2015 IEEE Symposium Series on Computational Intelligence

Evolving Snake Robot Controllers using Artificial Neural Networks as an Alternative to a Physics-Based Simulator

Grant W. Woodford
Department of Computing Sciences
Nelson Mandela Metropolitan University
Port Elizabeth, South Africa
Email: grant.woodford@nmmu.ac.za

Mathys C. du Plessis
Department of Computing Sciences
Nelson Mandela Metropolitan University
Port Elizabeth, South Africa
Email: mc.duplessis@nmmu.ac.za

Christiaan J. Pretorius
Department of Mathematics and Applied Mathematics
Nelson Mandela Metropolitan University
Port Elizabeth, South Africa
Email: cpretorius@nmmu.ac.za

***Abstract*—Traditional simulators can be complex, time-consuming and require specialized knowledge to develop while still being unable to adequately model reality. Artificial Neural Networks (ANNs) can be trained to simulate real-world robots and therefore serve as an alternative to traditional approaches of robot simulation during the Evolutionary Robotics (ER) process. ANN-based simulators require little specialized knowledge and can automatically incorporate many real-world peculiarities. This paper reports a simulator that consisted of ANNs which were trained to predict changes in the position of a real-world snake-like robot. Navigational behaviours were evolved in simulation and subsequently verified on the real-world robot. This paper demonstrated that ANNs are a viable alternative to traditional simulators for evolving controllers for snake-like robots.**

I. Introduction

The field of Evolutionary Robotics (ER) seeks to automate the development of intelligent control structures for robotic systems using Evolutionary Computing approaches [1]. The manual development of robot controllers becomes infeasible as robots, environments and tasks increase in complexity [2]. ER has been shown to automatically evolve many robot behaviours, such as path following, inverted pendulum stabilization, light following and obstacle avoidance [3], [4].

In ER, many controllers are evaluated and their relative performances quantified in order to evolve better controllers. The evaluation of many controllers on a real-world robot is time-consuming and can damage hardware [5]. These issues can be overcome through the use of simulators as an alternative to real-world evaluations [2]. Traditional simulators are physics-based or modelled on empirically collected data [3]. These simulators can be time-consuming and complicated to develop because it may require the use of complex physics-based models and/or the gathering of large amounts of experimental data.

It has been shown that alternatively, simulators can be constructed using ANNs that are trained to predict robot behaviours using experimentally collected data [3], [4], [6], [7]. In this paper, ANN-based simulators will simply be referred to as Simulator Neural Networks (SNNs). The use of SNNs have shown much promise as an alternative to traditional approaches to simulation during the ER process [4]. SNNs have been shown to be computationally efficient, require little specialized knowledge, possess good prediction accuracy, noise-tolerance and generalization abilities in modelling of certain robot phenomena [7]. Previous work has mainly focused on simple robots, therefore this paper aims to investigate the use of SNNs during the ER process on a complex robot.

This paper is structured as follows: Section II-A describes the ER process while Section II-B addresses the use of simulators during the ER process. Related work on snake-like robots is discussed (Section II-C) while the experimental robot used in this paper is considered (Section III). The simulator developed in this paper is proposed in Section IV-A. The robotic controller and method of behavioural tracking used in this paper are discussed in Sections IV-B and IV-C respectively. The experimental procedure followed for simulator training and the subsequent validation of the proposed simulator are addressed in Sections V-A and V-B. The results determined that the proposed simulator is indeed viable, with the simulator's accuracy and real-world validation experiments presented in Sections VI-A and VI-B respectively. These results are discussed in Section VII and finally conclusions are drawn and possible future work is discussed (Section VIII).

II. Background

A. Evolutionary Robotics

In ER, robotic controllers are evolved to develop behaviours using Genetic Algorithms (GA) [1]. To develop the appropriate behaviours, a population of encoded candidate controllers is created. Each controller's fitness relative to each other is determined based on how well the controller exhibits the target behaviour on the experimental robot or in simulation, after which a new generation of controllers is created to replace the previous generation. The new generation is generated using reproduction operators (crossover and mutation) between the controllers of the previous generation.

Controllers with a higher fitness have a greater probability of being chosen to produce offspring for the new generation.

978-1-4799-7560-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SSCI.2015.47

During the crossover process, genetic material between parent controllers are combined and passed onto the new generation. The mutation operator can then be applied, causing small random perturbations in controllers which allows for a more diverse controller population. This process is repeated for a large number of generations and controllers ideally converge towards the target behaviour. The end result of the ER process is an optimized controller that is validated on a real-world robot.

A major issue during the ER process is the evaluation of a large number of controllers for fitness determinations (Section I). Determining the fitnesses of many controllers on a real-world robot would be time-consuming and financially costly. Controller fitness evaluations can therefore be done in simulation to speed up the process.

B. Evolution in Simulation

Simulators are able to overcome issues inherent in real-world fitness evaluations. Controller evolution can explore the search space more rapidly in simulation than it would be possible in reality [15]. However, as previously mentioned, the design and construction of traditional physics-based simulators can become time-consuming and complicated [7]. Much research in ER is concerned with overcoming the difficulties inherent in utilizing simulators effectively [16]. Challenges in simulator design can be inaccuracies and/or over-simplification in the modelling of reality [7].

Over-simplification or inaccuracies in simulations may result in controllers relying on peculiarities that exist only in simulation but are non-existent in reality which results in behaviours evolved in simulation not transferring well to reality, commonly referred to as the *reality gap* problem [17]. Over-simplification can be avoided through the use of highly accurate simulators. However, even highly accurate simulators cannot perfectly model reality and will inevitably contain inaccuracies [18]. Additionally, highly accurate simulators are often computationally expensive [19]. Simulators ideally need to provide highly accurate representations of reality whilst not being too computationally expensive to use.

There are currently few alternatives to physics-based simulators in use by ER researchers. The notion of using SNNs as an alternative to traditional simulators has been investigated by few researchers [6], [20]. As previously mentioned (Section I), SNNs are computationally efficient, accurate, relatively simple to construct and potentially provide an effective alternative to physics-based approaches. The training of SNNs requires the evaluation of many randomized behaviours on a real-world robot and the collection of this behavioural data. This behavioural data is then used to train SNNs to predict robot behaviours.

SNNs have been effectively used during the ER process for tasks such as path following, obstacle avoidance, light approaching behaviour and inverted pendulum stabilization [3], [4], [6], [7]. Researchers have also shown that SNNs can simulate the dynamics of the pendulum swing-up problem [20].

C. Snake Robots

The ER process for snake-like robots is mostly carried out in simulation using physics-based approaches to estimate controller fitness [8]–[11]. A physics-based simulator and GA approach has been used for shape transformation planning to achieve stable, smooth transitions between snake locomotion modes [12]. Additionally, modular robotics research has been conducted using snake-like configurations and a GA to evolve controllers in a physics-based simulator [13], [14].

Snake-like robots are constructed by chaining together several independent actuators, each actuator commonly having one degree of freedom [13], [21], [22]. These snake robots are capable of both biological and non-biologically inspired locomotion modes.

Biologically inspired snake locomotion modes can be broadly classified into lateral undulation, slide-pushing, rectilinear motion, concertina, side-winding and various other forms [23]. Lateral undulation is the most common form where all parts of the body move in a wave-like pattern [23]. However, lateral undulation is not suited to smooth, low friction surfaces [24]. Side-winding locomotion has a sine-like wave while maintaining only two static points of contact with the ground at any time. Side-winding is more suited for low friction surfaces [24]. There are also non-biologically inspired locomotion modes, namely rolling where the snake rolls side over side and flapping motions where the robot flaps both of its ends across the ground [23].

One method of generating the above mentioned locomotion modes is the use of parametrized equations which generate the appropriate joint angles on the robot. Melo *et al.* [21] have made use of parametrized Equations (1) and (2) which are based on sinusoidal motion on two axes. These equations define $\phi(n,t)$ as the angle of the nth joint at discrete time steps t. Snake joints were numbered, starting at zero from front to back, with evenly numbered joints moving the snake laterally and odd ones moving the snake vertically. If the snake rolled on its side, then the evenly numbered joints moved vertically and oddly numbered joints moved laterally which similarly switched the angles used by Equation (1). The terms $O_{lateral}$ and $O_{vertical}$ are the offsets of the lateral and vertical joints respectively. The offset terms define the central angle value of their respective waves. Terms $A_{lateral}$ and $A_{vertical}$ refer to the wave amplitudes of the lateral and vertical joint angles respectively. The parameters ω and γ are respectively the spatial and temporal components of the sine wave moving through the robot. The α term is the phase offset between the lateral and vertical sine waves.

$$\phi(n,t) = \begin{cases} O_{lateral} + A_{lateral} \cdot sin(\theta + \alpha), n = even \\ O_{vertical} + A_{vertical} \cdot sin(\theta), n = otherwise \end{cases} \quad (1)$$

$$\theta = \omega n + \gamma t \quad (2)$$

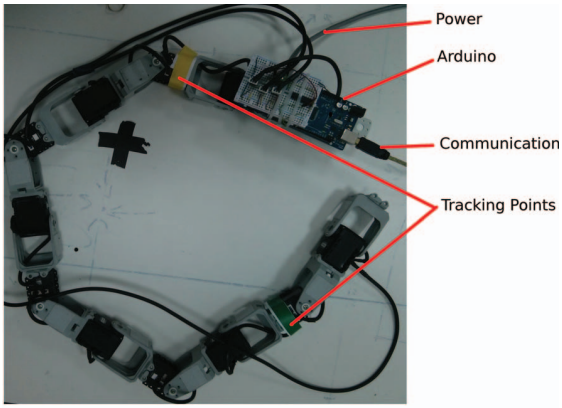

Fig. 1. Robot Morphology

III. EXPERIMENTAL ROBOT

A custom designed snake-like robot was developed for the experimental work. Similarly constructed snake-like robots have also been used by other researchers [14], [22].

This robot was chosen due to its simple construction and relatively low cost. The morphology of the robot used for experimental work is shown in Figure 1. An array of twelve Dynamixel AX-12 servo motors joined together with links formed the robot's body. Evenly numbered joints moved the robot laterally while the odd joints moved the robot vertically. The length of the robot was 114cm and its width and height were 5cm. The servos were controlled by an Arduino Mega micro-controller which received joint angles from a computer using serial communication.

The servos were powered using a tethered connection to the robot. A tethered approach was chosen due to the extra weight batteries would add and to eliminate the need for monitoring battery power levels. Near each end of the robot was a yellow or green tracking marker which was used for camera based tracking.

No snake-like skins or mechanisms such as wheels were used to allow for the directional friction required for forward locomotion seen in many biological snakes. This resulted in the robot having difficulty with forward locomotion and having to either side-wind, strafe laterally, perform helix-like rolling or flapping motions. The robot was not fitted with any sensors.

IV. IMPLEMENTATION DETAILS OF SIMULATOR AND EXPERIMENTAL SETUP

The design and construction of the developed simulator is addressed in Section IV-A. Details of the controller used and method of behavioural tracking are discussed in Sections IV-B and IV-C respectively.

A. Proposed Approach for using Simulator Neural Networks

A set of twelve joint angles represented a single command. When a command was sent to the robot, all of the robot's joints simultaneously positioned to the assigned angles. A cycle consisted of thirteen sequential commands which when evaluated, started and ended on the same joint angle set. During the evaluation of a cycle, commands were sent sequentially to the robot and upon reaching all the assigned joint angles of the given command, the robot moved onto the next command.

Equation (1) is used to generate a sequence of commands that perform cyclical behaviours when evaluated on a robot. The robot's change in position after the evaluation of a given cycle depends on the parameter settings used by Equations (1) and (2). SNNs take as input those parameter settings and subsequently predict the change in the robot's position on the planar operating surface at the end of the given cycle. The simulator also predicts whether cycles will fail and cause the robot to tip over, roll, reach torque limits or collide with itself.

The creation of SNNs and controllers using the ER process is achieved as follows:

1) Parameter settings for Equations (1) and (2) are randomly generated using a uniform distribution and used to generate cycles that are performed on the real-world robot. As a result of these commands, the robot moves around the environment and data is collected using motion tracking techniques.
2) When sufficient data has been collected, it is used to train SNNs to predict the real-world robot's behaviours. Data collection is concluded when the trained SNNs are able to evolve adequately transferable controllers that complete the goal task.
3) The trained SNNs are used to determine the fitness of candidate controllers during controller evolution.
4) At the end of the ER process, the fittest controller in simulation is validated using the real-world robot.

SNNs were trained to predict the change in the position of the tracked markers on the robot for a given cycle. The robot had two local coordinate systems with the origins located at the centre of each tracked marker and y directions taken from the yellow to green tracked markers. The change in the x and y-coordinates for the yellow tracked marker for a given cycle was represented by Δx_1 and Δy_1 respectively. Similarly, the changes in the x and y-coordinates for the green marker were represented by Δx_2 and Δy_2 respectively. The simulator also predicted whether or not the robot remained upright and did not collide with itself or reach any torque limits during a given cycle.

The simulator developed in this study consisted of five separate Feed Forward Neural Networks (FFNNs) (Figure 2), one for each of the x and y directions for each tracked marker and another to predict failed cycles. A previous study determined that separate FFNNs could produce more accurate results than a single FFNN [6]. Each FFNN took as input the values of $A_{lateral}$, $A_{vertical}$, α and ω that were used by Equation 1 to generate the cycle. The offset and γ terms were kept constant and not used as inputs. Sigmoid activation functions were used for all FFNN neurons and each FFNN had a single hidden layer of 100 neurons.

In order to evolve robust controllers that were able to cross the *reality gap* (Section II-B), noise was injected into the ER process during controller evaluations. Controllers were

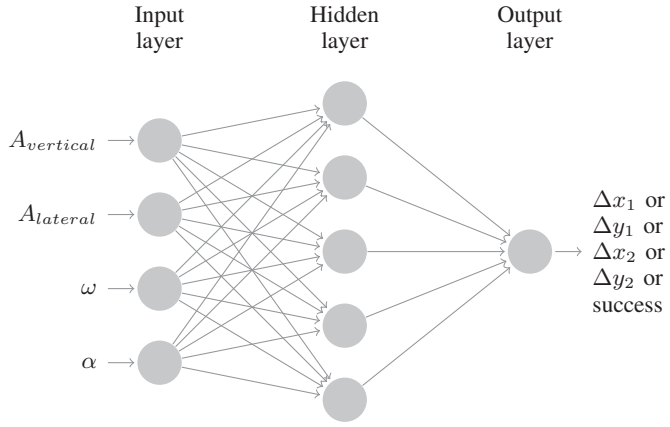

Fig. 2. Simulator Neural Networks

evaluated ten times in simulation and the average fitness was used. The distribution used for noise injection was Gaussian with a mean of zero and standard deviations of 10cm and 5cm for the $\Delta x_{1,2}$ and $\Delta y_{1,2}$ displacements respectively. These standard deviations were based on the observed training data errors.

B. Robot Controller

A controller consisted of a sequential list of four different parameters settings that were used to generate cycles using Equations (1) and (2). Each cycle in the list was repeated up to four iterations when evaluated. It was observed through manual experimentation that the chosen controller morphology is more than sufficient to accomplish the required behaviours. A single controller could consist of between four to sixteen cycles.

The γ term determined the number of commands contained in a given cycle. The set of joint angles for all cycles had to begin and end on the same set of angles which is exactly one period of the sine wave. The time-steps of t for every cycle went from zero to the number of commands per cycle plus one. The commands per cycle were fixed at thirteen which required that the γ term remained constant at $2\pi/12$. The offsets $O_{lateral}$ and $O_{vertical}$ determined the central angle value for the wave which was assumed to remain constant at zero. Non-zero offsets typically help steer the robot in a particular direction. Amplitudes $A_{lateral}$ and $A_{vertical}$ had a range of between 0 and $\pi/2$. The α and ω parameters were within the sine function, therefore their values ranged from 0 to 2π. The reason the number of commands per cycle, the offsets and γ terms were chosen to remain constant was to reduce the controller search space which in turn reduced the amount of training data the simulator required to perform adequately.

The vertical amplitude $A_{vertical}$ was chosen so that it was less than the lateral amplitude $A_{lateral}$ in order to reduce the number of cycles that failed to remain upright. Even with this restriction in place, many parameter sets could result in the robot tipping over or rolling. Ensuring that the robot always remained upright and stable during evaluations was important for the simulator predictions. The simulator was trained to only predict behaviours of upright cycles.

The angular velocity of the robot's joint movements was kept constant. The angular velocity was experimentally chosen to reduce slippage on the smooth operating surface. Depending on the locomotion mode, the degree of slippage varied greatly. Due to the flexibility of Equation 1, many different locomotion modes were possible, even modes that resulted in collisions with the robot itself or caused joint torque limits to be reached.

C. Motion Tracking

A roof-mounted camera was used to track robot behaviours, namely the change in position of the tracked markers on the robot for a given cycle. A camera based tracking approach was chosen over manual data acquisition methods in order to speed up the process and eliminate human error. An open source computer vision library called OpenCV [25] was used for the camera based tracking software.

Images from the tracking camera were used to locate the pixel coordinates of the tracked markers. The centre of each marker was determined and converted into real-world coordinates. Calibration techniques were employed and distortions were removed from the captured images before use. The automated tracking process was not used to identify if the robot tipped over, rolled, collided with itself or if the torque limits were reached, therefore these behaviours were manually noted during data acquisition.

V. EXPERIMENTAL PROCEDURE FOR VALIDATING SIMULATOR

Experimental data from the real-world robot was acquired and used to train the simulator (Section V-A). Once trained, the simulator was used to evolve robotic controllers to perform a navigational task on the real-world robot (Section V-B).

A. Simulator Training

To obtain sufficient data for SNN training, randomly generated cycles were evaluated on the real-world robot and behavioural data was collected. The change in position for each cycle was recorded by the roof-mounted camera. The robot operated on a working surface of dimensions 2.7m by 1.85m.

Training data was experimentally acquired from 400 randomly generated cycles. This training data was used to train an initial simulator and the worst twenty training data cycles that had the largest difference between their expected versus simulated displacements were subsequently selected for each of the SNNs. These poor performers were re-evaluated and each one used to generate four additional cycles with noise, generating 400 additional training data cycles. The SNNs were then re-trained utilizing all training data. A bidirectional approach was taken to assist the simulator predictions to become more accurate and stable. These SNNs were used to evolve controllers during the ER process and twenty of the worst performing cycles from these controllers were taken and used to generate a further 100 training data patterns. The

final SNNs were then trained using all available training data generated.

The SNNs were implemented using an open source machine learning library called Encog [26]. SNNs were trained using Resilient Backpropagation for 12000 iterations or just before over-fitting occurred. The sizes of the training data set and verification data of the stable cycles were 720 and 76 respectively. The sizes of the training data set and verification data set of both the stable and unstable cycles were 820 and 80 respectively. The data for the stable cycles were used to train the SNNs to predict the changes in the x and y direction of both tracked markers for a given cycle. The data for the stable and unstable cycles were used to train another SNN to predict whether a given cycle would be stable or unstable.

B. Validation Experiment

Due to this investigation being an initial proof-of-concept prototype, the navigational task was deliberately chosen to be simple in order to investigate the viability of the approach. The required navigational behaviour is shown in Figures 7 to 9. Three squares, each 40cm wide were placed in three quadrants of the operating surface and placed 20cm apart from each other. The robot was placed with the yellow marker on the origin and facing eastwards. The task was considered completed when the robot traversed all the blocks in a specific order (top right, bottom left, top left).

During the ER process, controllers were evaluated in simulation, generating a list of points which represented the simulated path. The fitness values assigned to each controller were determined using Algorithm 1. This algorithm took as input, the simulated path and the list of goal regions. The fitness was calculated based on the number of the goal regions reached in their specified order. A controller's fitness was penalized for containing positions outside the bounds of the working surface. If a position was reached from an unstable cycle, the fitness was also penalized. The fitness was further penalized for each goal region not reached.

Controllers consisted of encoded parameters for the variable terms in Equations (1) and (2). The controller evolution settings are given in Table I. An initial population of 400 controllers was generated randomly from a uniform distribution. During uniform cross-over operations, two parents were selected using tournament selection (using a tournament size of 20 parents). Child controllers were comprised of approximately 20% genetic material from one parent and 80% from the other. The probability that controller parameter values were mutated by a random amount was 10%. The ER process proceeded until the fittest controller was sufficiently able to complete the task in simulation. Upon completion, the final controller was evaluated on the real-world robot.

Algorithm 1: Controller fitness evaluation

Data:
$positions \leftarrow$ Simulated path list
$goalpoints \leftarrow$ List of goal regions
Result: Fitness of controller
$sum \leftarrow 0$
$curGP \leftarrow 0$ ▷ Current goal point
$penalty \leftarrow$ a large constant
for *each position in positions* **do**
  if *position out of bounds* **then**
    $sum \leftarrow sum + penalty$
  end
  if *position reached* $goalpoints[curGP]$ **then**
    $curGP \leftarrow curGP + 1$
  end
  if *position reached from failed cycle* **then**
    $sum \leftarrow sum + penalty$
  end
end
$missedGoalpoints \leftarrow length(goalpoints) - curGP - 1$
$sum \leftarrow sum + penalty \times missedGoalpoints$
$fitness \leftarrow 1/sum$
return *fitness*

TABLE I
PARAMETERS FOR CONTROLLER EVOLUTION

Population Size	400
Initialization	Random from a uniform distribution
Selection	Tournament (size 20)
Crossover Method	Uniform
Mutation Probability	10%
Mutation Method	Random Component Perturbation

VI. RESULTS

Section VI-A discusses the accuracy of the simulator developed in Section V by analysing the predicted versus expected change in the robot's position using the verification data set. Section VI-B presents the results of the optimized controllers run on a real-world robot and compares the behaviours to simulation.

A. Simulator Accuracy

Figures 3 to 6 demonstrate the predicted versus expected change in position of the tracked markers for the verification data set. Every dot represents the simulated versus real-world change in position of the robot for the specific coordinate axis and for the given verification cycle. The verification data set was not presented to the simulator during training and was therefore used to determine the accuracy and generalization abilities of the trained simulator.

For Figures 3 to 6, linear regression lines were fitted. All the y-intercepts of the regression lines were close to zero. The R-squared values for Figures 3 and 5 were 0.84 and 0.75 respectively. This indicted that the simulator could adequately model movement along the x direction of the robot. The regression line slopes were 0.85 and 0.9 for Figures 3 and 5 respectively and both y-intercepts close to zero, which indicated that the simulator modelled the real-world fairly well. The R-squared values, 0.17 and 0.05 for Figures 4 and 6 respectively indicted that the simulator could not adequately model movement along the y direction of the robot. The slopes for Figures 4 and 6 were 0.28 and 0.25 respectively, which indicated that the simulator and real-world did not closely relate for those movements.

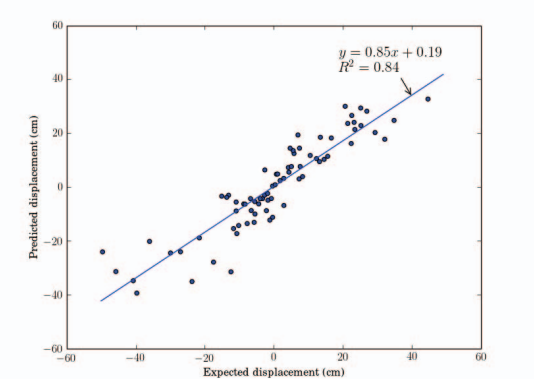

Fig. 3. Yellow tracking point's predicted versus expected displacement in the x-direction

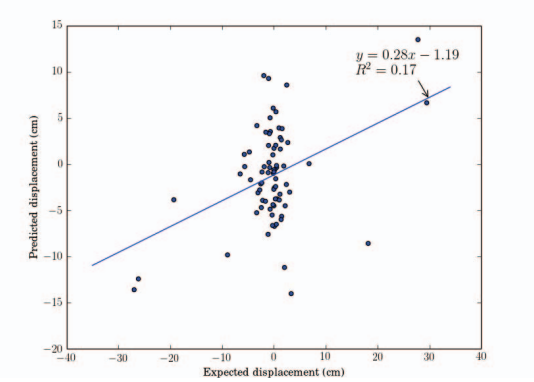

Fig. 4. Yellow tracking point's predicted versus expected displacement in the y-direction

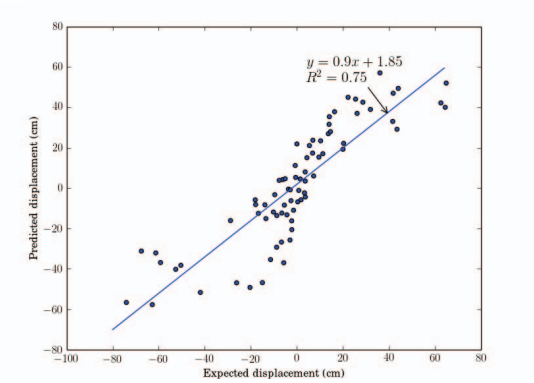

Fig. 5. Green tracking point's predicted versus expected displacement in the x-direction

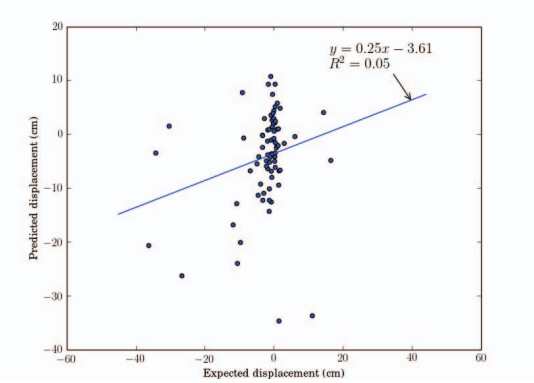

Fig. 6. Green tracking point's predicted versus expected displacement in the y-direction

The simulator was trained to predict the success or failure of any given cycle. The patterns used to verify the success or failure of a cycle contained 19 failed cycles and 63 successful cycles. The percentage of verification patterns that the simulator correctly predicted to succeed or fail was 85%. The simulator relied greatly on only predicting behaviours of successful cycles which meant that fitness estimations were inaccurate for controllers that contained any failed cycles. The percentage of verification patterns that the simulator correctly predicted to fail was 58% and the percentage of cycles that were correctly predicted to succeed was 94%.

Predictions for the robot's movement in the y direction were a great deal less accurate than that along the robot's x direction. The expected values for the robot's y directions tended to be close to zero. This was possibly due to the smooth operating surface and lack of directional friction properties seen in biological snakes or snake-like robots using wheels. These results possibly indicate that much of the movement in the y direction were close to random and that the main strategy used by the robot for locomotion was in the x direction.

The results provided a reliable indication as to the accuracy of the developed simulator and whether enough training patterns had been collected. Predicting the robot's change in positions is indeed possible which is significant considering the complexity of the robotic system.

B. Validation Results

The ideal paths obtained by validation controllers in simulation were compared to their real-world paths. These results are shown in Figures 7 to 9. The dashed lines represent the simulated path of the validation controller and the solid lines represent the real-world paths followed by the robot. The location of the robot was determined to be midway between the tracked markers. For each validation controller, three real-world evaluations were performed.

There were twelve independent trial runs, of which only the best three were presented in this paper. Three trials were excluded due to the robot moving out of bounds of the operating surface. Another three trials were excluded due to the robot rolling during transitions between cycles. One trial contained a failed cycle and two trials were left out due to poor transference to the real-world robot.

Evaluated cycles exhibited many types of locomotion modes, such as lateral strafing, side-winding, helical movements, rolling, flapping and many others. Certain behaviours, such as flapping were observed to be unable to transfer well to reality whereas other behaviours such as side-winding showed better transference to reality. Transitions between different cycles could also cause stability and accuracy problems.

The controller search space was large and often contained regions where small changes to a controller could result in large changes in behaviour. For each trial, the number of cycles used, percentage of goal regions reached, average displacement between the final positions in simulation and reality and lastly the number of controller evolution generations of the validation controllers are given in Table II. Even though

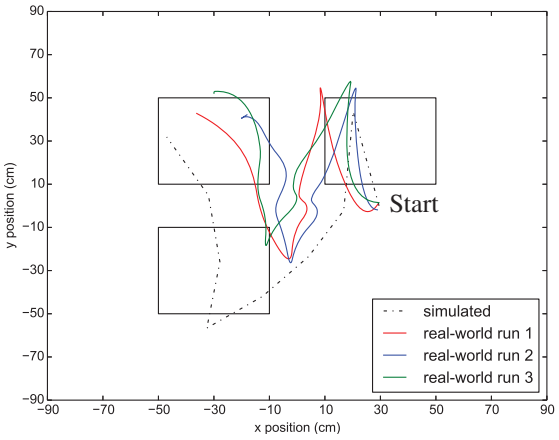

Fig. 7. Experimental run, trial 1

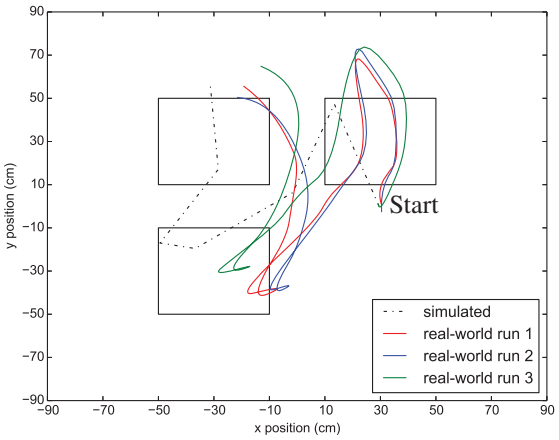

Fig. 8. Experimental run, trial 2

the navigational task was relatively simple, controllers needed many cycles to complete the task.

TABLE II
TRIAL RUN DETAILS

Trial number	Cycles	Goal regions reached	Final displacement	Controller generations
Trial 1	8	89%	21cm	200
Trial 2	6	78%	16cm	100
Trial 3	8	100%	53cm	100

The simulated and real-world controller behaviours were similar for the three trials (Figures 7 to 9). Inaccuracies between the simulator and real-world could be due to many reasons, such as path divergence over time due to an accumulation of errors. The simulator was simplified and did not predict changes in position due to transitions between cycles.

The first cycle in trial 1 (Figure 7) moved the robot into the initial goal region. The second cycle reversed the robot's direction towards origin of the working space. The slight shift eastwards around the origin was the result of the robot's shift in position due to switching to the third cycle. The third cycle was repeated three times and showed poor transference to reality which was possibly due to the lack of training data

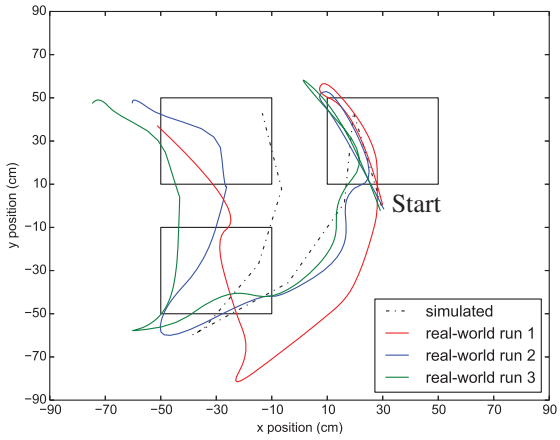

Fig. 9. Experimental run, trial 3

patterns near that region of the cycle search space. The last distinct cycle was repeated three times and brought the robot up towards the last goal region.

The first cycle in trial 2 (Figure 8) moved the robot into the initial goal region and overshot towards the right when compared to the predicted path. The next two cycles moved the robot towards and mostly into the second goal region, only one of the real-world runs missed this goal region. The fourth cycle which was supposed to move the robot westward however resulted in the real-world robot shifting position in place. The last two cycles formed a curved path towards the third goal region which closely matched the predicted behaviour.

The first cycle in trial 3 (Figure 9) moved the robot over the first goal region and overshot it slightly towards the left. The next three identical cycles brought the robot towards and just below the second goal region. The fifth cycle moved the robot slightly closer towards the second goal region and the last three cycles moved the robot over the second and third goal regions.

VII. DISCUSSION

The use of SNNs to estimate controller fitness during the ER process is indeed feasible. It was possible to use SNNs to evolve robot behaviours for a complex robot performing a simple navigational task. A direct comparison between the effectiveness of SNNs and physics-based approaches was infeasible for this limited investigation.

From the results shown in Section VI-A, it was observed that changes in position of the yellow marker could be modelled more accurately than that of the green marker. This could be due to the increased friction caused by the added weight of the Arduino and tether. The increased friction could cause the robot to exhibit less slippage. An environment or robot with different frictional properties could possibly yield better results.

The developed simulator was able to predict the general behaviour of the robot. It was also significant that behaviours could be evolved for a complex robot to navigate blindly in its environment with no feedback. This was noteworthy

considering the many sources of errors. Possible sources of errors included inaccuracies in motion tracking, inconsistencies in motor function, slippage on the operating surface and errors due to the simplifying assumptions of the simulator. Poor transference could also be due to insufficient training patterns around certain portions of the cycle search space. Much work needs to be done devising better training data sampling strategies.

The complexities in the search space were significant, where small changes in cycle parameter settings could result in significant differences in behaviour. Cycles that did not transfer well to reality were noted and similar training data was generated which tended to improve transferability overall.

The collection of sufficient behavioural data was a time-consuming process and it was infeasible to get complete coverage of the search space. However, the development and tuning of a physics-based engine could become equally, if not more, time-consuming and would require specialized knowledge about the dynamics of the robotic system.

VIII. CONCLUSIONS AND FUTURE WORK

For simple robots, SNNs have been shown to perform well as an alternative to physics-based approaches during the ER process. Research into using SNNs during the ER process for more complex robots is largely unexplored. The difficulty in developing adequate simulators increases as the complexity of the robots, environments and tasks increase. The SNN approach provides a simple way for researchers to construct models of reality that can be used to evolve behaviours that would be too costly to develop through real-world evaluations.

This paper demonstrated that SNNs are indeed viable for evolving behaviours for snake-like robots during the ER process. It was observed that certain locomotion modes were more stable and transferable than others. It is significant that this research was able to develop navigational controllers in simulation using a GA without the need for developing a physics-based simulation.

Future work could include, training separate SNNs for each locomotion mode could provide more accurate results. Future research could also include investigating the performance of various ANN morphologies, alternative training data sampling strategies and exploring a wider variety of tasks and robots.

ACKNOWLEDGMENT

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged (UID number: 89526). Opinions expressed and conclusions arrived at, are those of the authors and are not necessarily to be attributed to the NRF.

REFERENCES

[1] D. K. Pratihar, "Evolutionary robotics - A review," *Sadhana*, vol. 28, no. 6, pp. 999–1009, 2003.

[2] I. Harvey, P. Husbands, D. Cliff, and Others, *Issues in evolutionary robotics*. School of Cognitive and Computing Sciences, University of Sussex, 1992.

[3] C. J. Pretorius, M. C. du Plessis, and C. B. Cilliers, "Simulating robots without conventional physics: A neural network approach," *Journal of Intelligent & Robotic Systems*, vol. 71, no. 3-4, pp. 319–348, 2013.

[4] C. J. Pretorius, M. C. du Plessis, and J. W. Gonsalves, "A comparison of neural networks and physics models as motion simulators for simple robotic evolution," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2014, pp. 2793–2800.

[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] C. J. Pretorius, M. C. du Plessis, and C. B. Cilliers, "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*. ACM, 2009, pp. 170–178.

[7] C. J. Pretorius, "Artificial Neural Networks as simulators for behavioural evolution in evolutionary robotics," Masters thesis, Nelson Mandela Metropolitan University, 2010.

[8] S. Hasanzadeh and A. Akbarzadeh, "Development of a new spinning gait for a planar snake robot using central pattern generators," *Intelligent Service Robotics*, vol. 6, no. 2, pp. 109–120, 2013.

[9] S. Hasanzadeh and A. A. Tootoonchi, "Ground adaptive and optimized locomotion of snake robot moving with a novel gait," *Autonomous Robots*, vol. 28, no. 4, pp. 457–470, 2010.

[10] K. Inoue, S. Ma, and C. Jin, "Optimizationof CPG-network for decentralized control of a snake-like robot," in *Robotics and Biomimetics (ROBIO). 2005 IEEE International Conference on*. IEEE, 2005, pp. 730–735.

[11] J.-K. Ryu, N. Y. Chong, B. J. You, and H. I. Christensen, "Locomotion of snake-like robots using adaptive neural oscillators," *Intelligent Service Robotics*, vol. 3, no. 1, pp. 1–10, 2010.

[12] T. Kamegawa, F. Matsuno, and R. Chatterjee, "Proposition of twisting mode of locomotion and ga based motion planning for transition of locomotion modes of 3-dimensional snake-like robot," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 2. IEEE, 2002, pp. 1507–1512.

[13] A. Kamimura, H. Kurokawa, E. Yoshida, S. Murata, K. Tomita, and S. Kokaji, "Automatic locomotion design and experiments for a modular robotic system," *Mechatronics, IEEE/ASME Transactions on*, vol. 10, no. 3, pp. 314–325, 2005.

[14] S. Murata and H. Kurokawa, "Self-reconfigurable robots," *Robotics & Automation Magazine, IEEE*, vol. 14, no. 1, pp. 71–78, 2007.

[15] H. H. Lund and O. Miglino, "From simulated to real robots," in *Evolutionary Computation, Proceedings of IEEE International Conference*. IEEE, 1996, pp. 362–365.

[16] J. C. Bongard, *Evolutionary robotics*. ACM, 2013, vol. 56, no. 8.

[17] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *Advances in artificial life*. Springer, 1995, pp. 704–720.

[18] D. Floreano, P. Husbands, and S. Nolfi, "Evolutionary robotics," *Springer handbook of robotics*, pp. 1423–1451, 2008.

[19] O. Miglino, K. Nafasi, and C. Taylor, "Selection for wandering behavior in a small robot," *Artificial Life*, vol. 2, no. 1, pp. 101–116, 1994.

[20] S. Nakamura, R. Saegusa, and S. Hashimoto, "A hybrid learning strategy for real hardware of swing-up pendulum," *Journal of Advanced Computational Intelligence & Intelligent Informatics (JACIII)*, vol. 11, no. 8, 2007.

[21] K. Melo, M. Hernandez, and D. Gonzalez, "Parameterized space conditions for the definition of locomotion modes in modular snake robots," in *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*. IEEE, 2012, pp. 2032–2038.

[22] K. Melo, L. Paez, and C. Parra, "Indoor and outdoor parametrized gait execution with modular snake robots," in *2012 IEEE International Conference on Robotics and Automation*.

[23] K. J. Dowling, "Limbless locomotion: learning to crawl with a snake robot," Ph.D. dissertation, NASA, 1996.

[24] O. Shmakov, "Snakelike robots locomotions control," *Mechatronics–Foundations and Applications*, 2006.

[25] Itseez, "Official OpenCV Site," http://opencv.org, accessed: July 2015.

[26] Heaton Research, "Official Encog Site," http://www.heatonresearch.com/encog, accessed: July 2015.