J Intell Robot Syst (2017) 86:419–445
DOI 10.1007/s10846-017-0465-1

Neuroevolution of Inverted Pendulum Control: a Comparative Study of Simulation Techniques

Christiaan J. Pretorius · Mathys C. du Plessis · John W. Gonsalves

Received: 9 September 2015 / Accepted: 5 January 2017 / Published online: 19 January 2017

Abstract The inverted pendulum control problem is a classical benchmark in control theory. Amongst the approaches to developing control programs for an inverted pendulum, the evolution of Artificial Neural Network (ANN) based controllers has received some attention. The authors have previously shown that Evolutionary Robotics (ER) can successfully be used to evolve inverted pendulum stabilization controllers in simulation and that these controllers can transfer successfully from simulation to real-world robotic hardware. During this process, use was made of robotic simulators constructed from empirically-collected data and based on ANNs. The current work aims to compare this method of simulator construction with the more traditional method of building robotic simulators based on physics equations governing the robotic system under consideration. In order to compare ANN-based and physics-based simulators in the evolution of inverted pendulum controllers, a real-world wheeled inverted pendulum robot was considered. Simulators based on ANNs as well as on a system of ordinary differential equations describing the dynamics of the robot were developed. These two simulation techniques were then compared by using each in the simulation-based evolution of controllers. During the evolution process, the effects of injecting different levels of noise into the simulation was furthermore studied. Encouraging results were obtained, with controllers evolved using ANN-based simulators and realistic levels of noise outperforming those evolved using the physics-based simulators.

Keywords Inverted pendulum · Neuroevolution · Evolutionary robotics · Artificial neural networks · Control theory · System identification

Mathematics Subject Classification (2010) 34A30 · 34A34 · 65L06 · 68T40 · 93C15 · 93C85

The financial assistance of the National Research Foundation (NRF) towards this research is hereby gratefully acknowledged (UID number: 79570). Opinions expressed and conclusions arrived at, are those of the authors and are not necessarily to be attributed to the NRF.

C. J. Pretorius (✉) · J. W. Gonsalves
Department of Mathematics and Applied Mathematics, Nelson Mandela Metropolitan University, Port Elizabeth, South Africa
e-mail: cpretorius@nmmu.ac.za

M. C. du Plessis
Department of Computing Sciences, Nelson Mandela Metropolitan University, Port Elizabeth, South Africa

1 Introduction

Evolutionary Robotics (ER) is a field that allows for the semi-automatic development of robotic controllers for autonomous robots [43]. A controller manages the interaction between a robot and its environment and allows a robot to perform a required task. During the ER process, a population of candidate controllers is

maintained and iteratively optimized using an Evolutionary Algorithm (EA) [15]. The goal of this optimization process is to produce controllers that can adequately perform a pre-defined task when uploaded to a real-world robot. During each generation of the ER process, each controller in the population is assigned a fitness value based on the performance of said controller in performing the required task [40].

To determine the fitness associated with each controller in the population, each controller can be uploaded to the real-world robot and its performance evaluated [16]. This is, however, time consuming and can damage robotic hardware. Thus, as an alternative to real-world evaluation of controller fitness, use is often made of robotic simulators to accelerate and simplify the controller evolution process [36].

Although the use of a robotic simulator during the ER process can dramatically accelerate this process, using such simulators introduces some challenges of its own. The most important challenge associated with using simulators in the ER process is known as the *reality gap* problem [25, 29, 34]. If simulators used during controller evolution do not perfectly model the operation of the real-world robot, these simulator inaccuracies can lead to controllers that perform well in simulation, but perform suboptimally or even poorly when these controllers are executed on the real-world robot. Controllers evolved when making use of inaccurate and oversimplified simulators can furthermore potentially exploit inaccuracies in the simulator, adding to the suboptimal real-world performance of these controllers.

Simulators used in the ER process are typically based on the physics governing the robotic system under consideration [17, 33]. Such physics-based simulators have seen very wide usage in the ER community and are typically constructed by making use of physics engines [13]. The usage of physics-based robotic simulators has certain challenges associated with it. Firstly, even if use is made of a physics engine in the construction of such a simulator, the researcher needs at least a rudimentary understanding of the physics involved. This can complicate the simulator development process. Physics-based simulators can additionally be computationally expensive to run [13]. This can slow down the controller evolution process. Lastly, certain simplifying assumptions have to be made during the construction of physics-based simulators, meaning that these simulators can potentially not accurately model the real-world robot under consideration. For the simplifying assumptions often made in the construction of physics-based simulators for the robot considered in the current study, the reader is referred to Section 2.3.

Owing to the challenges associated with using physics-based simulators in the ER process, the authors previously considered the development of robotic simulators for usage in ER, with these simulators not explicitly based on the physics governing the robotic system. Rather, these simulators were *constructed directly from experimental data* collected from the robot under consideration, by *using this data to train Artificial Neural Networks (ANNs)* which could be used as simulators [47]. It was shown that, using this alternative method of simulation, controllers could be evolved in simulation and successfully transferred to real-world robots.

As a result of the promising results obtained by using ANNs as simulators in the ER process, the authors subsequently compared ANN-based and physics-based simulators in the evolution of simple navigation controllers for a differentially-steered mobile robot (the Khepera III) [48]. Encouraging results were obtained, with ANN-based simulators leading to better transferability of controllers to the real world, as compared to physics-based simulators. Gomez and Miikkulainen pointed out, however, that mobile robots such as the one used in the current authors' previous study lead to controllers which are "transfer friendly" [18]. Since the robot being considered is inherently stable, that is it will remain in its current state in the absence of control inputs, controllers tend to transfer quite accurately from simulation to the real-world robot. Gomez and Miikkulainen therefore advocated the investigation of transferability of controllers in more unstable domains [18].

The current work therefore aims to extend on the comparative study previously performed by the authors [48], by considering a more complex and unstable robotic platform: a wheeled inverted pendulum robot. ANN-based and physics-based simulators for the robot were constructed and the performance of these simulators compared in the simulation-based evolution of controllers which would aim to balance (stabilize) the robot about the vertical.

The remainder of this paper is presented as follows: Section 2 summarizes previous research by the authors and other researchers which is related to the current

study. The approach proposed in this study is outlined in Section 3, followed by an explanation of the specific robotic hardware and experimental methodology used to collect data from the robot (Section 4). The processes followed to construct the different robotic simulators are explained in Section 5. These simulators were subsequently used to evolve balancing controllers for the experimental robot in simulation (Section 6). Results are presented and discussed in Section 7 and finally conclusions are drawn and recommendations for future work are made in Section 8. Appendix A provides additional mathematical information not included in the body of this paper.

2 Related Work

This section outlines work applicable to the current study. Firstly, research that has been done towards reducing the reality gap problem in ER is presented (Section 2.1), followed by a summary of attempts that have been made to construct robotic simulators from empirical data (Section 2.2). Lastly, work done to simulate and control inverted pendulums is presented (Section 2.3).

2.1 Dealing with the Reality Gap in Evolutionary Robotics

When a robotic simulator is developed, it is inevitable that such a simulator, irrelevant of the level of its fidelity, will not be able to perfectly model the operation of a real-world robot. As was mentioned in Section 1, the *reality gap* problem thus plays an important role in the ER process, especially when this process is concerned with the transfer of evolved controllers to real-world robotic platforms. Various approaches have been taken to study the effect of this phenomenon, and numerous methods have been proposed to improve transferability of robotic controllers from simulation to the real world in the presence of such a reality gap.

Jakobi was one of the first researchers to propose a potential solution to the reality gap problem [28]. In his proposed approach, use was made of *minimal simulations*, that is crude simulations that only model the most important features of the robot-environment system. Noise was injected into the simulator to mask less important features of the robot-environment system which were not modelled accurately by the simulator and to prevent evolved controllers from relying on these inaccurate portions of the simulator. Jakobi demonstrated the promise in his technique by successfully applying it to the evolution of robotic controllers for various platforms. Jakobi, Husbands and Harvey [30] furthermore showed that the injection of noise into robotic simulators can aid in the transferability of evolved controllers to real-world domains, especially when realistic levels of noise are used.

Another method which has been used in an attempt to avoid the reality gap problem, is to continuously improve the accuracy of the employed robotic simulator by making use of data relating to the functioning of the real-world robot. Bongard, Zykov and Lipson [8], for example, showed that robust robotic simulators could be constructed by comparing behaviours of a real-world robot to those predicted by a simulator, even in the case where the robot's morphology underwent dramatic changes. Hartland and Bredeche followed a related approach by employing an *anticipation module* [21] to construct a partial model of the simulated environment which allowed for adaptation of the evolved controller once it was transferred to the real-world robot. This adaptation aided in better transferability of evolved controllers to the real world. Also in a related method, Zagal and Ruiz-del-Solar introduced the *back to reality* algorithm [57], in which robotic controllers and simulators are co-evolved with differences between controller fitness obtained in reality and simulation used to improve simulators.

A technique that can also be followed to alleviate the reality gap problem is to find optimal values of parameters which are used in physics-based robotic simulators, thus improving the accuracy of said simulators and, as a result, making successful transference to the real world more likely. Physics-based simulators frequently contain parameters, such as friction coefficients and the dimensions and mass distributions of objects being simulated [32]. Determining optimal values of these parameters using some optimization scheme, as opposed to hand-tuning of parameter values, has been shown to increase the accuracy of robotic simulators [37]. The current authors found optimal parameter values in a model of a differentially-steered robot, using data collected by motion tracking [48]. Other studies involving such calibration of model parameters have also been

performed [30, 32]. Various optimization algorithms can be used in order to find optimal values of parameters in robotic models. These include Genetic Algorithms (GAs) [48] and Particle Swarm Optimization (PSO) [37]. The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) algorithm has also previously been shown to be able to accurately determine optimal parameter values in simulation models [50].

2.2 Building Simulators from Experimental Data

It is clear from Section 2.1 that a vast number of approaches to simulator construction and optimization have been explored in ER. The majority of these approaches have in common, however, that they rely on physics-based simulation. To investigate whether an alternative method of simulation might be advantageous in the ER process, the authors have previously considered constructing robotic simulators directly from data sampled from the real-world behaviour of an experimental robot [44–47]. This essentially amounts to performing a system identification process [39]. ANNs were trained using the collected data, and were employed as simulators during the controller evolution process. This method of simulation was shown to produce controllers which can adequately perform numerous real-world tasks[44–47].

There are various potential advantages to using ANNs as simulators over physics-based robotic simulators. These include the fact that ANN-based simulators will compensate for idiosyncrasies in robotic behaviour, since these idiosyncrasies will also be present in the data from which the ANNs are constructed. These idiosyncrasies might not be taken into account in a physics-based simulator as a result of simplifying assumptions made during the construction of the simulator. Additionally, since one only requires real-world data to construct ANN-based simulators, the experimenter does not need any knowledge of physics to construct ANN-based simulators. This can potentially simplify the simulator development process. ANNs have furthermore been shown to be tolerant to noise, which will inevitably be present in data gathered from a real-world robot, and to have good generalization capabilities [47].

The notion of constructing robotic simulators from empirical data has been explored by other researchers. Nakamura and Hashimoto [38], for example, employed ANNs to learn the system dynamics of a swing-up pendulum system and subsequently successfully developed controllers for their robotic platform based on these simulators and using a Reinforcement Learning (RL) process. Genetic Programming has also been employed to construct robotic simulators directly from real-world data to model the flight dynamics of a miniature rotorcraft, following which a PID-based controller was developed using the simulator [12]. For a more general review of model learning techniques, the interested reader is referred to [41].

2.3 Simulation and Control of Inverted Pendulums

The application of neuroevolution to the inverted pendulum problem has largely been investigated purely in simulation. The feasibility of successfully evolving ANN-based controllers to balance an inverted pendulum in simulation has, for example, been explored by making use of the Neuroevolution of Augmenting Topologies (NEAT) method [51], GAs [31, 47, 54] or the CMA-ES algorithm [22, 27].

Since the above-mentioned studies illustrated the capabilities of neuroevolution purely in simulation, use was made of simplified and idealized physics-based simulators in the majority of these works. Examples of these simplifications include that masses in the system were considered to be point masses and external forces were applied to the inverted pendulum in simulation without regard to how a motor on a real-world robot would exert such a force, that is the working of a robotic motor was not considered. These simplifications mean that the developed controllers would likely not transfer successfully to a real-world robot, although the transferability of developed controllers was not considered in these studies.

The lack of studies investigating the transferability of neuroevolved controllers to real-world inverted pendulum hardware was pointed out by Gomez and Miikkulainen [18]. In order to specifically investigate the transfer of evolved controllers to reality, these authors constructed a surrogate model of the real-world functioning of a double inverted pendulum system by training an ANN with data obtained by solving the differential equations describing robotic motion. Noise was subsequently added to the predictions made by this ANN-based simulator to replicate uncertainties that would be present in the operation of a real-world

robot. This ANN-based simulator was then used to evolve controllers in simulation and the transferability of these controllers to a real-world robot was investigated by using the original differential equations as a stand-in for the real world. The authors concluded that making use of adequate quantities of noise can drastically improve transferability of simulation-evolved controllers to the real world. The notion of using a simulation as a stand-in for the real world in order to investigate transferability was also employed in another study not related to an inverted pendulum robot [32].

ER, as it has been described in this paper, can also be viewed in the context of Reinforcement Learning (RL). The usage of an Evolutionary Algorithm (EA) in the RL process to develop ANN-based control for a simulated inverted pendulum has, for example, been explored by Whitley, Dominic, Das and Anderson [53], and is strongly related to the investigation carried out in the current study. One of the potential advantages of using an RL approach to develop control, is that there is no *a priori* need for an explicit model of the dynamics of the system in which said control is being developed. Such an RL approach is referred to as a model-free approach [52]. Conversely, explicit models can also be used in the RL method, in which case the method is referred to as model-based.

The two RL approaches mentioned above were compared to each other by Atkeson and Santamaria [6] on a pendulum swing-up problem. In their study, the authors found that for the system considered model-based approaches could be used in RL while making use of smaller amounts of training data than model-free approaches. Model-free approaches to RL also have the disadvantage that it can be very time-consuming to collect data through direct interaction on a real-world robot [7]. Due to the previously mentioned constraints, various authors investigating inverted pendulum problems from an RL perspective have used models of the robot being considered [7, 27, 53]. The current study similarly made use of such robotic models.

Another approach which has seen relatively extensive application to the inverted pendulum problem, is the usage of fuzzy controllers [14, 26, 35, 42, 56]. Various authors have investigated techniques which can be used to develop and tune such fuzzy controllers for inverted pendulums, such as Evolutionary Algorithms (EAs) [19, 20, 23] and other estimation algorithms [42]. Multiple studies have also explored the implementation of adaptive control schemes for inverted pendulums based on fuzzy control [14, 35]. Hapke and Komosinski [20] compared fuzzy controllers and ANN controllers for a related balancing problem. This study concluded that ANN controllers are simpler to optimize than fuzzy controllers, but that fuzzy controllers have the advantage of being interpretable by a human expert, unlike ANNs. Fuzzy control has thus been shown to be promising in the inverted pendulum problem. However, taking into account that the current study is primarily concerned with investigating different simulation methods for an inverted pendulum, the simplicity offered by ANN controllers as found by the previously-mentioned authors supported the decision to make use of ANN-based controllers.

3 Proposed Approach and Contribution of the Current Study

The authors contend that, since the ultimate goal of ER is to produce controllers which can perform tasks effectively on real-world robots, the transferability of evolved controllers to a real-world robot can only truly be investigated by making use of an actual real-world robot. Therefore, in contrast to previous studies (Section 2.3), the current study will aim to develop simulators which can predict the real-world functioning of a physical inverted pendulum robot as accurately as possible. There are various challenges involved in developing an accurate simulator for a real-world inverted pendulum robot which are not encountered when evolving controllers exclusively in simulation, or testing evolved controllers in idealized versions of the real world (Section 2.3). These challenges include:

- Idealized versions of the mathematical equations describing robotic behaviour cannot be used. Assumptions such as that masses in a robotic system can be treated as point masses will likely lead to inaccuracies in a robotic simulator.
- Parameters in the mathematical equations describing robotic behaviour cannot arbitrarily be selected or roughly approximated, as is often done when evolution is only considered in simulation. Optimal parameter values relating to the real-world robot morphology have to be determined.

– The state of the real-world robot cannot be perfectly determined at any point in time, as is the case in idealized robotic simulations. Rather, an approximation to the robotic state can be obtained from real-world robotic sensors. Such robotic sensors will inevitably contain inaccuracies in their observations.
– Sensors on a real-world robot do not return readings instantaneously and neither can commands sent to robotic motors be effected immediately. A real-world robotic system will therefore encounter time lags, which have to be taken into consideration during simulator construction and controller evolution. On the robotic platform used in the current study (Section 4), for example, the accelerometer makes use of the I2C serial communications protocol [2], while the gyroscope sensor makes use of the analog sensor interface [3]. These communications protocols introduce time lags in the readings returned by these sensors and the fact that these two sensors make use of different communications protocols, means that the two sensors could have different time lags associated with them. This was confirmed in informal tests which showed that the time lag of the accelerometer was larger than that of the gyroscope sensor.

As a result of the challenges mentioned above, the authors contend that investigating the simulation and evolution-based controller development for a real-world inverted pendulum robot could contribute to the field of ER. The authors have previously shown that robotic simulators constructed by making use of ANNs can accurately model the functioning of a real-world inverted pendulum robot and can be used to evolve controllers that successfully transfer to the real-world robot [47]. It is not, however, known how this technique of simulator construction will compare to more traditional physics-based approaches.

The current work therefore proposes to extend on the authors' previous studies by quantitatively comparing physics-based and ANN-based techniques of simulation as applied to a real-world inverted pendulum robot. The effect of the reality gap problem (Section 2.1) as associated with both these types of simulation will additionally be investigated, by injecting different levels of noise into each of the developed simulators during controller evolution and examining the effect of this noise on transferability of evolved controllers to the real world.

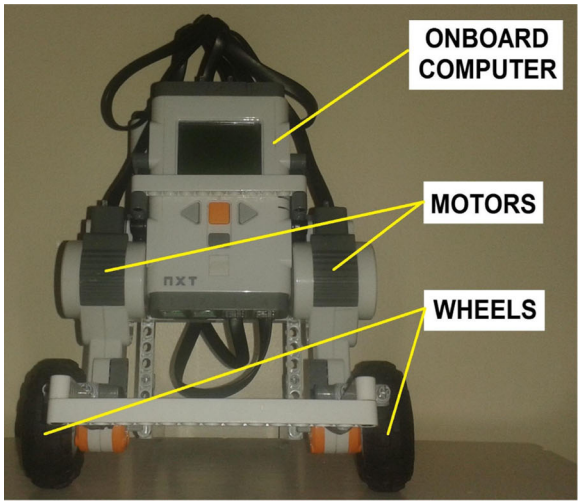
The main objectives of the current study were thus to answer the following questions:

– How accurately can ANNs simulate the operation of a wheeled inverted pendulum robot and how does this accuracy compare to that of a physics-based approach to simulating the same robot?
– How does the computational efficiency of an ANN-based simulator compare to that of a physics-based simulator for the robot?
– How well do controllers evolved in a simulation based on ANN-simulators transfer to the real-world robot, in comparison to controllers evolved in a physics-based simulation?

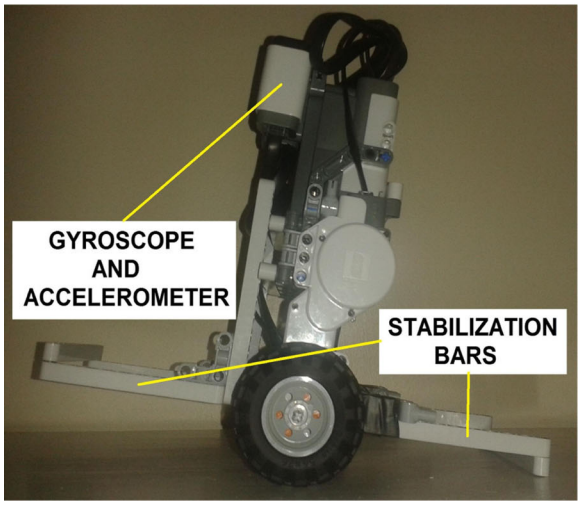
In order to answer the above questions, a physical robot was built and data collected from this robot from which robotic simulators could be constructed. Specifically, the collected data was used to optimize the values of parameters in physics-based models for the robot (as has been done by others, Section 2.1). This same data was also used to train ANN-based simulators for the robot. After constructing these different types of simulators (Section 5), the simulators were used to evolve balancing controllers for the robot in simulation (Section 6). The robot and data collection process are discussed in the next section.

4 Robotic Hardware and Data Acquisition

The study made use of a wheeled inverted pendulum robot constructed from a Lego Mindstorms NXT kit. This robot was built similarly to that used in a related study [55] and is shown in Fig. 1. As can be seen from this figure, the motion of the robot was controlled using two onboard motors, each connected to a separate wheel. To monitor the state of the robot, two sensors (only one of which is visible in Fig. 1) were also included on the robot: a gyroscope and accelerometer. The robot's onboard computer was configured to run the nxtOSEK firmware [4], which allowed for programming in C++. This firmware was chosen because a related study made use of this firmware in conjunction with the physics model which will be employed in the current study [55]. The basic physical quantities of interest in modelling the robot are shown in Fig. 2 and will be discussed in more detail in Section 5.



(a) Front view



(b) Side view

Fig. 1 Wheeled inverted pendulum robot

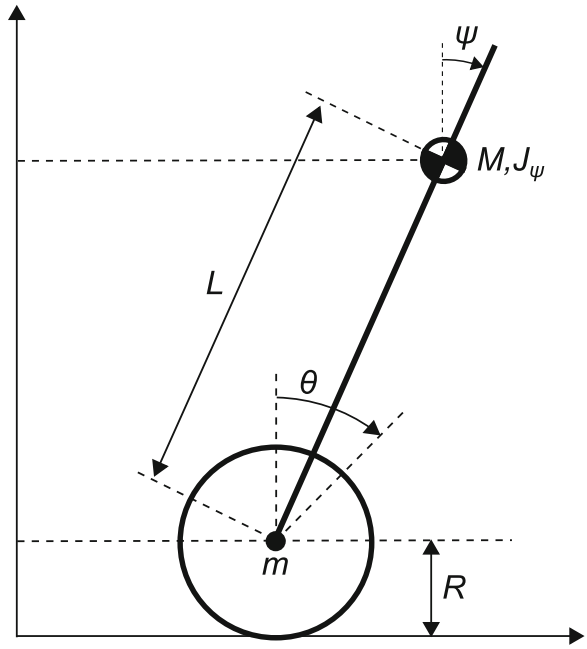


Fig. 2 Quantities involved in the robotic simulators (taken from [10])

ψ represents the angle between the robot's body and the perfect vertical, whereas θ is the angle between the vertical and an imaginary radial line fixed to either of the robot's wheels.

In order to construct simulators for the robot, data was gathered from the real-world robot. This data was used to optimize parameters in the physics-based models and to train ANN-based simulators constructed for the robot (Section 5). To acquire the needed data, the robot was placed on a horizontal surface and random motor commands were executed on the robot, while collecting data from its onboard sensors. Each motor command was maintained for a time period of 20 ms. This time period was selected in accordance with a related study [27] and was found to be long enough for the sensors on the robot to give relatively accurate readings (shorter time periods lead to very erratic movements of the robot and large levels of noise in sensor readings), but short enough to allow for adequate stabilization of the robot by the developed controllers. The stabilization bars (Fig. 1) were used to keep the robot from falling over during this data collection process. In order to simplify the considered system, it was assumed that both the robot's wheels would always move under the same control signal.

The robot's body angular velocity ($\dot{\psi}$) in response to motor commands sent to the robot was measured directly using the onboard gyroscope sensor, while its tilt angle (ψ) was determined using inputs from both the accelerometer and gyroscope sensors. Data obtained from these two sensors was filtered through a complementary filter [11] to reduce the noise coming from these sensors and produce an approximation of the tilt angle. A complementary filter operates by passing the tilt angle reading from the accelerometer through a low-pass filter, and applying a high-pass filter to the approximation of the tilt angle obtained through numerical integration of the gyroscope output. These two tilt values are then combined into a single

value by weighted averaging. Lastly, the robot's wheel angular velocity ($\dot{\theta}$) was determined by making use of the motor encoders of the robot's motors.

At the end of the data collection process, data patterns could be parsed containing the following eight quantities: u_{prev}, u_{new}, ψ_{prev}, $\dot{\theta}_{prev}$, $\dot{\psi}_{prev}$, ψ_{new}, $\dot{\theta}_{new}$ and $\dot{\psi}_{new}$. Here u_{prev} represents the PWM (Pulse Width Modulation) control signal given to the motors as part of the previous 20 ms time window and u_{new} is the PWM control signal maintained by the motors during the current 20 ms time window. ψ_{prev}, $\dot{\theta}_{prev}$ and $\dot{\psi}_{prev}$ are the body angle, wheel angular velocity and body angular velocity of the robot at the beginning of the 20 ms time window, whereas ψ_{new}, $\dot{\theta}_{new}$ and $\dot{\psi}_{new}$ represent these same quantities at the end of the 20 ms time window.

The collected data was separated into two distinct sets: a training set and validation set. The training set was used to develop both physics-based and ANN-based simulators for the motion of the experimental robot in response to motor commands issued to the robot, whereas the validation set was employed to test the accuracy and generalization capabilities of these developed simulators. The training set contained 3070 training patterns and the validation set contained 479 patterns. The development of these simulators is discussed in the following section.

5 Simulator Development

Simulators were developed for the robot, based on both a physics approach and ANN approach. Each of the simulators constructed in this work would be given as inputs u_{prev}, u_{new}, ψ_{prev}, $\dot{\theta}_{prev}$ and $\dot{\psi}_{prev}$, and from these quantities would be expected to predict ψ_{new}, $\dot{\theta}_{new}$ and $\dot{\psi}_{new}$. Thus, given the state of the robot at the beginning of a certain 20 ms time period and the motor command issued to the robot during this time period, the simulators would be expected to predict the state of the robot at the end of the time period. Additionally, it was anticipated that the motor command maintained during the previous time period could also factor into the calculation, and it was therefore presented as an additional input. The development of the different simulators for the robot is discussed in the following sections. The variables and parameters used in these simulators are summarized in Table 1 (adapted from [10]).

5.1 Physics-Based Simulation

By making use of Lagrangian mechanics, the equations of motion describing the wheeled inverted pendulum under consideration can be shown to be [10]:

$$\begin{aligned} T_\theta(t) \;=\; & ((2m+M)R^2 + 2n^2 J_m)\ddot{\theta}(t) \\ & - MLR\dot{\psi}^2(t)\sin\psi(t) \\ & + (MLR\cos\psi(t) - 2n^2 J_m)\ddot{\psi}(t) \end{aligned} \tag{1}$$

$$\begin{aligned} T_\psi(t) \;=\; & (MLR\cos\psi(t) - 2n^2 J_m)\ddot{\theta}(t) \\ & - MgL\sin\psi(t) \\ & + (ML^2 + J_\psi + 2n^2 J_m)\ddot{\psi}(t) \end{aligned} \tag{2}$$

The dependent variables in the above system of ordinary differential equations, θ and ψ, have physical interpretations as shown in Fig. 2 and discussed in Section 4. Although these quantities depend on time (for example, $\theta = \theta(t)$), time dependence will not be written explicitly in the body of this paper for these or any other time-dependent quantities, for sake of notational brevity. T_θ and T_ψ represent torques acting on the robot's wheels and body, respectively, and can be expressed in terms of the voltage, v, applied to the robot's motors as follows [10]:

$$T_\theta(t) = 2\alpha v(t) - 2(\beta + f_w)\dot{\theta}(t) + 2\beta\dot{\psi}(t) \tag{3}$$

$$T_\psi(t) = -2\alpha v(t) + 2\beta\dot{\theta}(t) - 2\beta\dot{\psi}(t) \tag{4}$$

where

$$\alpha = \frac{nK_t}{R_m} \text{ and } \beta = \frac{nK_tK_b}{R_m} + f_m. \tag{5}$$

When sending motor commands to the robot, these commands were sent in the form of a PWM input in the range $[-100, 100]$. The PWM signal applied at any point in time at either of the robot's motors, u, was assumed to be related to the corresponding motor voltage, v, by:

$$v(t) = \frac{V_{max}\,D(u(t-t_a), d)}{100G} \tag{6}$$

The above equation was taken from Yamamoto [55] with an adjustment for actuator delay incorporated from Canale and Brunet [10]. V_{max} is the maximum DC motor voltage. Furthermore, $D(u(t-t_a), d)$ is a dead zone function applied to $u(t-t_a)$ with lower limit $-d$ and upper limit d. This dead zone function models internal friction in the robot's motors. t_a represents the actuator delay, namely the amount of time taken for a motor command to reach the robot's motors

Table 1 Variables and parameters used in the robotic simulators

Variable/Parameter	Description
θ	Wheel angle
ψ	Body angle
$\dot{\theta}$	Wheel angular velocity
$\dot{\psi}$	Body angular velocity
u_{prev}	PWM signal maintained during previous time period
u_{new}	PWM signal maintained during current time period
ψ_{prev}	Body angle at beginning of current time period
$\dot{\theta}_{prev}$	Wheel angular velocity at beginning of current time period
$\dot{\psi}_{prev}$	Body angular velocity at beginning of current time period
ψ_{new}	Body angle at end of current time period
$\dot{\theta}_{new}$	Wheel angular velocity at end of current time period
$\dot{\psi}_{new}$	Body angular velocity at end of current time period
T_θ	Torque applied to robot's wheels
T_ψ	Torque applied to robot's body
v	Voltage applied to robot's motors
u	PWM signal applied to robot's motors
V_{max}	Maximum DC motor voltage
t_a	Actuator delay
g	Gravitational acceleration
m	Wheel mass
R	Wheel radius
M	Body mass
L	Wheel axle to center of mass distance
J_ψ	Body pitch inertia moment
J_m	DC motor inertia moment
R_m	DC motor resistance
K_b	DC motor back EMF constant
K_t	DC motor torque constant
f_m	DC motor friction coefficient
n	Gearbox ratio
f_w	Friction coefficient between wheel and floor
G	PWM gain factor

once it has been issued. The different sensors used on the robot had different time delays associated with them (Section 3). Direct incorporation of these delays into the physics-based models would therefore be a complex process. As such, these sensor delays were not directly incorporated into the physics-based models. However, it was anticipated that the actuator delay parameter t_a could compensate for the actual actuator delay, as well as partially compensate for the sensor delays. It was thus decided to lump all delays (actuator and sensor delays) into the t_a parameter.

The remaining quantities shown in Eqs. 1 to 6 are parameters related to the physical configuration of the robot and the working of its motors. The meaning of each parameter is given in Table 1.

Assuming that the robot remains close to the vertical position during the execution of a successful balancing controller, that is $\psi \approx 0$, a small angle approximation can be used to linearize the differential equations given in Eqs. 1 and 2. This linearization process is of interest, since linearizing the differential equations means that an analytical solution can be

found to these equations. This, in turn, means that predictions from the linearized equations can be obtained with much less computational effort (computer processing time), at the possible expense of accuracy. In the limit $\psi \rightarrow 0$, it can be assumed that $\sin\psi \rightarrow \psi$ and $\cos\psi \rightarrow 1$ [55]. Additionally, higher order terms such as $\dot{\psi}^2$ can be ignored, since they will be negligibly small [55]. Under these assumptions, the linear version of Eqs. 1 and 2 becomes:

$$T_\theta(t) = ((2m+M)R^2 + 2n^2 J_m)\ddot{\theta}(t) + (MLR - 2n^2 J_m)\ddot{\psi}(t) \quad (7)$$

$$T_\psi(t) = (MLR - 2n^2 J_m)\ddot{\theta}(t) - MgL\psi(t) + (ML^2 + J_\psi + 2n^2 J_m)\ddot{\psi}(t) \quad (8)$$

Each of the coupled systems of second-order non-linear or linear differential equations (Eqs. 1 and 2, as well as Eqs. 7 and 8, respectively) can be rewritten as systems consisting of four first-order differential equations. By solving these systems, the state of the robot can thus be determined numerically in the variables θ, ψ, $\dot{\theta}$ and $\dot{\psi}$. Even though θ mathematically forms part of the state description of the robot at any point in time, knowledge of this angle has no bearing on the controllers developed for the robot in the current study, since these controllers took as inputs (amongst other inputs) ψ, $\dot{\theta}$ and $\dot{\psi}$ (Section 6). Careful examination of Eqs. 1 and 2, as well as Eqs. 7 and 8 also reveals that the predictions made by these models for the quantities ψ, $\dot{\theta}$ and $\dot{\psi}$ (excluding θ) are independent of the initial condition imposed on θ. Therefore, prediction of θ was not required from the physics-based models. It is for this reason that θ was not collected during the data acquisition process (Section 4).

The non-linear system of differential Eqs. 1 and 2 was solved numerically using the fourth-order Runge-Kutta algorithm [9] with a time-step of 0.001 seconds. Authors who have previously investigated the inverted pendulum problem used a similar numerical scheme [51]. For the implementation of this numerical scheme, use was made of the open-source Apache Commons Mathematics Library [5]. As mentioned previously, the linear system of differential Eqs. 7 and 8 was solved analytically. A brief discussion on this analytical solution is given in Appendix A.

Clearly, the predictions made by the linear or non-linear physics models would be affected by the values of parameters used in the corresponding equations. Optimal values of these parameters were therefore determined separately for both the linear and non-linear models, based on the data obtained from the real-world robot (Section 4). Parameters were optimized by using a GA [24] to minimize the mean-squared error (MSE) obtained when using a specific combination of parameter values. It was decided to employ a GA as opposed to a gradient-based optimization algorithm, since the acquired data used to optimize parameter values was found to contain noise (resulting from inaccuracies in robotic sensors, amongst other sources), which could lead to local optima in the search space in which gradient-based algorithms could potentially be trapped. The MSE (ε) to be minimized by the GA was defined as:

$$\varepsilon = \frac{\sum_{i=1}^{N}\left[(\psi_{i,e}-\psi_{i,p})^2+(\dot{\theta}_{i,e}-\dot{\theta}_{i,p})^2+(\dot{\psi}_{i,e}-\dot{\psi}_{i,p})^2\right]}{N} \quad (9)$$

In Eq. 9, $\psi_{i,e}$, $\dot{\theta}_{i,e}$ and $\dot{\psi}_{i,e}$ are the expected values of each of the indicated quantities from the real-world data collected from the robot, corresponding to training pattern i. The quantities $\psi_{i,p}$, $\dot{\theta}_{i,p}$ and $\dot{\psi}_{i,p}$ are the corresponding values predicted by each of the physics models when using a specific combination of parameters. The number of patterns in the training set was $N = 3070$. As mentioned before (Section 4), only data in the training set was presented during this parameter optimization process. Data in the validation set was used after parameter optimization to check the accuracy of the optimized physics models. By minimizing the MSE, a model could thus be found for which the predictions matched those expected from the real world data as closely as possible. The quantities in above equation were scaled so that each of the three terms contributed roughly equally to the total MSE, to avoid dominance of one term during the optimization process. Parameter values used in the GA for this optimization process are given in Table 2. During the optimization process, the validation data (Section 4) was used to monitor for overfitting. The GA was stopped if overfitting was detected or, otherwise, after a fixed number of generations of the GA.

Each chromosome in the GA population directly encoded the parameter values which were to be optimized as real-numbers. This meant that all chromosomes in

Table 2 GA parameters used for physics model optimization

Parameter	Value
Population size	150
Selection operator	Tournament selection
Tournament size	25
Crossover probability	80 %
Crossover operator	Simulated Binary Crossover
Mutation probability	5 %
Mutation operator	Random component perturbation from uniform distribution with zero mean

the population had the same length. Mutation was performed on a certain component of a given chromosome by adding a random number sampled from a uniform distribution with zero mean to that component. The ranges from which these random numbers were generated were chosen to be different for each component of the chromosome, since the orders of magnitude of the parameters to be optimized varied widely (Table 3). These ranges were chosen to have a magnitude of roughly 10 % of the initial estimate for each parameter. Crossover of chromosomes was performed using the Simulated Binary Crossover technique [15], which is an extension of single-point crossover from binary encoded chromosomes to chromosomes encoding real-numbers. In this crossover process, each component of an offspring chromosome is formed as a linear combination of the corresponding components from each of the two parent chromosomes. Table 2 shows that a large tournament size was used, when related to the population size. It was found through informal testing that this tournament size accelerated the optimization process without sacrificing the final accuracy of the optimized physics models.

Initial estimates for the values of parameters in the physics models were taken from the literature [10, 55] or were determined through physical measurements. These initial estimates, along with the intervals in which the search for optimal parameter values was conducted, are given in Table 3 for each parameter considered. It should be noted that some of the information given in Table 3 will only be discussed in a later section. Also, not all parameters in the physics models are present in Table 3. This follows from the fact that some parameters could be optimized directly, without needing to also optimize parameter values used to calculate them. See, for example, Eq. 5: α could be optimized directly, without needing to also optimize parameter values on which α depends (for example R_m). Additionally, quantities which could be determined directly, such as masses (determined by weighing) and the value of g, which is assumed to be constant, were not optimized. For the sake of brevity, numerical values for parameters which were not optimized, will not be given. Numerical values in Table 3 are all given in SI units. These units are omitted for brevity.

5.2 Neural Network Simulators

As an alternative to the physics-based approach to simulating the inverted pendulum robot described in Section 5.1, the usage of ANNs as simulators was

Table 3 Optimal parameter values for the physics models

Parameter	Initial estimate	Interval searched	Optimal value found for linear physics model	Optimal value found for non-linear physics model
J_m	10^{-5}	$[10^{-6}, 10^{-2}]$	0.0027	0.0022
f_w	0	[0, 0.3]	0.0126	0.0120
G	1	[0.8, 1.2]	0.8001	0.8222
d	7	[5, 9]	5.0000	5.0005
L	0.072	[0.06, 0.084]	0.0840	0.0799
J_ψ	0.001	[0.0001, 0.003]	0.0025	0.0027
α	0.047	[0.02, 0.07]	0.0700	0.0586
β	0.024	[0.01, 0.05]	0.0472	0.0372
t_a	0.004	[0, 0.019]	0.0131	0.0131

investigated. These ANN-based simulators will be referred to as Simulator Neural Networks (SNNs). These SNNs were constructed based on the same data as was used to find optimal values of parameters in the physics simulators (Section 5.1). In contrast to these physics-based simulators, however, the SNNs were constructed based solely on this data. No explicit derivation of any mathematical equations was required during the construction of the SNNs. The predictive function to be performed by the SNNs was identical to the physics-based simulators, that is, a mapping was sought of the form:

$$\{u_{prev}, u_{new}, \psi_{prev}, \dot{\theta}_{prev}, \dot{\psi}_{prev}\} \rightarrow \{\psi_{new}, \dot{\theta}_{new}, \dot{\psi}_{new}\} \quad (10)$$

It was shown by the authors in a previous work [47] that splitting the simulation task into specialized SNNs, one for the prediction of each of the quantities ψ_{new}, $\dot{\theta}_{new}$ and $\dot{\psi}_{new}$, leads to more accurate SNNs than training one SNN for the prediction of all three quantities. This method was therefore used in the current work.

Various SNN topologies were investigated to compare the simulation capabilities of each of these different topologies. This study considered simple Feed-Forward Neural Networks (FFNNs) [15], Elman Recurrent Neural Networks (ERNNs) [15] and also investigated SNNs evolved by making use of the Neuroevolution of Augmenting Topologies (NEAT) method [51]. ERNNs were investigated since it was anticipated that the recurrent nature of these Neural Networks could potentially simulate the robotic system more accurately than the FFNNs by making use of the temporal nature of the problem being solved. The FFNNs and ERNNs investigated in this work were each given a single hidden layer, and the effect of varying the number of neurons in each network's hidden layer was investigated. Each FFNN and ERNN investigated in this work therefore had a fixed topology and the weight parameters of each network was simply optimized by making use of the training data. In contrast, the NEAT algorithm optimizes both the topology and weight parameters of an ANN simultaneously. Therefore, the possible advantage to using the NEAT algorithm is that a fixed network topology does not have to be decided upon before network training can begin.

In addition to investigating different network topologies, this study furthermore explored the effect of not presenting one of the inputs (10) to the SNNs during SNN training. Previous studies have investigated the effect of evolving ANN controllers for inverted pendulums, with some inputs related to the state of the robot not presented to the controller during controller evolution [51]. The effect of similarly removing inputs from SNNs has not yet been explored. Not presenting a certain input to the SNNs during training and noting the effect of this on the accuracy of trained SNNs would give an indication of the sensitivity of the SNNs to each of the inputs shown in Eq. 10. If it was found that not presenting a certain input to the SNNs did not affect the accuracy of these SNNs, this could possibly mean that a balancing controller could be evolved in simulation and transferred to a real-world robot making use of less sensors than was used in the current work, although this was not tested. An investigation into the accuracy of SNNs presented with reduced inputs therefore appeared warranted. In order to test this, one of the five inputs (10) was not presented to the SNNs during training. This was done for each of the five inputs.

The neurons in all SNNs investigated in this study were implemented as summation units and used hyperbolic tangent activation functions, with inputs and outputs in the training data appropriately scaled. The different SNN topologies investigated in this study were trained as follows. FFNNs were trained using the resilient backpropagation training algorithm [49], whereas a GA [24] was used to train the ERNNs. In this GA, each weight associated with a certain ERNN was encoded as a real-number. Crossover was performed by making use of a splice operator (see [1]) and mutation was performed by adding a random number sampled from a uniform distribution with zero mean to a certain weight. The NEAT-based SNNs were evolved using the NEAT algorithm provided in the Encog Machine Learning Framework [1], and this package was also used to implement the training of the FFNNs and ERNNs. The parameters used during these training processes are briefly shown in Table 4. These values are mostly default values used by the Encog Machine Learning Framework [1], although, in some cases, other values were chosen to improve training accuracy or speed. An exponentially decaying mutation probability was used in the case of the NEAT networks, as it was found to accelerate the training process. The interested reader is referred to the documentation of the Encog Machine Learning

Table 4 Parameter values used in SNN training

Network type	Parameter	Value
FFNN	Initial step size	0.1
	Minimum step size	10^{-7}
	Maximum step size	50
	η^+ (see [49])	1.2
	η^- (see [49])	0.5
ERNN	Population size	150
	Crossover probability	80 %
	Crossover operator	Splice operator (see [1])
	Mutation probability	5 %
	Mutation operator	Random component perturbation from uniform distribution with zero mean
NEAT	Population size	150
	Selection operator	Truncation selection (see [1])
	Population pool size used in selection	45
	Crossover probability	50 %
	Mutation probability	40 % $* e^{-0.001k}$ (k = generation number)
	Probability of adding a node	5 %
	Probability of adding a link	15 %
	Probability of removing a link	15 %

Framework [1] for details of additional parameters not given in Table 4. The training algorithm applied to each SNN aimed to minimize the mean-squared error (MSE) of said SNN based on data from the training set (Section 4). To ensure that overfitting did not occur during SNN training, the MSE of each SNN was also calculated based on data in the validation set, and this validation error was monitored during SNN training. Each training algorithm was stopped after a predefined number of iterations, or when it was found that overfitting had occurred.

6 Controller Evolution

After developing the SNNs and physics-based simulators for the operation of the real-world robot, these simulators were used in the simulation-based evolution of stabilization controllers for the robot. During this process these simulators were compared to one another based on the performance obtained from controllers evolved using each of the simulators, when these controllers were transferred to the real-world robot. The stabilization controllers to be evolved were implemented as Elman Recurrent Neural Networks (ERNNs). Each controller took as input the state of the robot at the beginning of a certain 20 ms time period (as predicted by the relevant simulator), and from this input was expected to produce a PWM control value to be maintained by the robot's two motors for the 20 ms time period in question (u_{new}), in order to keep the robot stable in its upright position (prevent the robot from falling over). Each controller took four inputs: ψ_{prev}, $\dot{\psi}_{prev}$ and $\dot{\theta}_{prev}$ at the beginning of the 20 ms time period, as well as the previously predicted PWM control value (u_{prev}). Although u_{prev} did not describe the state of the robot, it was given as an additional input since it was anticipated that the controller could possibly be aided by explicitly being provided with its own previous output. The controller topology implemented in this work is much simpler than that used by the authors in a previous study [47].

Each controller ANN consisted of five hidden neurons and therefore also had five neurons in the context layer. All neurons in the controller were implemented as summation units and implemented hyperbolic tangent activation functions. In order to develop controllers, use was made of a GA to evolve a population of

candidate ANN controllers (the weights of each controller were evolved). The parameter values used in this GA are identical to those given in Table 2 and discussed in Section 5.1. The fitness function used to evaluate the performance of a candidate controller in the GA population was chosen as:

$$F = \sum_{i=1}^{10} \left[k_1 \sum_{j=1}^{250} \left(\psi_{i,j}\right)^2 + k_2 \sum_{j=1}^{250} \left(\dot{\psi}_{i,j}\right)^2 + k_3 \left| \sum_{j=1}^{250} \left(u_{i,j} \left|u_{i,j}\right|\right) \right| \right] \quad (11)$$

In Eq. 11, $i = 1...10$ represents 10 individual test runs that were performed in simulation. At the beginning of each test run, the robot was placed perfectly vertically in simulation with a zero angular velocity. The candidate controller being evaluated was then used to attempt to balance the robot for a period of 250 fires of the ANN controller corresponding to a real-world period of 5 seconds. For each 20 ms period during these 5 seconds, $\psi_{i,j}$, $\dot{\psi}_{i,j}$ and $u_{i,j}$ represent the robot's body tilt angle, the robot's body angular velocity and the PWM value maintained by the robot (that predicted by the ANN controller) for the i^{th} run's j^{th} time period. In this equation, k_1, k_2 and k_3 represent negative constants.

The fitness value assigned to a candidate controller in the population would therefore be large if each of the three summation terms shown in Eq. 11 were as small as possible. The fitness function therefore aimed to minimize the deviation from perfect vertical (first term), the angular velocity of the robot (second term) and the distance by which the robot moved forwards or backwards over the working surface during the execution of the controller (third term). The values of k_1, k_2 and k_3 acted as weights for the relative importance of each of these three terms. It was found through informal experimentation that the values of k_1 and k_2 were more important than that of k_3 in evolving controllers which could keep the robot balanced successfully. The numerical values of these three constants were therefore selected such that the k_1 and k_2 terms contributed roughly equally to the fitness of a given controller, with less emphasis given to the k_3 term. These constants were chosen to have numerical values as follows: $k_1 = -1.1\ rad^{-2}$, $k_2 = -0.001\ s^2.rad^{-2}$ and $k_3 = -0.0006$ (k_3 is unitless since the PWM motor value has no specific unit).

In order to aid in the successful transference of evolved controllers from simulation to the real-world robot, noise (random perturbations) was injected into the predictions (added to the outputs) coming from each of the simulators during the controller evolution process. This noise was used to make evolved controllers more robust to the unpredictability in readings returned from sensors on the real-world robot. The noise furthermore could potentially mask inaccuracies in the robotic simulators, and in this way aid in increasing the transferability of evolved controllers from simulation to the real-world robot. Such an injection of noise into a simulator during the ER process has been advocated by other researchers [30]. Various levels of noise were injected into each of the simulators, in order to investigate the effect that this would have on the real-world performance of evolved controllers. Noise was modeled by assuming that this noise could be sampled from a Gaussian distribution. The standard deviation for this distribution was determined based on differences observed between validation data (Section 4) and corresponding predictions made by each of the simulators. Since the two physics-based models were found to make predictions which were almost identical in accuracy (Section 7.1), only one noise model was constructed for both these models. Four levels of noise were tested: low noise, SNN noise, physics noise and high noise. The SNN noise and physics noise models were generated as explained above. The low noise model was based on half the standard deviation of the SNN noise model, whereas the high noise model was based on double the standard deviation of the physics models.

7 Results

The following sections discuss the results obtained in terms of the accuracy and computational efficiency of the developed simulators (Sections 7.1 and 7.2), as well as the performance of evolved controllers on the real-world robot (Section 7.3).

7.1 Simulator Training Accuracy

After the training process of the SNNs and the parameter optimization of the physics-based models were completed (Section 5), the accuracy of each of these simulators was analysed. Figures 3, 4 and 5 give, for

each of the developed simulators, the mean-squared error (MSE) of the simulator after training or optimization. The given values represent the average of MSE values obtained from 30 training runs for each simulator, as well as the standard deviation in each case (denoted by *SD* in each graph). For the physics-based simulators, corresponding MSE values are also given for the simulators before the optimization process of these models' parameter values. To allow for a direct comparison between the physics models and the SNNs, the MSE values shown in Figs. 3 to 5 for the physics models are not based on Eq. 9, but were calculated for each of the quantities of interest (ψ_{new}, $\dot{\theta}_{new}$ and $\dot{\psi}_{new}$) individually. Values corresponding to each of these quantities in Figs. 3 to 5 are given in arbitrary units and have been normalized so that the maximum average MSE for each quantity is one (the unoptimized physics models were not taken into account in this normalization process). Results are presented for both physics models, as well as for the different SNN topologies with variations with one of the five inputs to the SNNs not presented to the SNNs during simulator training (Section 5.2). The MSE values were calculated based on data in the validation set captured during data collection, since said data was not presented to the simulators during training and could thus be used as an accurate measure of simulator generalization ability. It should be noted that some of the quantities shown in these figures were purposefully truncated at the maximum vertical axis (indicated by *trunc* in the figures). It was decided to do so, since including these values in the plots caused the scaling to be such that visual comparison between high-accuracy simulators became impossible (these MSE values became visually indistinguishable). In the cases where truncation

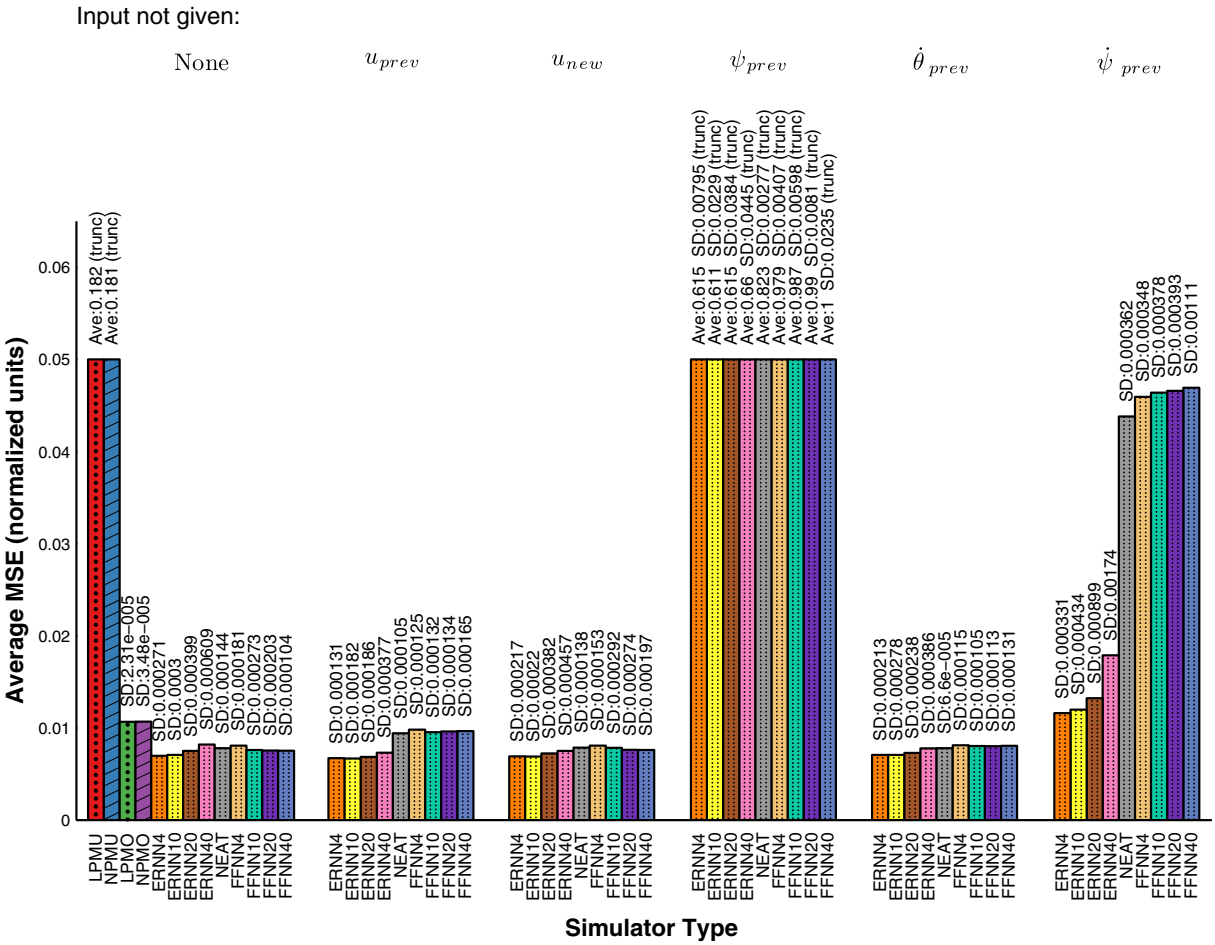


Fig. 3 Training accuracy of simulators in predicting ψ_{new}

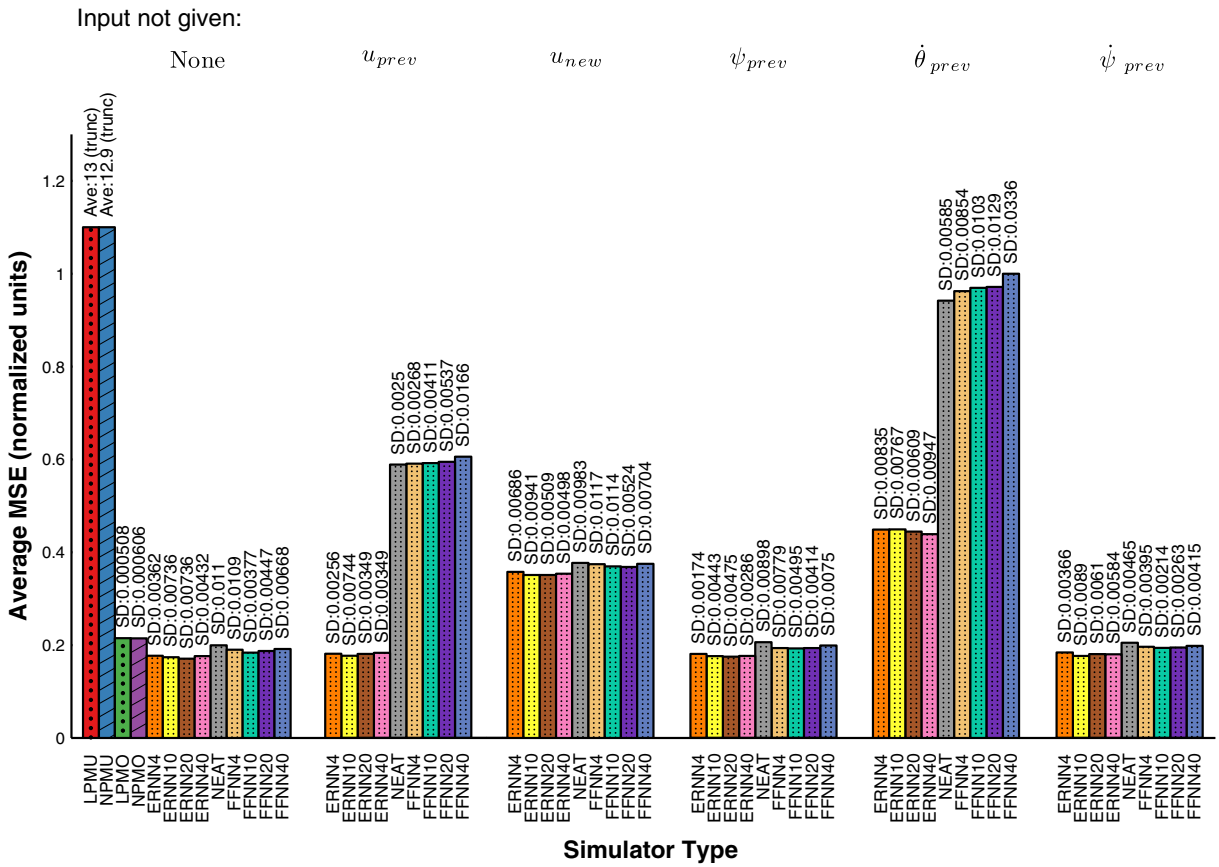


Fig. 4 Training accuracy of simulators in predicting $\dot{\theta}_{new}$

was used, the average MSE is shown numerically above the relevant bar in the figures (as *Ave*). In Figs. 3 to 5 the abbreviations used are as follows: LPMU - linear physics model unoptimized, NPMU - non-linear physics model unoptimized , LPMO - linear physics model optimized, NPMO - non-linear physics model optimized. A number appended to the type of simulator on the horizontal axis, represents the number of hidden neurons used (when applicable). Since the LPMU and NPMU values shown in Figs. 3 to 5 were based on the physics models using fixed initial (unoptimized) estimates of the parameters, the MSE values associated with these entries were always the same (30 training runs were not performed in this case). Thus the average values for LPMU and NPMU shown in these figures represent this constant MSE value for each unoptimized physics model and there is no standard deviation associated with these entries. Three patterns are used in Figs. 3 to 5, to distinguish between the linear physics models, non-linear physics models and SNNs.

The major finding from Figs. 3 to 5 is that the majority of the SNNs could make predictions more accurately than the physics models (evident from the smaller average MSE values of the SNNs). This is likely a result of the fact that the SNNs were not based on any *a priori* assumptions about the physics underlying the operation of the real-world robot. As such, these SNNs could automatically account for any idiosyncrasies in the data during the SNN training process. Since the physics-based models, conversely, were based on such assumptions, any idiosyncrasies in the data resulting from noisy operation of the robot's sensors, time lags in these sensors and so on, could potentially have contributed to the lower accuracies of these simulators. These physics-based simulators could potentially have made more accurate predictions if time lags and other idiosyncrasies were incorporated

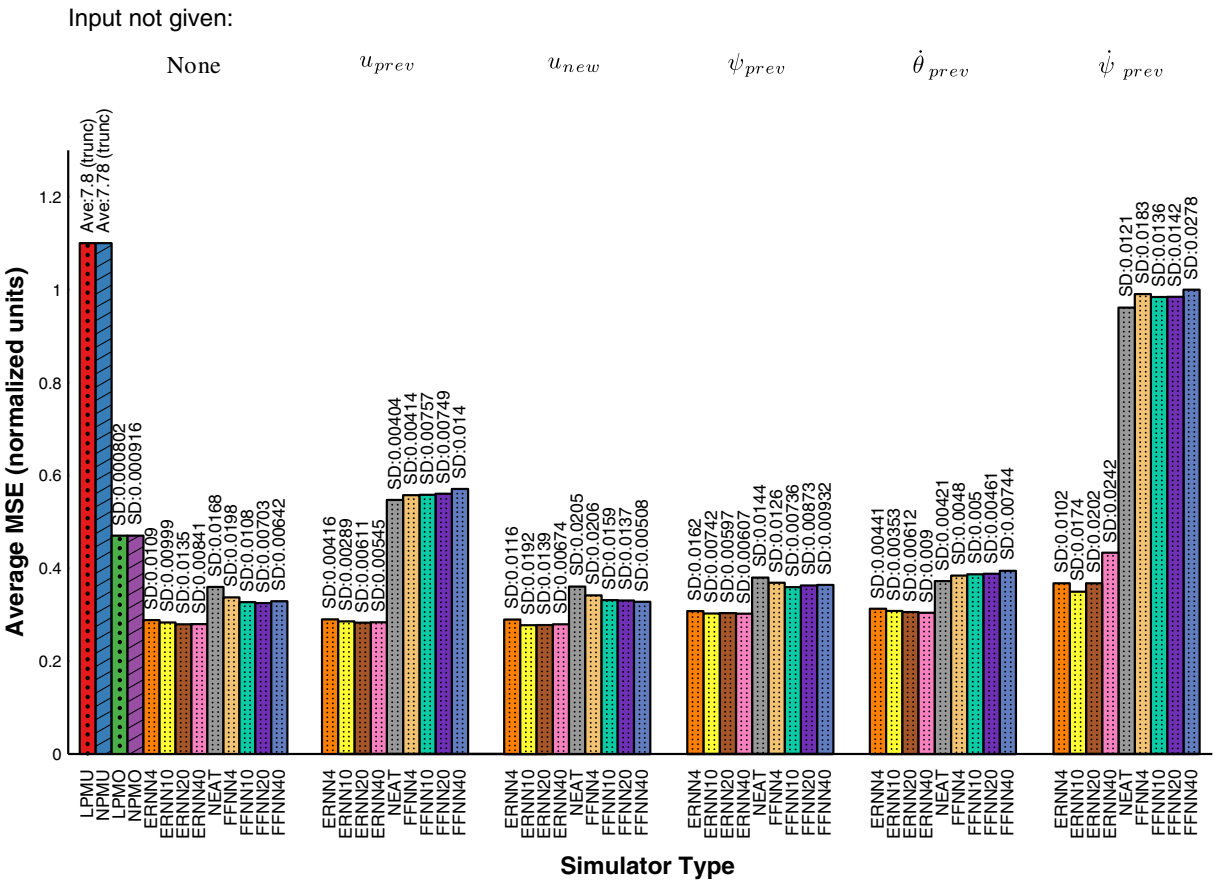


Fig. 5 Training accuracy of simulators in predicting $\dot{\psi}_{new}$

in more detail in these models, but such incorporation was not needed in the case of the SNNs.

The accuracies of the optimized linear and non-linear physics models were very close to each other for each of the three quantities being predicted. This is likely a result of the training and validation data being collected in a relatively small angle range around the vertical (Section 4), meaning that the small angle approximation used to arrive at the linear physics model (Section 5.1) was valid, resulting in predictions made by both physics models being similar. It is clear from Figs. 3 to 5 that the optimization process applied to the parameters of the physics models drastically improved the accuracy of these models, evident from the large accuracy differences between the optimized and unoptimized physics models.

The Elman Recurrent Neural Networks (ERNNs) outperformed the Feed-Forward Neural Networks (FFNNs) in almost all cases, especially in the cases where one of the inputs was not presented. This indicates that these ERNNs successfully made use of the temporal memory provided by their context layers to aid in improving the accuracy of their predictions. It should, however, be noted that the FFNNs also outperformed the accuracy of both the physics-based models in the case where all inputs were provided to the networks. Varying the number of hidden neurons in each SNN was not found to have a marked influence on the accuracy of the trained SNNs.

The NEAT networks did not train very accurately, with the accuracy of these networks being roughly on par with the FFNNs, and slightly better than the FFNNs in some circumstances. The fact that the NEAT networks could have evolved recurrent connections similar to the ERNNs, but that the NEAT networks were considerably less accurate than the ERNNs after training, suggests that these networks did not train as well as they potentially could have. Further

investigation into different parameter values to use for the evolution of NEAT networks could improve on this accuracy, but was not attempted in this study.

Interesting to note is the relative insensitivity of the SNNs to not being presented with either u_{prev} or u_{new}, respectively. This is true especially in the case of the ERNNs predicting ψ_{new} and $\dot{\psi}_{new}$. Since the context layer of these ERNNs can potentially memorize the previous PWM value given (u_{prev}), this finding suggests that ψ_{new} and $\dot{\psi}_{new}$ are potentially affected more by u_{prev} than u_{new}, as a result of the inertia of the robot. It was found that not presenting an input relating to the previous state of the robot (ψ_{prev}, $\dot{\theta}_{prev}$ or $\dot{\psi}_{prev}$) to the SNNs generally degraded their accuracy. The mathematical formulation of the physics-based models dictates that all three these quantities at a given point in time should be known to be able to predict any one of these quantities at a later point in time, and this finding is thus to be expected. It should be noted, however, that ψ_{new} and $\dot{\psi}_{new}$ can be predicted quite accurately (more accurately than predictions made by the physics-based simulators) by ERNNs without taking $\dot{\theta}_{prev}$ into account. Although this was not explicitly tested in this study, this suggests

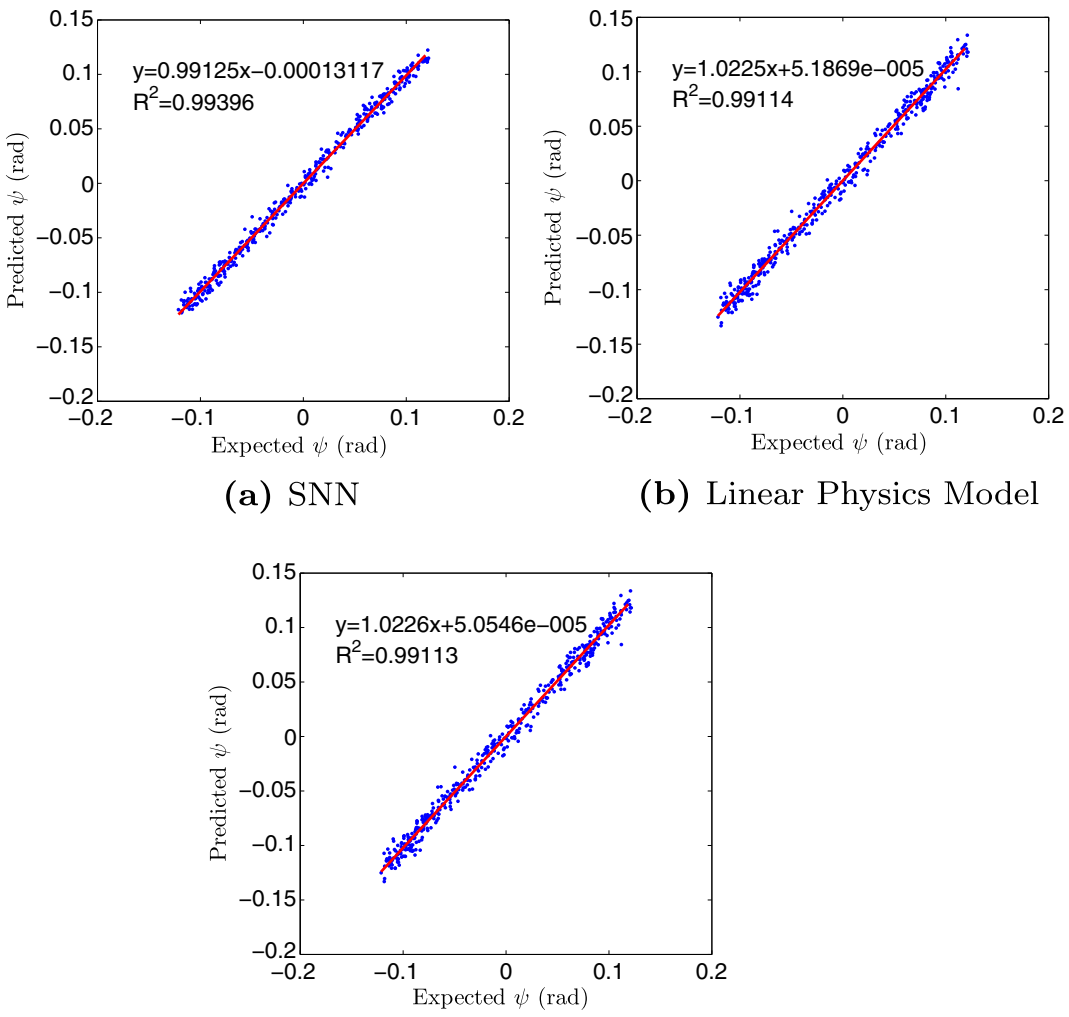


Fig. 6 Predicted vs expected values of ψ_{new}

that SNNs could potentially be used to evolve controllers without these SNNs needing to take $\dot{\theta}_{prev}$ into account.

Based on the data shown in Figs. 3 to 5, the most accurate simulator of each type (SNN, linear and non-linear physics models) was selected to be used in the evolution of balancing controllers for the robot (Section 6). The selected SNNs were a ERNN10 network (without u_{prev} as input) for the prediction of ψ_{new}, a ERNN20 network (with all inputs given) for the prediction of $\dot{\theta}_{new}$ and a ERNN10 network (without u_{new} as input) for the prediction of $\dot{\psi}_{new}$.

Table 3 shows the optimized values of the parameters for the most accurate linear and non-linear physics models. Although the majority of parameter values after optimization are relatively close to their initial estimates, some parameter values underwent large changes during the optimization process. These large changes could be due to inaccuracies in the initial estimates of the parameter values, or these changes could be attributed to the optimization process compensating for noise present in the data which was used to optimize the physics models, time lags in the sensors used to gather this data or any inaccuracies in the mathematical formulation of the physics models. It is, for example, unlikely that a true value of t_a (the actuator delay) would be 13 ms, if a value of only 4 ms is given in the literature. It should, however, be taken

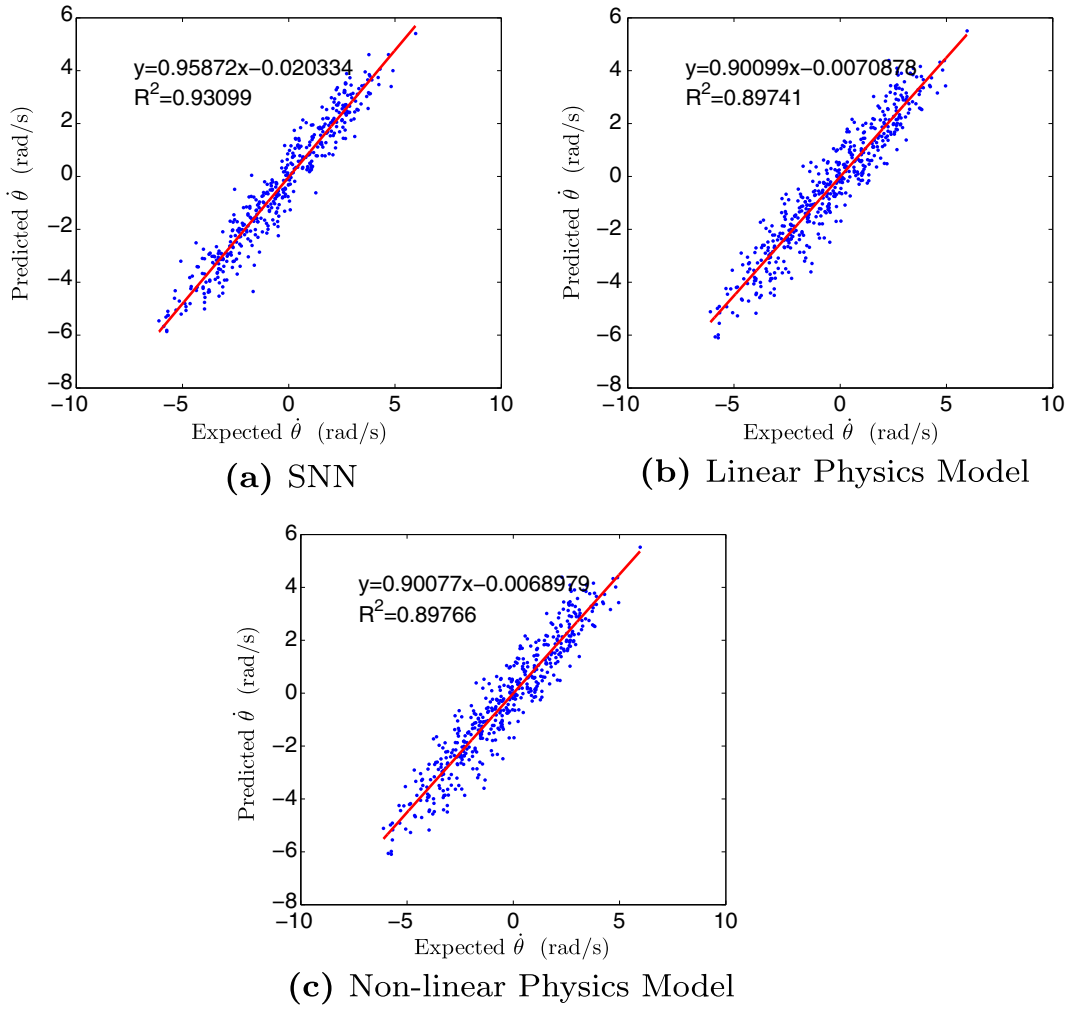


Fig. 7 Predicted vs expected values of $\dot{\theta}_{new}$

into account that this value of t_a could have potentially compensated not only for the actuator delay, but also for the delay in the sensors (Section 5.1). It was also noticed that the optimization process in some cases found parameter values which were on one of the boundaries of the interval considered for a certain parameter. Larger intervals were not considered, since these would have represented unrealistic parameter values. It was, however, found that if no restrictions were placed on the values that parameters could take on, this did not lead to a marked increase in the final accuracy of the optimized physics models.

Figures 6, 7 and 8 show a comparison between the values expected in the validation set for each of the three quantities to be predicted and the corresponding values predicted by the most accurate simulator of each type found during the training process. Also shown in each figure is a linear trendline added to the data as well as the equation of this line and the R^2 value (coefficient of determination) of the fit obtained.

It can be seen from the presented figures that all three the simulator types made predictions which correlated with expected values relatively well. It can, again, also be observed that the linear and non-linear physics models made predictions which were almost identical, as was discussed before.

The predictions of ψ_{new} were very accurate from all three simulators (evident from trendline slopes near one, trendline y-intercepts of nearly zero, and R^2-values

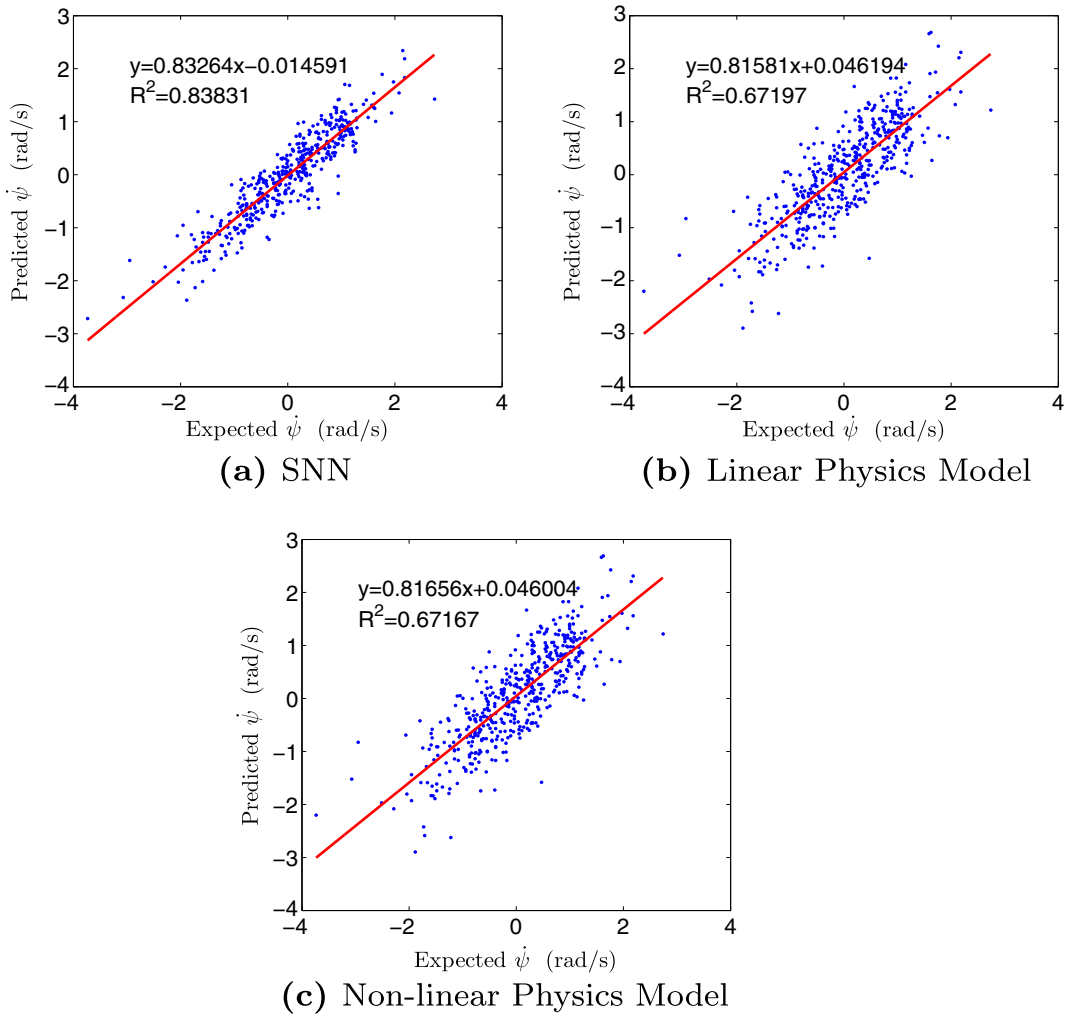


Fig. 8 Predicted vs expected values of $\dot{\psi}_{new}$

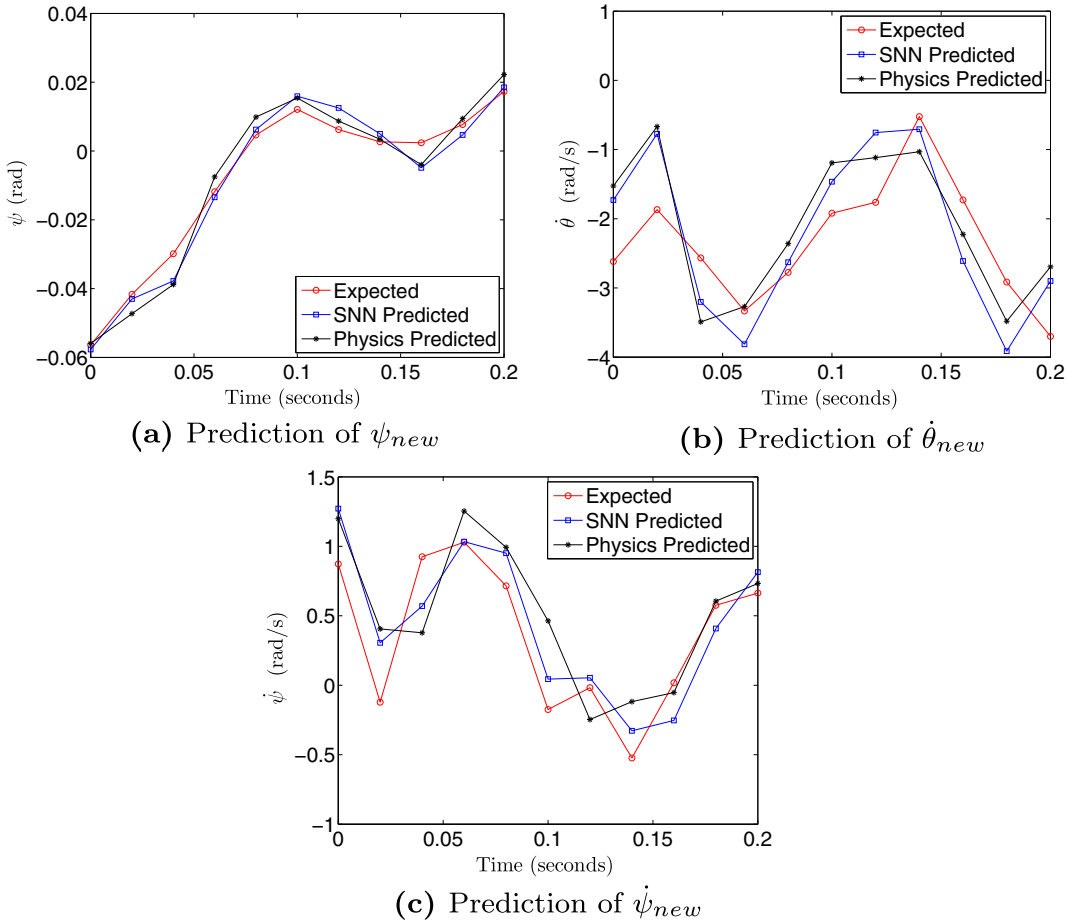
near one for all three plots in Fig. 6). Although the SNN was more accurate than the physics-based simulators, it was only barely so.

For the prediction of $\dot{\theta}_{new}$ and $\dot{\psi}_{new}$ more prominent differences are present in the accuracy of the SNN as opposed to the physics-based simulators, with the SNN in each case being more accurate. This is especially evident in Fig. 8, where the SNN has a trendline R^2-value of 0.83831, which is much closer to one than the corresponding values of the physics simulators ($R^2 \approx 0.67$). Since $\dot{\psi}_{new}$ is important in the prediction of the state of the robot and presumably also for the control of the robot, inaccuracies in this quantity could affect the transferability of evolved controllers from simulation to the real-world robot.

The three simulators are directly compared to expected data from the validation set in Fig. 9. This figure shows the expected values of ψ_{new}, $\dot{\theta}_{new}$ and $\dot{\psi}_{new}$ from the validation set for a 200 ms period of random motion of the robot, as well as the values predicted by the SNNs and the physics-based simulators. Since the predictions made by the two physics simulators were almost indistinguishable, these values are shown only once. Figure 9 shows (as was concluded from Figs. 6 to 8) that all the simulators were quite accurate in predicting ψ_{new}, with more inaccuracies present in predictions of $\dot{\theta}_{new}$ and $\dot{\psi}_{new}$.

7.2 Simulator Computational Efficiency

Apart from the accuracy of simulators used in the ER process being vital to the success of this approach, the computational efficiency of simulators also plays an important role, since a large number of candidate



(a) Prediction of ψ_{new}

(b) Prediction of $\dot{\theta}_{new}$

(c) Prediction of $\dot{\psi}_{new}$

Fig. 9 Comparison of prediction accuracies of different simulators

Table 5 Time taken by each simulator to determine fitness of one controller in simulation

Simulator	Simulation time (ms)
SNN	20.15
Linear Physics Model	9.22
Non-linear Physics Model	88.57

controllers typically have to be evaluated in simulation during the ER process. Table 5 shows the average amount of time taken by each of the developed simulators to assign a fitness value to one controller during the simulation-based controller evolution process (Section 6).

Since the linear physics model could be solved explicitly (Appendix A), the calculations needed in order to generate predictions from this model were simple and could therefore be performed quickly. This is clearly evident from Table 5, in that the linear physics model takes by far the least computational time to assign a fitness value to a candidate controller. The non-linear physics model took roughly nine times longer than the linear physics model. This is a result of the fact that this non-linear physics model was solved numerically (Section 5.1). The SNNs are comparatively reasonably computationally efficient, taking roughly twice as long as the linear physics model and roughly only a quarter as much time as the non-linear physics model.

7.3 Controller Performance on Real-World Robot

Each selected simulator (Section 7.1) was used to evolve 30 balancing controllers in simulation while injecting one of the four levels of noise into the simulation, as explained in Section 6. It should be noted that even though the linear and non-linear physics models produced very similar accuracies based on validation data (Section 7.1), both these types of simulators were used for controller evolution. This was done since, in the beginning stages of controller evolution a controller being simulated might have moved erratically (through large angles). The small angle approximation made in arriving at the linear physics model (Section 5) might not be valid under these circumstances, meaning that the non-linear physics model would be more accurate (at least in the beginning stages of evolution). Therefore, since it was believed that this could possibly influence the evolution process, both the linear and non-linear physics models were tested for controller evolution.

After the evolution process was finished in each case, the best controller (the controller with the largest fitness in simulation) was selected from the relevant controller population and evaluated on the real-world robot. Each controller was executed on the real-world robot and the amount of time measured that the controller was able to keep the robot balanced within a ψ range of $[-6, 6]$ degrees. A controller which allowed the robot to remain balanced for a time period of 30

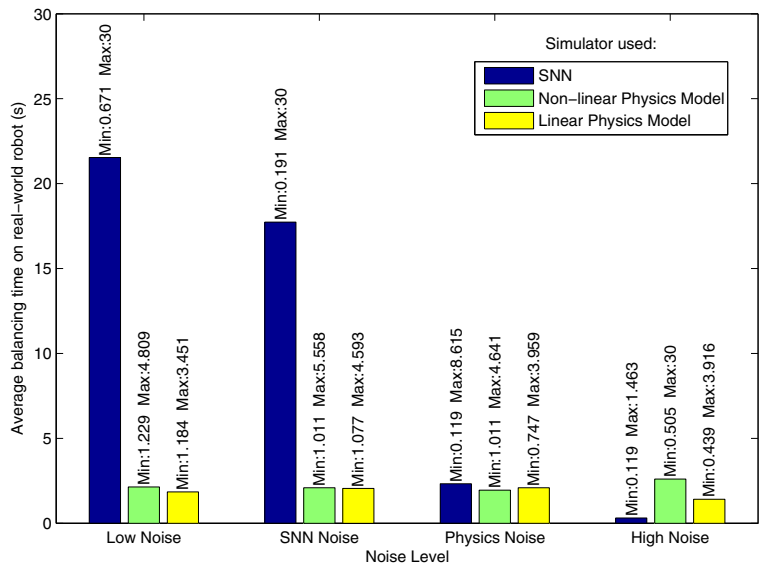


Fig. 10 Balancing times of controllers on real-world robot

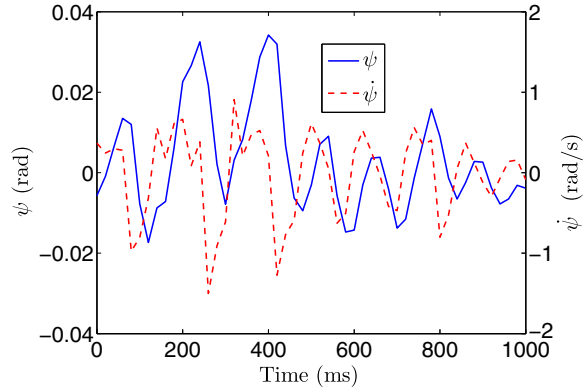
seconds was assumed to be capable of balancing the robot indefinitely and balancing times were therefore measured up to a maximum period of 30 seconds. The timer was also stopped if the robot moved more than 30 cm forwards or backwards from its starting position during execution of the controller. Results are shown in Fig. 10, and are based on the recorded balancing times from each real-world execution of a given controller. The figure shows the average, minimum and maximum balancing times in each case.

It can be seen from Fig. 10 that the balancing controllers evolved in simulation using the SNNs transferred to the real-world robot with much greater success than those evolved using either of the physics simulators. This was the case especially for those controllers evolved with the low noise and SNN noise levels. Few controllers were evolved that could keep the real-world robot balanced for extended periods of time using the physics-based simulators. It should, however, be noted (Fig. 10) that one controller evolved using the non-linear physics model was capable of balancing the robot for the entire 30 second time period. Apart from this one successful controller, large differences were not apparent in the performance of controllers evolved using the linear and non-linear physics models. The performance differences observed between controllers evolved using the SNNs and those evolved using the physics-based simulators, can likely be attributed to the differences in accuracy of these simulators (Section 7.1). Since the SNNs were shown to be more accurate than the optimized physics-based simulators, it was expected that these SNNs would lead to controllers which would be more successfully transferable than those evolved using the physics-based simulators.

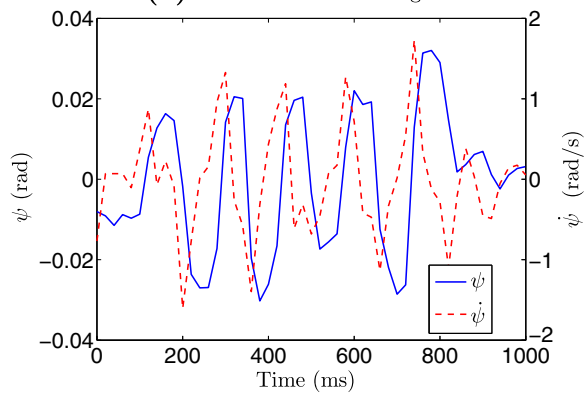
The fact that the best-performing controller evolved using the physics-based simulators was evolved on the highest level of noise tested (Fig. 10), suggests that this high level of noise could have partially compensated for some of the inaccuracies in the non-linear physics model. Conversely, the finding that controllers evolved using the SNNs performed best on the real-world robot when said controllers were evolved on the lower two levels of noise tested, suggests that the higher accuracy of the SNNs meant that less noise was required to compensate for the inaccuracies in these SNNs.

Figure 11 shows the performance of two controllers on the real-world robot which were able to stabilize the robot for the full 30 second period: one evolved using the SNNs, and the other evolved using the non-linear physics model (the only controller evolved using the physics models capable of balancing the robot for the full 30 second period). The figure shows ψ and $\dot{\psi}$ over a one second period of operation on the real-world robot. It can be seen from the figure that both controllers keep the robot stabilized, with ψ and $\dot{\psi}$ both remaining relatively close to zero over the time period and oscillating around the zero point. The operation of the two controllers is roughly the same, but the controller evolved using the SNNs leads to slightly more stability in that it restricts $\dot{\psi}$ to a smaller range than was the case with the controller evolved using the non-linear physics model.

The performance of evolved controllers which successfully managed to stabilize the real-world robot was thus not found to differ greatly, irrelevant of the simulator in which a given controller was evolved. The major difference between the SNNs and



(a) Controller evolved using SNN



(b) Controller evolved using non-linear physics model

Fig. 11 Real-world operation of evolved controllers

physics-based simulators was that the SNNs allowed for much greater transferability from simulation to the real-world robot.

8 Conclusions and Future Work

This study compared different approaches to constructing robotic simulators for a wheeled inverted pendulum robot, and the subsequent usage of these simulators in the simulation-based evolution of balancing controllers for the robot. Although this was not measured quantitatively, the discussion presented in this paper clearly demonstrated that much less human effort was required in the construction of the SNNs than was required for the physics-based simulators. This follows from the fact that, in developing the SNNs, no derivation of any mathematical equations was necessary (Section 5). A possible argument against the usage of SNNs as simulators in the ER process could be that data needs to be collected from the real-world robot from which these SNNs are constructed (Section 4). This data collection process can, in itself, be time-consuming. It was, however, shown that an experimenter would need to collect data from the real-world robot even in the case where physics-based simulators are used. This is a result of the fact that parameter values in these physics models needed to be optimized using experimental data, since unoptimized parameter values were shown to lead to inaccurate predictions from these physics models (Section 7.1).

Not only were the SNNs simpler to construct than the physics-based simulators, but this study furthermore illustrated that the SNNs were more accurate than the physics-based models, based on validation data, even after optimization of the parameters in the physics models (Section 7.1). This likely follows from the fact that the SNNs were constructed making use of only experimental data, and were not based on any assumptions about the structure of a model, as the physics models were. Any idiosyncrasies present in the data could thus be compensated for by the SNNs, whereas the physics models were likely not capable of compensating for these idiosyncrasies.

The SNNs were also shown to be computationally efficient in comparison to the physics-based simulators (Section 7.2). Although the linear physics model was more computationally efficient than the SNNs, this study illustrated that the SNNs were more computationally efficient than the non-linear physics model – the physics model which eventually produced the only balancing controller capable of keeping the robot upright for a time period of 30 seconds.

Most importantly, this study clearly illustrated that SNNs can not only be used as an alternative to physics-based simulators in the ER process, but, for the platform investigated in this study, drastically improve the transferability of evolved controllers from simulation to the real-world robot (Section 7.3). Therefore, SNNs can potentially provide a means of reducing the reality gap problem (Section 2.1), at least for certain robotic platforms.

The results presented in this study indicate that SNNs offer a viable alternative to physics-based simulation in ER, with these SNNs being simple to construct, computationally efficient and accurate. It is not known, however, whether the encouraging results obtained in this study using SNNs will scale to more complex robotic systems. Future studies could therefore compare SNNs and physics-based simulators in the ER process applied to robotic systems governed by more complex dynamics.

Appendix A: Analytical Solution of Linearized Equations

Taking into account the optimal values of parameters determined for the linear physics model, Eqs. 7 and 8 can be written as a corresponding non-homogeneous system of first-order linear differential equations as follows:

$$\begin{bmatrix} \dot{\theta} \\ \dot{\psi} \\ \dot{p} \\ \dot{q} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 35.352 & -19.324 & 13.828 \\ 0 & 52.316 & 1.435 & -3.237 \end{bmatrix} \begin{bmatrix} \theta \\ \psi \\ p \\ q \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 20.518 \\ -4.802 \end{bmatrix} v \tag{12}$$

where $p = \dot{\theta}$, $q = \dot{\psi}$. Since it is linear, the above system of differential equations can be solved analytically. A general solution to the system can be shown to be:

$$\begin{aligned}\begin{bmatrix}\theta\\ \psi\\ p\\ q\end{bmatrix} &= c_1\begin{bmatrix}9.594\times10^{-2}\\ 0.126\\ 0.599\\ 0.785\end{bmatrix}e^{6.241t} + c_2\begin{bmatrix}1.0\\ 0.0\\ 0.0\\ 0.0\end{bmatrix}\\ &+ c_3\begin{bmatrix}7.842\times10^{-2}\\ 9.038\times10^{-2}\\ -0.651\\ -0.750\end{bmatrix}e^{-8.297t}\\ &+ c_4\begin{bmatrix}4.848\times10^{-2}\\ -4.727\times10^{-3}\\ -0.994\\ 9.693\times10^{-2}\end{bmatrix}e^{-20.504t}\\ &+ \begin{bmatrix}1.171vt\\ (5.968\times10^{-2})v\\ 1.171v\\ 0\end{bmatrix}\end{aligned} \tag{13}$$

where $c_1, c_2, c_3, c_4 \in \mathbb{R}$.

References

1. Encog Machine Learning Framework — Heaton Research (Accessed January 2015). http://www.heatonresearch.com/encog
2. NXT Acceleration / Tilt Sensor (Accessed July 2016). https://www.hitechnic.com/cgi-bin/commerce.cgi?preadd=action&key=NAC1040
3. NXT Gyro Sensor (Accessed July 2016). https://www.hitechnic.com/cgi-bin/commerce.cgi?preadd=action&key=NGY1044
4. nxtOSEK/JSP: ANSI C/C++ with OSEK/μITRON RTOS for Lego MindStorms NXT (Accessed November 2014). http://lejos-osek.sourceforge.net/
5. Commons Math: The Apache Commons Mathematics Library (Accessed September 2014). http://commons.apache.org/proper/commons-math/
6. Atkeson, C.G., Santamaria, J.C.: A Comparison of Direct and Model-Based Reinforcement Learning. In: International Conference on Robotics and Automation, pp. 3557–3564. IEEE Press (1997)
7. Bonarini, A., Caccia, C., Lazaric, A., Restelli, M.: Batch Reinforcement Learning for Controlling a Mobile Wheeled Pendulum Robot. In: Artificial Intelligence in Theory and Practice II, pp. 151–160. Springer (2008)
8. Bongard, J., Zykov, V., Lipson, H.: Resilient machines through continuous self-modeling. Science **314**, 1118–1121 (2006)
9. Burden, R., Faires, J. Numerical Analysis, 8th Edn. Thomson Brooks/Cole, London (2005)
10. Canale, M., Brunet, S.C.: A Lego Mindstorms NXT Experiment for Model Predictive Control Education. In: European Control Conference (ECC) (2013)
11. Colton, S.: The balance filter: A simple solution for integrating accelerometer and gyroscope measurements for a balancing platform. White paper Massachusetts Institute of Technology (2007)
12. De Nardi, R., Holland, O.E.: Coevolutionary modelling of a miniature rotorcraft. Intelligent Autonomous Systems **10** (2008). IAS-10
13. Drumwright, E., Hsu, J., Koenig, N., Shell, D.: Extending Open Dynamics Engine for robotics simulation. Simulation, Modeling, and Programming for Autonomous Robots **6472**, 38–50 (2010)
14. El-Hawwary, M.I., Elshafei, A.L., Emara, H.M., Fattah, H.A.A.: Adaptive fuzzy control of the inverted pendulum problem. IEEE Trans. Control Syst. Technol. **14**(6), 1135–1144 (2006)
15. Engelbrecht, A.P. Computational Intelligence: An Introduction, 2nd edn. Wiley, West Sussex (2007)
16. Floreano, D., Mondada, F.: Evolution of homing navigation in a real mobile robot. IEEE Trans. Syst. Man Cybern. **26**(3), 396–407 (1996)
17. Glette, K., Klaus, G., Zagal, J.C., Torresen, J.: Evolution of Locomotion in a Simulated Quadruped Robot and Transferral to Reality. In: Proceedings of the Seventeenth International Symposium on Artificial Life and Robotics (2012)
18. Gomez, F.J., Miikkulainen, R.: Transfer of Neuroevolved Controllers in Unstable Domains. In: Proceedings of the Genetic Evolutionary Computation Conference (GECCO-04) (2004)
19. Gürocak, H.: A genetic-algorithm-based method for tuning fuzzy logic controllers. Fuzzy Set. Syst. **108**(1), 39–47 (1999)
20. Hapke, M., Komosinski, M.: Evolutionary design of interpretable fuzzy controllers. Foundation of Computing and Decision Sciences **33**(4), 351 (2008)
21. Hartland, C., Bredeche, N.: Evolutionary Robotics, Anticipation and the Reality Gap. In: IEEE International Conference on Robotics and Biomimetics (2006)
22. Heidrich-Meisner, V., Igel, C.: Neuroevolution strategies for episodic Reinforcement Learning. J. Algoritm. **64**(4), 152–168 (2009)
23. Herrera, F., Lozano, M., Verdegay, J.: Tuning fuzzy logic controllers by genetic algorithms. Int. J. Approx. Reason. **12**(3), 299–315 (1995)
24. Holland, J.H.: Adaptation in Natural and Artificial Systems. MIT Press, Cambridge (1992)
25. Hotz, P., Gómez, G.: The Transfer Problem from Simulation to the Real World in Artificial Evolution. In: Workshop and Tutorial Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX) (2004)

26. Huang, C.H., Wang, W.J., Chiu, C.H.: Design and implementation of fuzzy control on a two-wheel inverted pendulum. IEEE Trans. Ind. Electron. **58**(7), 2988–3001 (2011)
27. Igel, C.: Neuroevolution for Reinforcement Learning Using Evolution Strategies. In: Congress on Evolutionary Computation (CEC), vol. 4, pp. 2588–2595 (2003)
28. Jakobi, N.: Minimal simulations for evolutionary robotics. Ph.D. thesis University of Sussex (1998)
29. Jakobi, N.: Running across the Reality Gap: Octopod Locomotion Evolved in a Minimal Simulation. In: Evolutionary Robotics, pp. 39–58. Springer (1998)
30. Jakobi, N., Husbands, P., Harvey, I.: Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics, vol. 929. Springer, Berlin (1995)
31. Kawada, K., Yamamoto, T., Mada, Y.: A Design of Evolutionary Recurrent Neural-Net Based Controllers for an Inverted Pendulum. In: Fifth Asian Control Conference (2004)
32. Klaus, G., Glette, K., Tørresen, J.: A Comparison of Sampling Strategies for Parameter Estimation of a Robot Simulator. In: Proceedings of the Third International Conference on Simulation, Modeling, and Programming for Autonomous Robots (2012)
33. Koos, S., Cully, A., Mouret, J.: Fast damage recovery in robotics with the T-Resilience algorithm. Preprint version (available: arXiv:1302.0386,2013)
34. Koos, S., Mouret, J., Doncieux, S.: The Transferability Approach: Crossing the reality gap in Evolutionary Robotics. IEEE Trans. Evol. Comput., 1–25 (2012)
35. Li, Z., Xu, C.: Adaptive fuzzy logic control of dynamic balance and motion for wheeled inverted pendulums. Fuzzy Set. Syst. **160**(12), 1787–1803 (2009)
36. Lipson, H., Bongard, J., Zykov, V., Malone, E.: Evolutionary Robotics for Legged Machines: From Simulation to Physical Reality. In: Proceedings of the 9th International Conference on Intelligent Autonomous Systems (2006)
37. Moeckel, R., Perov, Y.N., Nguyen, A.T., Vespignani, M., Bonardi, S., Pouya, S., Sproewitz, A., van den Kieboom, J., Wilhelm, F., Ijspeert, A.J.: Gait Optimization for Roombots Modular Robots - Matching Simulation and Reality. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2013)
38. Nakamura, S., Hashimoto, S.: Hybrid Learning Strategy to Solve Pendulum Swing-Up Problem for Real Hardware. In: IEEE International Conference on Robotics and Biomimetics (2007)
39. Nelles, O.: Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models. Springer, Berlin (2010)
40. Nelson, A.L., Barlow, G.J., Doitsidis, L.: Fitness functions in evolutionary robotics: A survey and analysis. Robot. Auton. Syst. **57**(4), 345–370 (2009)
41. Nguyen-Tuong, D., Peters, J.: Model learning for robot control: A survey. Cogn. Process. **12**(4), 319–340 (2011)
42. Oh, S.K., Pedrycz, W., Rho, S.B., Ahn, T.C.: Parameter estimation of fuzzy controller and its application to inverted pendulum. Eng. Appl. Artif. Intell. **17**(1), 37–60 (2004)
43. Pratihar, D.K.: Evolutionary Robotics - A review. Sadhana **28**(6), 999–1009 (2003)
44. Pretorius, C.J., du Plessis, M.C., Cilliers, C.B.: Towards an Artificial Neural Network-Based Simulator for Behavioural Evolution in Evolutionary Robotics. In: Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists, pp. 170–178. ACM (2009)
45. Pretorius, C.J., du Plessis, M.C., Cilliers, C.B.: A Neural Network-Based Kinematic and Light-Perception Simulator for Simple Robotic Evolution. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1–8 (2010)
46. Pretorius, C.J., du Plessis, M.C., Cilliers, C.B.: A Neural Network Ultrasonic Sensor Simulator for Evolutionary Robotics. In: INFOCOMP 2012, the Second International Conference on Advanced Communications and Computation, pp. 54–61 (2012)
47. Pretorius, C.J., du Plessis, M.C., Cilliers, C.B.: Simulating robots without conventional physics: a Neural Network approach. J. Intell. Robot. Syst. **71**(3–4), 319–348 (2013)
48. Pretorius, C.J., du Plessis, M.C., Gonsalves, J.W.: A Comparison of Neural Networks and Physics Models as Motion Simulators for Simple Robotic Evolution. In: IEEE Congress on Evolutionary Computation (CEC) (2014)
49. Riedmiller, M., Braun, H.: A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. In: IEEE International Conference on Neural Networks (1993)
50. Schneider, S., Igel, C., Klaes, C., Dinse, H.R., Wiemer, J.C.: Evolutionary adaptation of nonlinear dynamical systems in computational neuroscience. Genet. Program Evolvable Mach. **5**(2), 215–227 (2004)
51. Stanley, K.O., Miikkulainen, R.: Evolving Neural Networks Through Augmenting Topologies. Evol. Comput. **10**(2), 99–127 (2002)
52. Sutton, R., Barto, A.: Reinforcement Learning: an Introduction. MIT Press, Cambridge (1998)
53. Whitley, D., Dominic, S., Das, R., Anderson, C.W.: Genetic Reinforcement Learning for neurocontrol problems. Mach. Learn. **13**(2), 259–284 (1993)
54. Wieland, A.: Evolving Neural Network Controllers for Unstable Systems. In: International Joint Conference on Neural Networks (1991)
55. Yamamoto, Y.: NXTway-GS Model-Based Design - Control of self-balancing two-wheeled robot built with LEGO Mindstorms NXT. Online (2008). http://www.pages.drexel.edu/dml46/Tutorials/BalancingBot/files/NXTway-GS
56. Yi, J., Yubazaki, N.: Stabilization fuzzy control of inverted pendulum systems. Artif. Intell. Eng. **14**(2), 153–163 (2000)
57. Zagal, J.C., Ruiz-del Solar, J.: Combining simulation and reality in Evolutionary Robotics. J. Intell. Robot. Syst. **50**(1), 19–39 (2007)

Christiaan J. Pretorius obtained his MSc at the Nelson Mandela Metropolitan University. He is currently a lecturer in Applied Mathematics at the same institution and is pursuing a PhD in Applied Mathematics. Mr Pretorius has interests in Computational Intelligence and Evolutionary Robotics, as well as quantitative and qualitative analysis of differential equations.

Mathys C. du Plessis obtained his PhD at the University of Pretoria. He is currently a lecturer in Computer Science at the Nelson Mandela Metropolitan University. Dr du Plessis has interests in Computational Intelligence and Evolutionary Robotics.

John W. Gonsalves obtained his PhD from the University of Port Elizabeth (now known as Nelson Mandela Metropolitan University). He has held posts at this institution as well as the University of KwaZulu-Natal, and the Council for Scientific and Industrial Research. Prof Gonsalves has interests in mathematical modelling, theoretical chemistry, coastal engineering and dynamical systems.