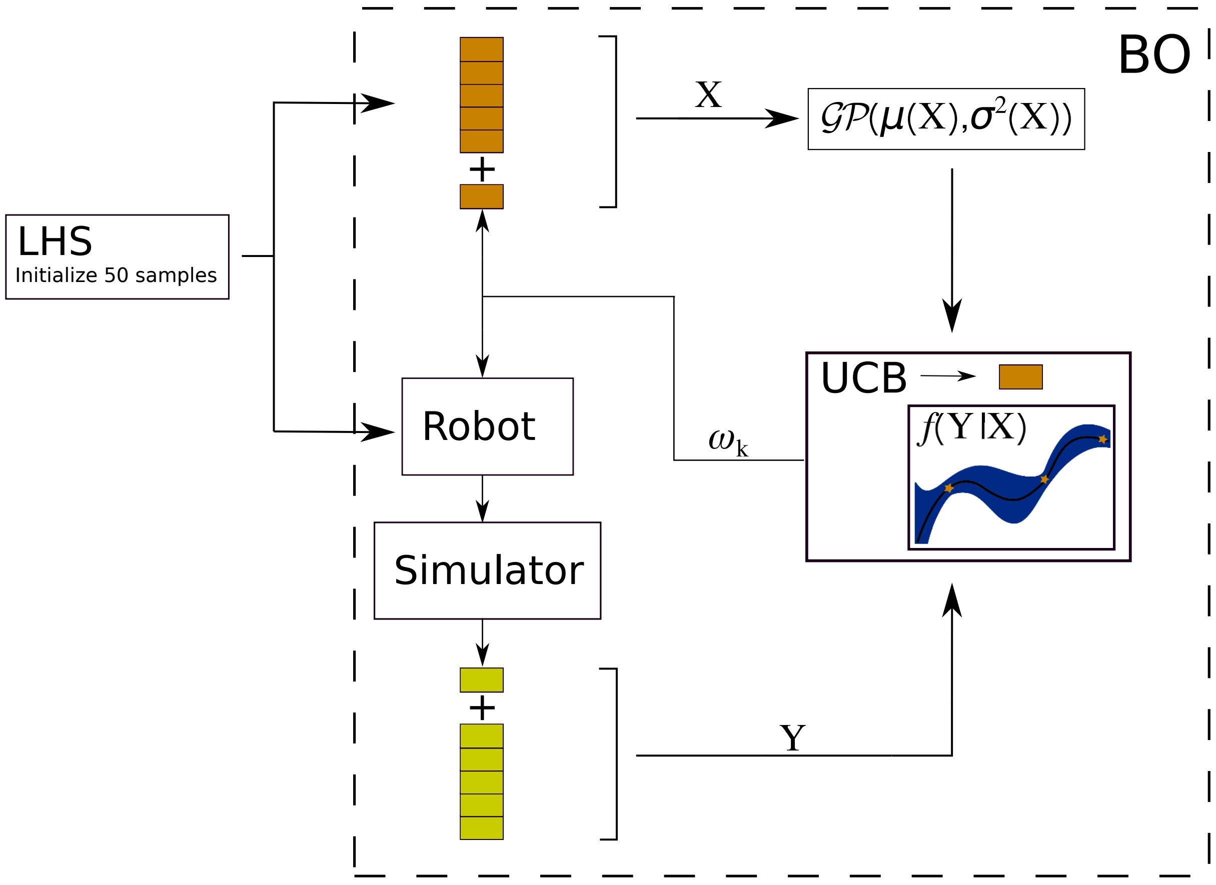


Figure 14: Schematic overview of the learning directed locomotion loop. At first we initialize our data-set, by evaluating 50 samples in weight space using Latin Hypercube Sampling. The initial samples (orange blocks, $\boldsymbol{X} = [\vec{\boldsymbol{w}}_1, \vec{\boldsymbol{w}}_2, \dots, \vec{\boldsymbol{w}}_n]$) and their corresponding fitness values (yellow blocks, $\boldsymbol{Y} = [\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_n]$) are used to approximate the fitness function (f) using Gaussian Processes ($\mathcal{GP}$). With this approximation the Upper Confidence Bound (UCB) approach selects a new sample for the subsequent learning trial ($\boldsymbol{w}_k$).

F Learning the Internal Models

Learning the two internal models is done by adapting the DNN of each internal model every time the CPG output is updated (which is 0.125 s) in simulation. Adaptation of the internal models is done by using two different error functions ($error_{inv}$ and $error_{ff}$). For the model$_i nv$ the error is the difference between the reference state and the current state at the next time step. For the model$_f f$ the error is the difference between the predicted state and the current state at the next time step. A schematic overview of the information flow for updating the DNN is shown in Figure 15.

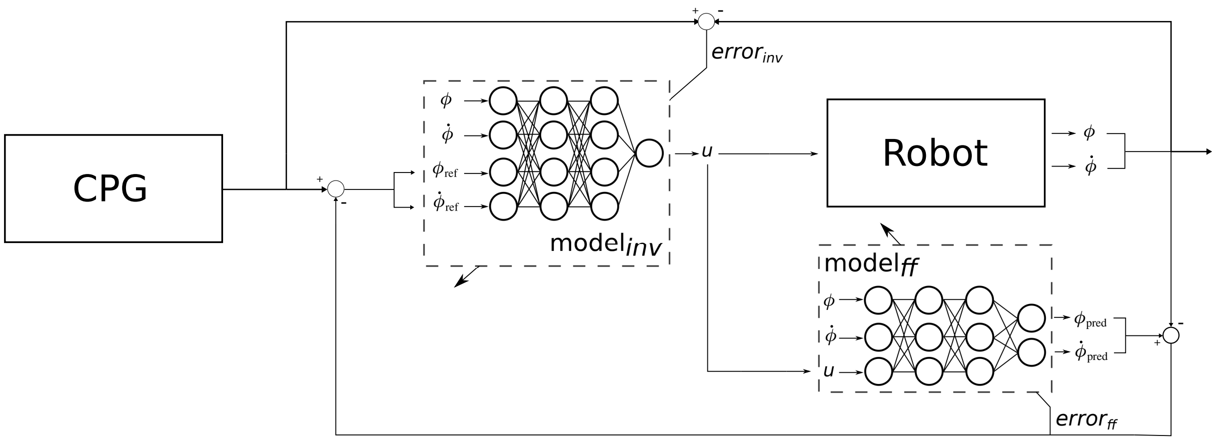


Figure 15: Schematic overview of the information flow for updating the DNN. It should be noted that we omitted the flow of the current state (robot output) to the model inputs for the sake of clarity.

With the error functions, we can formulate the loss as mentioned in Equation 9. In the ADAM optimization algorithm, running averages of both the gradients of the loss function with respect to the network weight ($\nabla_w \mathcal{L}$) and the second moments of the gradients are used ($\nabla_w \mathcal{L} \otimes \nabla_w \mathcal{L}$). We use the same calculations as formulated by [41].

Calculate the moving average of the first (m) and second (v) moments of $\mathcal{L}$ at time t:

$$\begin{aligned} m_w^{t+1} &\leftarrow \beta_1 m_w^t + (1-\beta_1)\nabla_w \mathcal{L}^t \\ v_w^{t+1} &\leftarrow \beta_2 v_w^t + (1-\beta_2)\nabla_w \mathcal{L}^t \otimes \nabla_w \mathcal{L}^t \end{aligned} \tag{15}$$

Implement bias correction:

$$\begin{aligned} \hat{m}_w &= \frac{m_w^{t+1}}{1-\beta_1} \\ \hat{v}_w &= \frac{v_w^{t+1}}{1-\beta_2} \end{aligned} \tag{16}$$

Update the weights accordingly:

$$w^{t+1} = w^t - \alpha \frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \epsilon} \tag{17}$$

G Overview Experimental Setup

In the end, our experimental setup consists of 2 tests. In the first test, we learn directed locomotion in all 6 robots with all 3 controllers. Learning is done in 300 learning trials. An overview of the first test is shown in Figure 16. In the second test, we retest the best fitness from each experiment in all 6 robots for the Open-loop and the best IMC controller, while we add noise in the simulator. An overview of the second test is shown in Figure 17.

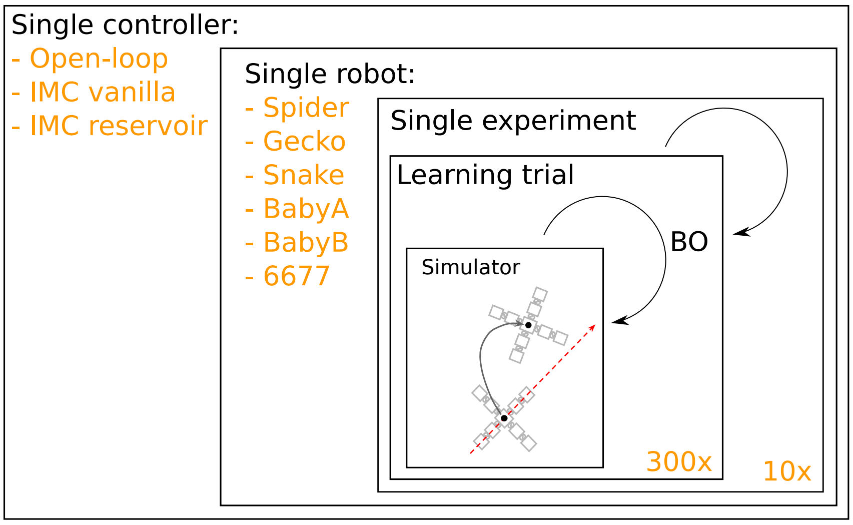


Figure 16: Schematic overview of all learning experiments that are being conducted. For 3 controllers we will learn locomotion in 6 morphologically different robots in 10 runs, with 300 learning trials per run. Each trial entails a simulation of 60 s.

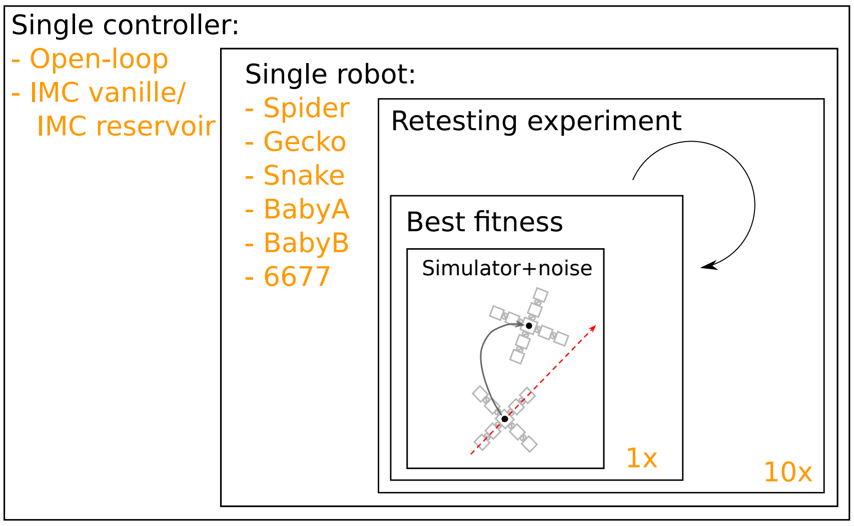


Figure 17: Schematic overview of all retesting experiments that are being conducted. From the open-loop and the best IMC controller we take the best controller per 10 runs of all 6 morphologies. We only retest once to see the effect of adding noise on the controllers' performance.