2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

A Comparison of Neural Networks and Physics Models as Motion Simulators for Simple Robotic Evolution

Christiaan J. Pretorius, Mathys C. du Plessis and John W. Gonsalves

Abstract— **Robotic simulators are used extensively in Evolutionary Robotics (ER). Such simulators are typically constructed by considering the governing physics of the robotic system under investigation. Even though such physics-based simulators have seen wide usage in ER, there are some potential challenges involved in their construction and usage. An alternative approach to developing robotic simulators for use in ER, is to sample data directly from the robotic system and construct simulators based solely on this data. The authors have previously shown the viability of this approach by training Artificial Neural Networks (ANNs) to act as simulators in the ER process. It is, however, not known how this approach to simulator construction will compare to physics-based approaches, since a comparative study between ANN-based and physics-based robotic simulators in ER has not yet been conducted.**

This paper describes such a comparative study. Robotic simulators for the motion of a differentially-steered mobile robot were constructed using both ANN-based and physics-based approaches. These two approaches were then compared by employing each of the developed simulators in the ER process to evolve simple navigation controllers for the experimental robot in simulation. Results obtained indicated that, for the robotic system investigated in this study, ANN-based robotic simulators offer a promising alternative to physics-based simulators.

I. Introduction

Evolutionary Robotics (ER) is a process used to semi-automatically develop controllers for robots through artificial evolution [1]. The fact that evolution is used to construct and optimize the controller of a robot during the ER process means that the need for human input is minimal, making ER an attractive alternative to manual controller development by human programmers [2].

As with other Evolutionary Algorithms, the ER process evolves a population of candidate solutions (robot controllers), with the aim of producing an optimal controller that can effectively perform a pre-defined task when uploaded to a real-world robot. An important aspect during this evolution process is the ability to quantify the relative performance of each controller in the population in performing the required task. A fitness function is used to achieve this [3].

To determine the fitness of a candidate controller in the ER population, the controller can be uploaded to the real-world robot [2]. Such real-world evaluation of controller performance can, however, be an extremely time-consuming process and can possibly damage robotic hardware, since many (typically in the order of tens of thousands [4]) candidate controllers need to be evaluated during a typical ER process. Owing to the challenges in performing ER on real-world robots, evolution in simulation has been considered by various researchers [4], [5]. This process involves making use of a robot simulator to simulate the robot and its interaction with its environment, so that the fitness of candidate controllers can be evaluated without needing to upload these controllers to the real-world robot.

The current work aims to compare two different approaches to constructing such simulators for ER, namely Artificial Neural Network (ANN)-based simulators and physics-based simulators. The remainder of this paper is presented as follows: Section II introduces different approaches that can be followed in robotic simulator construction for ER. A motivation is provided for the investigation attempted in the current study, and the methodology employed to compare different approaches to simulator construction is then outlined in Section III, followed by a discussion on the experimental hardware employed and the method used to acquire data from which robotic simulators were constructed (Section IV). Section V discusses the processes followed to construct the different robotic simulators. In order to gauge the effectiveness of each of these simulators in the ER process, robotic controllers were evolved using each simulator (Section VI). Results are presented and discussed in Section VII and conclusions are drawn (Section VIII).

II. Approaches to Simulation in ER

As discussed in Section I, robotic simulators are often used to accelerate and simplify the ER process. It is vital that controllers evolved in ER using such robotic simulators perform the required task effectively when uploaded to real-world robotic hardware, since this is the ultimate goal of ER. It is thus very important that robotic simulators used during the ER process accurately model the real-world behaviour of the robot under consideration. In addition, since a very large quantity of candidate controllers are typically evaluated during the ER process (Section I), the computational efficiency of robotic simulators employed during the ER process is of great importance [6].

To construct robotic simulators for ER, various techniques can be applied. The two methods of simulator construction for ER that will be investigated in the current study are now briefly outlined and contrasted.

Christiaan J. Pretorius and John W. Gonsalves are with the Department of Mathematics and Applied Mathematics, and Mathys C. du Plessis with the Department of Computing Sciences, Nelson Mandela Metropolitan University (NMMU), Port Elizabeth, South Africa (email addresses: {cpretorius, mc.duplessis, john.gonsalves}@nmmu.ac.za).

Financial support towards this research from the National Research Foundation (UID number: 79570), as well as from the NMMU Science Faculty is gratefully acknowledged.

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

A. Physics-based Robotic Simulation

Robotic simulators used in ER are often based on the physics governing the robotic system under consideration [4], [7]. The construction of physics-based robot simulators can be a complex process, since various characteristics of the robot system to be simulated need to be incorporated into such a simulator, such as complex body shapes and factors such as friction [2]. Physics-based robotic simulators furthermore often make simplifying assumptions of real-world physics to simplify the implementation of these simulators and improve their computational efficiency [6]. This can have a negative impact on the accuracy of these simulators.

Evidence does exist that controllers evolved in simulation using physics-based simulators do not always perform effectively when these controllers are executed on real-world robots [3], meaning that these simulators are not necessarily accurate enough to simulate the relevant features of the experimental robot. The calculations involved in physics-based simulators can also be computationally demanding [8] and, as a result, the computational efficiency of physics-based simulators can become a prohibitive factor for using these simulators in ER.

B. ANNs as Robotic Simulators

As a result of the challenges involved in using physics-based simulators in the ER process, the authors previously explored an alternative approach to simulator development by constructing ANNs from experimentally-collected data to act as simulators during the ER process [9], [10]. ANNs were advocated for usage as robotic simulators due to their noise tolerance and generalization capabilities [10]. It was shown that various robotic controllers can be evolved in ER and successfully transferred to real-world robots when making use of such ANN simulators. The notion of developing robotic simulators by learning from empirical data has been explored by others [11], [12] and is essentially an application of system identification [13]. This approach has, however, not seen wide application in the field of ER.

It is proposed that employing ANNs as robotic simulators in ER can potentially address some of the challenges involved in traditional physics-based approaches to simulator construction. Since ANN-based robotic simulators are constructed exclusively from empirically-collected data, the physics governing the robotic system under consideration does not need to be explicitly considered in the construction of ANN-based simulators. This can potentially reduce the amount of human effort required during simulator construction. Also, idiosyncrasies of the robotic system which could be omitted from a physics-based simulator due to simplifying assumptions made during the construction of said simulator (Section II-A), could be taken into account in an ANN-based simulator, since these idiosyncrasies will also be present in the data from which such a simulator is constructed. This means that ANN-based robotic simulators offer the potential of accurately simulating the operation of real-world robots. Furthermore, since ANNs can avoid the complex calculations involved in some physics-based robotic simulators (Section II-A), it is envisaged that ANN-based simulators could be computationally efficient as robotic simulators in ER.

III. Motivation for and Outline of the Current Study

Since both the approaches to robotic simulator construction for ER presented in Section II have been shown to be viable for usage in ER, and both approaches have apparent advantages and disadvantages to their usage, a comparative study between these two approaches appears warranted. Such a comparative study of ANN-based and physics-based simulators in ER has not yet been attempted. The research presented here therefore attempts such a comparative study, and represents a first attempt at quantitatively comparing these two approaches to simulator construction in ER.

As an initial attempt at such a comparative study, a simple robotic platform was selected, namely a differentially-steered mobile robot (Section IV). Both physics-based and ANN-based simulators were developed for the motion of this robot (Section V) and were subsequently used in the ER process to evolve simple navigation controllers for the robot (Section VI). During this process, the human effort needed during simulator construction was compared for the different robotic simulators[1], as well as the accuracy and computational efficiency of each simulator, and the real-world performance of controllers evolved using each simulator.

IV. Robotic Hardware and Data Acquisition

This study made use of a Khepera III mobile robot [14]. The Khepera III is a differentially-steered robot, meaning that it is driven by two wheels, each connected independently to a motor. In this study, simulators were developed for the motion of the experimental robot in a horizontal plane (Section V). The operating environment of the robot therefore consisted of a horizontal skid-proof surface. A Nintendo Wii remote [15] was mounted directly above this surface using a purpose-built supporting frame, so as to face perpendicularly downward onto the surface from a height of roughly 1.8m.

In order to collect data related to the real-world motion of the experimental robot, a motor speed for each of the robot's two wheels was randomly generated, as well as a time period that these speeds were to be maintained. Each speed and time period was generated from a uniform distribution to ensure that the resulting data would uniformly cover the entire search space. The generated wheel speeds were then maintained by the robot for the required time period after which another pair of wheel speeds and time duration were generated randomly and maintained by the robot. This process was repeated numerous times. Collisions with the boundaries of the working surface were not considered in this study. While the robot was moving in response to the

[1] A quantitative comparison of the human effort needed during simulator construction was not done in this study due to the inherent subjectivity that would be involved in such a comparison. Rather, comments are given in the text to illustrate the possible differences in the amount of human effort needed to construct the different robotic simulators.

generated commands, the Wii remote was used to track the motion of the robot by tracking an arrangement of infrared lights mounted on top of the robot. This tracking process and the calibration of the Wii remote needed to allow for accurate tracking were implemented using a modification of code by Lee [16].

The acquired data was separated into three distinct sets: a *training set*, *validation set* and *test set*. These sets contained 1000, 230 and 230 data patterns, respectively. The usages of these data sets as well as the different simulators developed for the motion of the experimental robot are discussed in the next section.

V. Simulator Development

The main aim of this study was to quantitatively compare physics-based and ANN-based robotic simulators for the motion of the experimental robot in the ER process. Since the robot was constrained to moving in a horizontal plane (Section IV), the state of the robot at any point during its motion could be described by three variables: two for the position of the robot in the plane (x- and y-coordinates in a Cartesian coordinate system) and one for its orientation angle (denoted by θ), that is the angle between the forward-facing direction of the robot and some fixed vector in its operating environment.

The task of each of the simulators developed in this work would be as follows: Given that a series of commands are issued to the robot, each in the form of a motor speed for each of its two wheels and a time duration for these speeds to be maintained by the robot, the simulators were to predict the nett change in position and orientation of the robot as a result of each of the commands in the series. For each command issued to the robot, v_L^{cur} and v_R^{cur} are used to denote the linear speeds of the left and right wheels, respectively, issued as part of said command. Since the wheel speeds maintained by the robot before receiving a certain command (that is, as part of the preceding command) would also have an influence on the robot's motion while executing the current command (due to the inertia of the robot), these speeds were also taken into account by some of the simulators developed in this work. The left and right wheel speeds maintained by the robot as part of the previous command are denoted, respectively, by v_L^{prev} and v_R^{prev}. Lastly Δt is used to denote the time period for which a certain command is to be executed by the robot.

A Cartesian coordinate system was constructed and allowed to move with the robot during its motion. The nett changes in the x-coordinate and y-coordinate of the axle-centre of the robot in response to a certain command were to be predicted in this local coordinate system by the developed simulators. These changes are denoted by Δx and Δy, respectively. The change in orientation angle of the robot in response to a certain command was simply measured in a static global coordinate system and is denoted by $\Delta\theta$.

Three different simulators were developed for the motion of the robot: two physics-based simulators and one ANN-based simulator. The development of each of these simulators is now discussed.

A. Physics-based Simulators

Two physics models were implemented in this work: a kinematic model and a dynamic model. The kinematic model makes various simplifying assumptions and was thus anticipated to be computationally efficient, but the predictions made by this model were expected to be relatively inaccurate. The dynamic model, conversely, takes various factors related to the motion of the robot into account which are neglected by the kinematic model. This meant that the dynamic model would likely be less computationally efficient but more accurate than the kinematic model. These physics-based simulators are now presented in more detail.

1) Kinematic Model: The kinematic model of differential steering does not take the inertia of the robot or any other complicating factors related to robotic motion into account. Under these assumptions Δx, Δy and $\Delta\theta$ during the time period $[0, \Delta t]$ can be calculated using [17]:

$$\Delta x = \frac{d(v_R^{cur} + v_L^{cur})}{2(v_R^{cur} - v_L^{cur})}\left[\sin\left(\frac{(v_R^{cur} - v_L^{cur})\Delta t}{d} + \theta_0\right) - \sin(\theta_0)\right] \quad (1)$$

$$\Delta y = -\frac{d(v_R^{cur} + v_L^{cur})}{2(v_R^{cur} - v_L^{cur})}\left[\cos\left(\frac{(v_R^{cur} - v_L^{cur})\Delta t}{d} + \theta_0\right) - \cos(\theta_0)\right] \quad (2)$$

$$\Delta\theta = \frac{(v_R^{cur} - v_L^{cur})\Delta t}{d} \quad (3)$$

where d is the robotic axle length and $\theta_0 = \theta(0)$. It can be seen from equations (1) to (3) that due to the simplifying assumptions on which this model is based, the kinematic model does not take into account the wheel speeds maintained by the robot before receiving a certain command (v_L^{prev} and v_R^{prev}). The axle length (d) and other parameters required in the kinematic model could be accurately measured directly. These parameter values were thus considered fixed and were not subjected to optimization as was done with the dynamic model (Section V-A.2), since changes made to their values through optimization would lead to a physically unrealistic model.

2) Dynamic Model: The dynamic model applied in this study is based on work by Laut [18]. For the sake of brevity the derivation of this model and other technical details related to this model will not be presented here. The interested reader is referred to Laut [18] for more details.

In the dynamic model, α_1,..., α_6 are parameters depending on various physical characteristics of the robot and a is the distance between the axle of the robot and the point on the robot for which the motion is to be predicted. Since this work was concerned with the motion of the axle-centre of the robot, $a = 0$. v_{ref} and ω_{ref} are the reference linear and angular velocities of the robot and can easily be expressed in terms of the left and right wheel speeds (v_L^{cur} and v_R^{cur}). Using these quantities, the rate of change in the x-coordinate, y-coordinate, orientation angle, linear velocity (v) and angular velocity (ω) of the robot respectively, can be

expressed by the following non-linear system of five first-order coupled differential equations [18]:

$$\frac{dx}{dt} = v\cos\theta - a\omega\sin\theta \quad (4)$$

$$\frac{dy}{dt} = v\sin\theta + a\omega\cos\theta \quad (5)$$

$$\frac{d\theta}{dt} = w \quad (6)$$

$$\frac{dv}{dt} = \frac{\alpha_3}{\alpha_1}\omega^2 - \frac{\alpha_4}{\alpha_1}v + \frac{v_{ref}}{\alpha_1} \quad (7)$$

$$\frac{d\omega}{dt} = -\frac{\alpha_5}{\alpha_2}v\omega - \frac{\alpha_6}{\alpha_2}\omega + \frac{\omega_{ref}}{\alpha_2} \quad (8)$$

To determine Δx, Δy and $\Delta\theta$ in response to wheel speeds maintained by the robot, the system of differential equations (equations (4) to (8)) was solved for each time interval $[0, \Delta t]$ corresponding to a command issued to the robot, taking appropriate initial conditions into account in each case. These initial conditions were based on v_L^{prev} and v_R^{prev} meaning that, unlike the kinematic model, the dynamic model did take into account these previous wheel speeds. The system was solved numerically using a Runge-Kutta fourth-order method [19]. This numerical method has a high-order local truncation error [19] and was thus expected to give numerical results with high accuracy. In the Runge-Kutta method, a constant time-step of 10ms was used, as this was determined to be the largest time-step that would lead to stable solutions of the system of differential equations.

Accurate determination of the parameters α_1,..., α_6 can be a challenging task since these parameters depend on various physical quantities, for example the moment of inertia of the robot about its local z-axis [18]. Furthermore, deriving the differential equations describing the motion of the robot (equations (4) to (8)) is also not a trivial task and takes roughly five pages of mathematical derivation by Laut [18]. This illustrates some of the challenges involved in the construction of accurate physics-based robotic simulators.

In this study the values of the parameters to be used in the dynamic model (α_1,..., α_6) were determined using a Genetic Algorithm (GA) [20]. Other optimization techniques could also have been employed, but since the GA allowed for successful optimization of the parameters, other techniques were not considered. Each individual in the GA population encoded potential parameter values to be used in the dynamic model. The error function to be minimized by the GA was calculated as:

$$\sum_{i=1}^{1000}[(\Delta x_i^E - \Delta x_i^P)^2 + (\Delta y_i^E - \Delta y_i^P)^2 + (\Delta\theta_i^E - \Delta\theta_i^P)^2] \quad (9)$$

where Δx_i^E is the expected change in robotic x-coordinate associated with pattern i from the training set (Section IV) and Δx_i^P is the corresponding change in x-coordinate as predicted by the dynamic physics model making use of the parameters encoded in a certain individual in the GA population. Δy_i^E, Δy_i^P, $\Delta\theta_i^E$ and $\Delta\theta_i^P$ are defined analogously. The quantities in equation (9) were normalized to ensure that each of the three terms in this equation contributed roughly equally to the error function. Further details of the GA used during this optimization process are given in Table I. The GA was run for 1000 generations. During this process the algorithm was monitored for overfitting [20] using the validation set (Section IV), but no overfitting was found to occur. In multiple runs of the GA, the GA was consistently found to converge to within 3 decimal places of a specific combination of parameter values and these values were thus considered optimal. The optimal parameter values are shown in Table II and were subsequently used in the dynamic model for controller evolution (Section VI). Laut also determined optimal values for α_1,..., α_6 [18], but these were based on a slightly different physical configuration of the robot. The values determined in the current work were found to lead to more accurate predictions from the dynamic model than those determined by Laut, as expected.

TABLE I
DETAILS OF GA USED FOR PARAMETER OPTIMIZATION

Population Size	200
Initialization	Random from a uniform distribution
Selection Method	Tournament Selection (Tournament size 40)
Crossover Probability	80%
Crossover Method	Simulated Binary Crossover
Mutation Probability	5%
Mutation Method	Random Component Perturbation

B. ANN-based Simulators

Unlike the physics-based simulators, ANN-based simulators developed in this work did not explicitly take any of the physics governing the motion of the robot into account. Instead, these ANN-based simulators were constructed entirely from data collected from the robot system under consideration. This illustrates that ANN-based robot simulators can potentially reduce the need for human input in the simulator development process. The topologies used for the ANN-based simulators are now discussed, along with the ANN training process.

1) ANN Topologies: To develop the ANN-based simulators for the motion of the experimental robot, use was made of Feed-Forward Neural Networks (FFNNs) [20] with one hidden layer each, consisting of 20 hidden neurons. Similar FFNN-based simulators have previously been shown to be viable in modeling differential steering [9]. Furthermore, previous research by the authors [21] suggested that more involved ANN topologies for modeling differential steering (such as Recurrent ANNs) do not offer considerable accuracy improvements over simpler FFNN topologies. All neurons in the ANNs developed in this work were implemented as summation units and neurons in the hidden layer of each

TABLE II
OPTIMAL PARAMETER VALUES FOR DYNAMIC MODEL

Parameter	Value
α_1	0.087
α_2	0.101
α_3	1.17×10^{-8}
α_4	0.989
α_5	6.00×10^{-6}
α_6	0.989

ANN used sigmoid activation functions, whereas neurons in the output layer used linear activation functions. Three separate ANNs were constructed: one each for the prediction of Δx, Δy and $\Delta\theta$. Each ANN took five inputs: v_L^{cur}, v_R^{cur}, v_L^{prev}, v_R^{prev} and Δt.

2) ANN Training: For ANN training, use was made of the Levenberg-Marquardt training algorithm [22]. Informal tests on other training algorithms, such as Gradient Descent, revealed that the Levenberg-Marquardt algorithm generally trained the ANNs more accurately and thus further lead to the decision to employ this training algorithm. In order to construct and train the ANNs employed in this work, use was made of the Encog Machine Learning Framework [23]. All weights corresponding to each ANN were randomly initialized and network training was employed to minimize the Mean Squared Error (MSE) [20] of each ANN based on the data in the training set of the ANN (Section IV). Training was allowed to continue until the MSE of a given ANN as calculated over the relevant ANN's validation set was seen to increase in ten successive iterations of the training algorithm. This stopping condition was used to avoid overfitting [20]. The training process was performed 30 times for each ANN and the ANN with the lowest MSE value based on validation data out of the 30 training runs was selected for further usage as a robotic simulator in the controller evolution process (Section VI). This was done since ANN simulators were required that could generalize well on data not presented during the training phase.

VI. Controller Evolution

To test the viability of each of the developed simulators in the ER process, each simulator was subsequently used in the evolution of a simple navigation controller for the experimental robot. The task to be performed by an evolved controller was chosen as follows: Four square blocks of side length 30cm were placed in the corners of the rectangular working surface of the robot. These blocks can be seen in Figure 1. A navigational controller evolved for the robot would then be expected to allow the robot, originally placed in the centre of the working surface, to pass through each of these squares (in no particular order) after which the robot was to return to its original position (the centre of the working surface). This task was also to be completed as quickly as possible without the robot leaving the outer perimeter of the working surface at any time during execution of the task.

Simple open-loop controllers were evolved to achieve the desired robotic behaviour (these controllers took no sensory inputs).[2] Controllers encoded a list of 10 commands (each command consisting of a speed for each of the two robot wheels as well as a time period to maintain each of these pairs of speeds). The possibility was also introduced for one or more of the 10 commands in a certain controller to be a null command (a command that would be ignored during execution of the controller), effectively meaning that 9 or less commands could also be used in a certain controller.

During the evolution process, the fitness of a candidate controller in the ER population was determined by using one of the developed simulators to construct the path followed by the robot in response to the commands encoded in that particular controller. Based on this path, the controller was assigned a fitness using the following fitness function:

$$Fitness = \begin{cases} k_1 n - k_2 t - k_3 f - k_4 e; & \text{if } n = 4 \\ k_1 n - k_2 t - k_3 c - k_4 e; & \text{otherwise} \end{cases} \quad (10)$$

In equation (10), k_1,..., k_4 are constants for which values were chosen manually as $k_1 = 1000$, $k_2 = 2.5s^{-1}$, $k_3 = 5cm^{-1}$ and $k_4 = 2.5cm^{-1}$. Furthermore, n is the number of blocks that the robot passes through successfully, t is the total execution time of the controller, f is the distance between the final position of the robot and the target position in the centre of the working surface, c is the distance between the final position of the robot and the centre of the nearest block not yet visited by the robot and e is the sum of the distances by which the robot moves outside the perimeter of the working surface, for all points making up a specific path.

This fitness function thus rewards controllers that allow the robot to pass through as many blocks as possible. Simultaneously, controllers are punished if they cause the robot to take a large amount of time to perform the required task or to leave the perimeter of the working surface. It can be seen that the two cases in the fitness function differ in their respective third terms. In the first case, since $n = 4$, the robot has successfully moved through all four blocks and therefore the third term in the fitness function aims to reward a controller for finishing as close as possible to the target position in the centre of the working surface. In the second case, since $n < 4$, there exists at least one block that has not been visited by the robot. The third term in the fitness function then aims to reward a controller for allowing the robot to finish as close as possible to the closest block not yet visited. This was done to encourage the evolution process to evolve controllers that would cause the robot to gradually move closer to an unvisited block as the evolution process proceeded and, eventually, visit this block.

The ER process applied to evolve navigation controllers was implemented as a GA. Each chromosome in this GA simply encoded the wheel speeds and command durations for each of the commands of which a certain controller comprised. The details of the GA employed for controller evolution are the same as those listed in Table I, with the exception that a mutation probability of 15% was used since this increase in exploration capabilities of the GA was found to accelerate the evolution process whilst still producing effective controllers. Evolution was allowed to proceed for 10 000 generations. This evolution process was repeated 20 times using each of the developed simulators, and thus 20 controllers were evolved using each simulator. After 10 000

[2]Although inputs from sensors might have allowed the robot to perform the task more effectively, open-loop controllers were specifically chosen since this would allow for rigorous testing and comparison of the different motion simulators developed in this work. The authors have, however, previously shown that closed-loop controllers can be successfully evolved in simulation when using ANN-based simulators [10].

generations of each evolution process, the best-performing controller (the controller with the highest fitness value in the population) was selected for further analysis on the real-world robot.

VII. Results and Discussion

The ANN-based and physics-based simulators developed in this work (Section V) were compared based on their accuracy, computational efficiency and performance of controllers evolved using each simulator when executed on the real-world robot. Results are now presented and discussed.

A. Accuracy of Simulators

In order to compare the accuracy of the simulators, predictions made by each simulator were compared to data collected from the real-world robot. This data was taken from the test set (Section IV), since the data in this set was not presented to the dynamic model during parameter optimization or the ANNs during their training phase. Using this unseen data would thus gauge the generalization capabilities of each of the simulators.

Findings are summarised in Table III. The data presented in this table is based on the absolute errors produced by each simulator, that is the absolute value of the difference between a prediction made by a given simulator and the corresponding expected value in the test set. These errors were calculated for all data points in the test set, and Table III shows the average absolute error, the minimum absolute error, maximum absolute error and the standard deviation in absolute errors for each simulator in predicting Δx, Δy and $\Delta\theta$. To give perspective on the magnitude of errors, the average magnitude of all expected values of Δx, Δy and $\Delta\theta$ in the test set is also provided in the table.

TABLE III

Accuracy of each motion simulator

		Δx (cm)	Δy (cm)	$\Delta\theta$ (degrees)
Ave Magnitude in Test Set		2.8940	4.7725	81.7871
ANN	Ave Error	0.4483	0.6241	3.0239
	Min Error	0.0053	0.0079	0.0096
	Max Error	5.2392	3.9437	22.5320
	Std Dev	0.5485	0.6014	3.3638
Dynamic Model	Ave Error	0.3467	0.3628	4.1985
	Min Error	0.0013	0.0025	0.0311
	Max Error	7.8446	4.0399	26.7574
	Std Dev	0.6586	0.4363	3.9652
Kinematic Model	Ave Error	1.0093	1.2788	13.2385
	Min Error	0.0008	0.0036	0.0586
	Max Error	19.0230	15.2045	50.2058
	Std Dev	1.7967	1.5346	10.1716

It can be seen from Table III that both the ANN simulators and the dynamic model were accurate at simulating the motion of the experimental robot, both having sub-cm average accuracy in predicting Δx and Δy. The kinematic model was much less accurate, as expected (Section V-A). The dynamic physics model was, on average, more accurate than the ANN-based simulators in predicting Δx and Δy, whereas the ANN simulators were more accurate at predicting $\Delta\theta$.

After an ANOVA Omnibus test revealed statistically significant differences between the average errors shown in Table III, a Tukey's HSD test [24] was performed using a 0.05 significance level. This test indicated that both the ANN simulators and dynamic model were statistically significantly more accurate than the kinematic model for the prediction of each of Δx, Δy and $\Delta\theta$ (all p-values were much smaller than 0.05). Differences observed in the accuracies of the ANN simulators and the dynamic model were not statistically significant for Δx ($p = 0.612$) and $\Delta\theta$ ($p = 0.138$), but were statistically significant in the prediction of Δy ($p = 0.013$). Thus, in summary, both the ANN-based simulators and the dynamic model dramatically outperformed the kinematic model, with comparable accuracies between the ANN simulators and the dynamic model, although the dynamic model was generally more accurate than the ANN simulators (statistically significantly more accurate in predicting Δy).

Considering that the ANN-based simulators were constructed without taking any of the physics related to the operation of the real-world robot into consideration (as was done by the physics-based simulators), the accuracy achieved by these ANN-based simulators is encouraging. The ANN-based simulators require no derivation of physics equations and no determination of physical parameters as was the case for the physics-based simulators, especially the dynamic model (Section V-A.2). The accuracy offered by these simulators thus clearly indicates that such ANN-based simulators can be comparable to physics-based simulators in terms of simulator accuracy, while arguably requiring much less human effort in their construction.

B. Computational Efficiency of Simulators

As was established in Section II, the computational efficiency of robotic simulators used in the ER process is of paramount importance. To evaluate the computational efficiency of each of the developed simulators, the time taken by each simulator to predict the motion of the experimental robot corresponding to 100 000 randomly-generated commands was measured. Tests for all the simulators were performed on the same computer and all the simulators were implemented in the same language (Java) to avoid any biasing of results.

TABLE IV

Time taken by each simulator to simulate 100 000 commands

	Running time (ms)
ANN	484
Dynamic Model	16700
Kinematic Model	24

Table IV shows the results obtained. It can be seen from this table that the kinematic model is extremely computationally efficient. This follows from the fact that this model can be expressed in the form of three closed-form equations (Section V-A.1). This computational efficiency, however, comes at the cost of accuracy (Section VII-A). Considering the level of accuracy offered by the ANN-based simulators (Section VII-A), the computational efficiency of these simulators is promising. These simulators execute roughly 30 times faster than the dynamic model, while still making predictions with accuracy comparable to that of the dynamic model.

C. Real-World Performance of Evolved Controllers

Since the ultimate goal of ER is to produce robotic controllers that can perform required tasks adequately when uploaded to real-world robots, the efficacy of the simulators developed in this work was evaluated by comparing the real-world behaviour produced by each evolved controller to that expected in simulation. This was achieved by comparing fitness values (equation (10)) produced by each evolved controller in simulation, to the fitness values produced by the same controller when executed on the real-world robot.

Results are provided in Table V. The data presented in this table is based on the absolute difference between the simulated fitness value of each of the 20 controllers evolved using each of the simulators and the corresponding real-world fitness value. The average, minimum and maximum of these differences are given for each simulator, based on the 20 controllers evolved using said simulator. Also shown in Table V are the number of controllers evolved using each simulator that successfully allowed the real-world robot to pass through all four blocks, and the number that caused the robot to miss one or two blocks respectively (no controller missed more than two blocks).

TABLE V
COMPARISON OF SIMULATORS BASED ON CORRESPONDENCE BETWEEN SIMULATED AND REAL-WORLD FITNESS OF EVOLVED CONTROLLERS

	ANN	Dynamic Model	Kinematic Model
Average fitness difference	328.91	393.35	745.81
Minimum fitness difference	13.90	9.52	21.33
Maximum fitness difference	2299.27	2267.64	2271.56
# All Blocks Visited	17	15	11
# 1 Block Missed	1	4	6
# 2 Blocks Missed	2	1	3

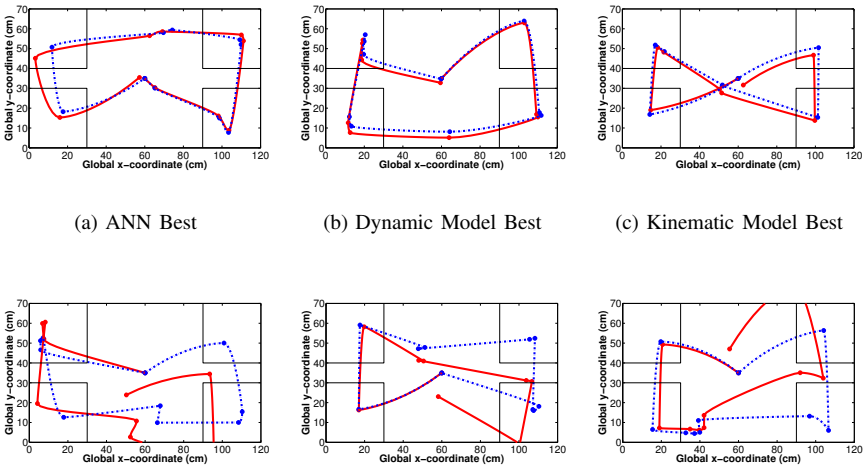
Figure 1 shows, for each of the developed simulators, the best and worst performing out of the 20 evolved controllers in terms of the absolute difference between simulated and real-world fitness values. In each case, these figures give the real-world path followed by the robot and that predicted by the relevant simulator. The points along each path indicate the positions of the robot as predicted by the simulators and the corresponding points on the real-world path followed by the robot. It should be noted that these points were interpolated to produce the provided figures and the lines in each figure are thus only an approximation to the actual path followed by the robot.

The results presented in Figure 1 indicate that all the simulators allowed for the successful evolution of controllers that could perform the required navigation task in simulation. However, it can clearly be seen that the evolved controllers transferred to the real-world robot with varying degrees of success. The best performing controllers evolved using each of the three simulators allowed the robot to perform the required task well in the real world, with the ANN and dynamic simulators allowing for slightly better performance on the real-world robot than the kinematic model: a cumulative error is visible in that the robot does not accurately return to its original position in Figure 1(c). The worst performing controllers can all be seen to produce real-world behaviours which differ quite drastically from the robotic behaviours expected in simulation. Since the controllers evolved in this study are open-loop (Section VI), a certain amount of deviation between simulated and real-world robotic behaviours is inevitable.

Although only minor differences are visible between the different simulators in Figure 1, Table V does indicate clear disparities between these simulators. It can be seen from this table that the ANN-based simulators evolved controllers which, on average, transferred more successfully to the real-world robot as compared to either of the physics-based simulators. This is evident from the fact that the average absolute difference between real-world and simulated fitness values is smallest for the ANN-based simulators and that 17 out of the 20 controllers evolved using the ANN-based simulators allowed the robot to pass through all four blocks successfully in the real world (the most of all the simulators). It is believed that the ANN-based simulators lead to the evolution of the most successful controllers due to the high accuracy offered by these simulators in predicting $\Delta\theta$ (Table III). Errors in robotic orientation prediction have a large influence on the correspondence between simulated and real-world paths: See, for example, Figure 1(d)-(f), where the real-world paths clearly deviate from the simulated paths due to errors in orientation prediction. It can further be seen from Table V that the dynamic model also produced controllers that performed relatively well on the real-world robot. The fact that both the ANN-based simulators and the dynamic physics model produced controllers that transferred relatively well to the real-world robot, follows directly from the fact that both these simulators were accurate in predicting the robot's motion (Section VII-A). Similarly, the relatively low accuracy offered by the kinematic physics model (Section VII-A) was likely the cause of poor transferability of controllers evolved using this simulator. This poor transferability is clear from the fact that barely one half (11 out of 20) of the controllers evolved using the kinematic model allowed for the real-world robot to successfully pass through all four blocks.

VIII. CONCLUSIONS AND FUTURE WORK

This study endeavoured to compare ANN-based and physics-based approaches to simulator development in ER. The results presented clearly indicate that ANNs do, at least for the robotic system considered in this study, offer a viable alternative to more commonly-used physics-based simulators. This follows from the fact that the ANN-based simulators were simpler to construct than the physics-based simulators, since no prior knowledge was required of the physics governing the robotic system being simulated in order to construct ANN-based simulators to operate in said system. In addition, the ANN-based simulators were shown to be computationally efficient (more efficient than the dynamic physics model by a factor of 30) and made predictions that were comparable to the more accurate physics model (the dynamic model). As a result of the accurate predictions



(a) ANN Best (b) Dynamic Model Best (c) Kinematic Model Best

(d) ANN Worst (e) Dynamic Model Worst (f) Kinematic Model Worst

Fig. 1. Best and worst performing controllers evolved using each motion simulator (dotted line = simulated path, solid line = real-world path)

made by the ANN-based simulators, these simulators also enabled the evolution of controllers which showed greater transferability to the real-world robot when compared to the physics-based simulators.

From what has been demonstrated in this study, the authors thus contend that ANN-based simulators may offer a valuable tool in ER. This study illustrated that such simulators can offer an excellent trade-off between accuracy and computational efficiency, both of which are vital in the ER process. Although encouraging results were obtained in this study, the suitability of ANN-based simulators for usage in ER can only truly be established through further investigation. Since the robotic platform considered in this study is relatively simple, more research is needed in order to determine how ANN-based simulators will compare to physics-based simulators for more involved robotic systems governed by more complex physical laws.

REFERENCES

[1] D. A. Sofge, M. A. Potter, M. D. Bugajska, and A. C. Schultz, "Challenges and opportunities of Evolutionary Robotics," in *Proceedings of the Second International Conference on Computational Intelligence, Robotics and Autonomous Systems*, 2003.

[2] D. Floreano and F. Mondada, "Evolution of homing navigation in a real mobile robot," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 26, no. 3, pp. 396–407, 1996.

[3] S. Koos, J. Mouret, and S. Doncieux, "The Transferability Approach: Crossing the reality gap in Evolutionary Robotics," *IEEE Transactions on Evolutionary Computation*, pp. 1–25, 2012.

[4] K. Glette, G. Klaus, J. C. Zagal, and J. Torresen, "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.

[5] J. C. Zagal and J. Ruiz-del Solar, "Combining simulation and reality in Evolutionary Robotics," *Journal of Intelligent and Robotic Systems*, vol. 50, no. 1, pp. 19–39, Mar. 2007.

[6] A. Boeing, "Design of a physics abstraction layer for improving the validity of evolved robot control simulations," Ph.D. dissertation, University of Western Australia, 2009.

[7] S. Koos, A. Cully, and J. Mouret, "Fast damage recovery in robotics with the T-Resilience algorithm," *Preprint version*, available: http://arxiv.org/abs/1302.0386, 2013.

[8] E. Drumwright, J. Hsu, N. Koenig, and D. Shell, "Extending Open Dynamics Engine for robotics simulation," in *Simulation, Modeling, and Programming for Autonomous Robots*. Springer Berlin Heidelberg, 2010, vol. 6472, pp. 38–50.

[9] C. J. Pretorius, M. C. du Plessis, and C. B. Cilliers, "A Neural Network-based kinematic and light-perception simulator for simple robotic evolution," in *IEEE Congress on Evolutionary Computation (CEC)*, 2010, pp. 1–8.

[10] ——, "Simulating robots without conventional physics: A Neural Network approach," *Journal of Intelligent and Robotic Systems*, vol. 71, no. 3-4, pp. 319–348, 2013.

[11] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: A survey," *Cognitive Processing*, vol. 12, no. 4, pp. 319–340, 2011.

[12] T. Lee, U. Nehmzow, and R. J. Hubbold, "Mobile robot simulation by means of acquired Neural Network models," in *Proceedings of the 12th European Simulation Multiconference on Simulation - Past, Present and Future*, 1998, pp. 465–469.

[13] O. Nelles, *Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models*. Springer Berlin Heidelberg, 2010.

[14] "Khepera III," accessed June 2013. Available: http://www.k-team.com/mobile-robotics-products/khepera-iii.

[15] "Wii Official Site at Nintendo," accessed June 2013. Available: http://www.nintendo.com/wii.

[16] "Johnny Chung Lee - Projects - Wii," accessed July 2013. Available: http://johnnylee.net/projects/wii/.

[17] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. Cambridge: MIT Press, 2004.

[18] J. Laut, "A dynamic parameter identification method for migrating control strategies between heterogeneous wheeled mobile robots," Master's thesis, Worcester Polytechnic Institute, 2011.

[19] R. Burden and J. Faires, *Numerical Analysis*, 8th ed. London: Thomson Brooks/Cole, 2005.

[20] A. P. Engelbrecht, *Computational Intelligence: An Introduction*, 2nd ed. West Sussex: Wiley, 2007.

[21] C. J. Pretorius, "Artificial Neural Networks as simulators for behavioural evolution in Evolutionary Robotics," Master's thesis, Nelson Mandela Metropolitan University, 2010.

[22] K. Levenberg, "A method for the solution of certain problems in least squares," *Quarterly of Applied Mathematics*, vol. 5, pp. 164–168, 1944.

[23] "Encog Machine Learning Framework — Heaton Research," accessed July 2013. Available: http://www.heatonresearch.com/encog.

[24] W. J. Steinberg, *Statistics Alive!*, V. Knight, Ed. Thousand Oaks: Sage Publications, 2010.