A Neural Network Ultrasonic Sensor Simulator for Evolutionary Robotics

Christiaan J. Pretorius[*], Mathys C. du Plessis[†], Charmain B. Cilliers[†]
[*]*Department of Mathematics and Applied Mathematics*
[†]*Department of Computing Sciences*
Nelson Mandela Metropolitan University
Port Elizabeth, South Africa
emails: {cpretorius, mc.duplessis, charmain.cilliers}@nmmu.ac.za

***Abstract*—Evolutionary Robotics is concerned with using simulated biological evolution to automatically create controllers for robots. Simulation, which reduces the amount of real-world testing, is typically used to accelerate the evolution process. However, the creation of robotic simulators is a difficult and time-consuming process which requires expert knowledge. As an alternative to manual simulator creation, this paper describes the use of Neural Networks to act as simulators for an ultrasonic distance sensor in the Evolutionary Robotics process. The creation of the simulator Neural Networks is discussed and motivated. The simulators are evaluated by means of a comparison with test data. Finally, the simulators are validated by evolving a controller for an obstacle avoiding robot using the simulator Neural Networks. The experimental results show that Neural Networks can indeed be used to simulate an ultrasonic sensor in the Evolutionary Robotics process.**

Keywords-Robotics; Genetic Algorithms; Neural Networks; Simulators.

I. Introduction

Great advances have been made in recent years in the field of robotics. Robots are becoming cheaper, with more hardware capabilities and faster onboard computing [1]. A robot's behaviour is determined by a *controller*. The controller continuously receives input from the robot's sensors and gives output in the form of commands, for example motor speeds [2]. The underlying implementation of a controller depends on the application domain of the robot.

The manual creation of a controller by human experts is a time-consuming and complicated task, which may be infeasible due to the complexity of the robotic task [3]. The unstructured, noisy and dynamic nature of the real world environments in which robots are required to function, adds to the difficulty of creating controllers [4]. The cost of creating controllers, which currently constitutes up to a third of total expenses [5], will increase in future as robotic hardware continues to advance and the tasks required from the robots become more complex [6].

Evolutionary Robotics (ER) is a field that aims to simplify the creation of controllers by means of the principles of biological evolution [7]. Engineers can use ER to create complex controllers with minimum human input. The ER process is an extension of the theory behind Evolutionary Algorithms (EAs) to the realm of robotic controllers. A population of candidate controllers compete for the ability to produce offspring, based on the effectiveness of each controller.

Simulation has been used extensively in the ER process to avoid having to evaluate the performance of candidate controllers in the real world. The creation of these simulators may in itself be extremely time-consuming or infeasibly complex. This paper consequently reports on a different technique of creating robot simulators, based on Neural Networks (NNs). NNs have previously been used as controllers [8], but are not commonly used as simulators in the ER process. Previous research by the current authors have shown that NNs can be used as simulators in the ER process for motion simulation [9], and for modeling various sensors [10][11].

The focus of this paper is on NN simulators for an ultrasonic distance sensor. The ultrasonic sensor used in the study differs from previously modeled sensors in that it is less reliable and that its functioning is intermittent. These deficiencies require a slightly different treatment when simulating this sensor.

The remainder of this paper is structured as follows: Section II details the ER process and describes the role of simulation in the ER process. Section III provides related work on using NNs as simulators. Section IV gives a brief overview of robotic controllers. The experimental robot that was used in this study is described in Section V. Section VI describes the NN simulators that were used in this study along with the data acquisition approach. The training of the NNs is discussed in Section VII. An analysis into the accuracy of the trained NNs is given in Section VIII. The NN simulators are validated in Section IX by their use in the ER process to evolve a controller. Conclusions are drawn in Section X.

II. ER and Simulation

The ER process, in the context of this study, is a technique that can be used to automatically create controllers for robots by means of artificial intelligence. The basic procedure is as follows:

1) Randomly create a set (referred to as a population) of controllers for the robot.
2) Evaluate the effectiveness of each controller (known as the fitness of each controller). Stop the process if an adequate controller has been found.
3) Create offspring from the current population, giving preference to the more fit controllers by means of:
   - Mutations (Small random changes to each controller)
   - Crossovers (Combinations of subcomponents of parent controllers)
4) Replace the current population with the offspring population.
5) Return to Step 2.

Step 2 is typically the most time-consuming of the ER process as it requires the candidate controllers to be transferred to a real-world robot to evaluate their performance. The evaluation of controllers in the real world may not be possible, as a large number of controllers have to be repeatedly evaluated [1][7][12]. Furthermore, certain controllers could lead to erratic robotic movements which may potentially damage the robot hardware [1]. Evolution in simulation has been used by several researchers to avoid the time-consuming task of real-world evaluation [12][13][14].

The computational complexity of simulators must allow for relatively fast evaluation of controllers [15], while still approximating the real robotic environment [16]. Simulators have been created by several researchers [6][12][13][17][18][19][20] and can be categorised in three classes [11]:

- **Physics-based simulators** mathematically model the physical behaviour of robotic components. This type of simulation is typically accurate [21], although physics simulators use complex physics models [22] and their construction requires a considerable amount of human input [23]. Furthermore, physics models often contain simplifications or approximations of the real world, which could lead to factors like friction and inertia being overlooked [14].
- **Empirical models** make use of data collected from the real world [2][12][14]. This approach has the advantage of capturing the fuzzy characteristics of the robotic components [12]. A disadvantage of empirical models is that elementary data analysis techniques are often used in their construction, for example, basic interpolation (although more advanced techniques have been investigated [19][20]).
- **Hybrid models** combine physics and empirical models by utilising experimental data to optimise the parameters of the physics model [13]. A disadvantage of this modeling technique is that assumptions from the physics model are inevitably incorporated into the simulator.

The challenges in existing robotic simulators is the motivation for an alternative simulation scheme, namely simulators implemented using NNs.

III. Neural Network Simulators

Artificial Neural Networks are used to model the biological brain in software and consequently harness the computing power of the biological neurons by training the network to perform certain tasks. This research involves the use of NNs as robotic simulators in the ER process.

NNs are well suited for use as robotic simulators because of their noise tolerance and generalisation ability. The environment that is to be modeled is typically employed to create training data used in the construction of the NNs. Large amounts noise is inevitably present in the training data due to inaccuracies in the acquisition process [13]. NNs are known to be noise tolerant [24][25], which makes their application to this domain appropriate.

Training data inherently contains only a sample of the set of all states of the environment. Movements of a robot through its environment can only be sampled at discrete intervals. The number of samples are also constrained by the amount of time that is available to create training data. The ability to generalise over training data [26] makes it possible for NNs to interpolate to unseen environmental states.

NNs have been used as simulators in non-ER fields, for example, to produce realistic graphic animations [27], to animate a robotic arm [28], and to model the environmental interaction of a sonar sensor of an experimental robot [29].

To the best of the authors' knowledge, however, no previous investigations have been conducted into the usage of NNs as simulators in the ER process, apart from previous works by the authors. The current authors have previously demonstrated the use of NN simulators in the ER process for motion simulation [9] and modeling of several sensors, including: light sensors [10], touch sensors, tilt sensors and gyroscopic sensors [11]. This study demonstrates that Simulator Neural Networks (SNNs) can also be used for ultrasonic sensors.

IV. Robotic Controllers

This study is focused on a controller to perform obstacle avoidance. The goal of such a controller is to move a robot through a scene to a destination point as fast as possible without colliding with any obstacles [8][13][30][31]. A NN controller has previously been evolved to perform the obstacle avoidance task [8].

The controllers that were evolved in the current study were evaluated in the real world on an experimental robot which was created to perform the obstacle avoidance task.

V. Experimental Robot

The Lego® Mindstorms NXT [32] robotic components were used in this study to construct the experimental robot.

The components that were used in this study include the *central micro-computer*, an *ultrasonic sensor* and *servo motors*.

An *Obstacle Avoidance Robot (OAR)* was constructed from the NXT components. Two motors connected to wheels and two castor wheels were used to create the differential steering of the OAR. The orientation of the robot was obtained from a compass sensor during training, while the ultrasonic sensor was used to determine the distance between the robot and obstacles in the scene. SNNs were constructed to model the OAR's motion and sensor functioning.

VI. SNN Parameters and Data Acquisition

Empirically obtained training data was used to construct the SNNs for motion and ultrasonic sensor simulation. The ultrasonic sensor's readings were recorded as random commands were sent to the robot. This information was later extracted to create training data for the SNNs. The training data for the motion SNNs were obtained through motion tracking. Discretised time steps of 400ms were used as the time frame in which simulation was performed.

A. Motion Simulation

The SNNs for the motion simulation of the robot were described in [10][11] and are consequently only briefly mentioned here. The robot operated on a horizontal slip-resistant surface in a coordinate system that moves and rotates with the robot. Three separate SNNs were created to simulate changes in the x and y coordinates, and the orientation angle (Δx, Δy and $\Delta\theta$, respectively).

The current and previous motor speeds of the motors were given as inputs to each SNN. The inertia of the robot was thus taken into consideration by including previous motor speeds. Frames from a camera mounted on the ceiling were analysed to track the position of the robot. The tracking data concerning the orientation change angle was supplemented using the compass sensor to obtain averaged values. The state changes in response to random commands, in terms of orientation angle and position, were used to generate a set of training data for the motion SNNs.

B. Ultrasonic Sensor Simulation

To simplify the construction of the SNNs for the ultrasonic sensor, the assumption was made that all obstacles in the robot's environment have straight and perpendicular edges and that any obstacle could thus be represented simply by its bounding rectangle. The parameters used in the ultrasonic sensor SNNs are indicated in Figure 1. The distance, D, between the ultrasonic sensor and the relevant obstacle was taken into account, as well as the orientation angle, α, between the pulse emitted from the sensor and the normal to the relevant edge of the obstacle in question.

The ultrasonic sensor sometimes does not return a reading at all if the emitted pulse does not find its way back to the receiver on the sensor. Testing suggests that the ultrasonic sensor only successfully produces a value roughly 80% of the time when it is at random positions and orientations relative to obstacles. In order to model the functioning of the ultrasonic sensor realistically, it was thus deemed necessary to model the probability of the ultrasonic sensor returning a value. The assumption was made that the probability of the ultrasonic sensor successfully producing a reading depended on D and α (Figure 1).

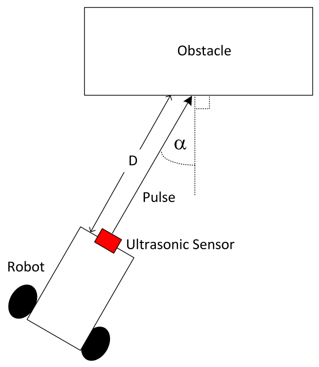


Figure 1. Parameters used in ultrasonic sensor models

Two different SNNs were thus employed to model the operation of the ultrasonic sensor. The mapping expected from each of these SNNs is shown in equations (1) and (2).

$$SNN_{ultra\ prob} : \{D, \alpha\} \rightarrow \{ultra_{prob}\} \tag{1}$$

$$SNN_{ultra\ value} : \{D, \alpha\} \rightarrow \{ultra_{value}\} \tag{2}$$

The first of the two ultrasonic sensor SNNs was used to predict the probability of the ultrasonic sensor producing a value ($ultra_{prob}$), given as inputs the distance and orientation angle to a relevant obstacle. This probability was expressed in the range [0, 1]. In the event of the ultrasonic sensor producing a reading, the second SNN would then be used to predict the actual reading produced by the ultrasonic sensor ($ultra_{value}$), again given as inputs the distance and orientation angle.

The random movement commands given during the motion tracking phase were used to move the robot through a scene containing various solid, straight-edged obstacles. By making use of the known position and orientation of the robot at any given point in time as well as the ultrasonic sensor values recorded to onboard memory, data could be parsed relating the probability of the sensor producing a value and this value itself to various distances from and orientations relative to obstacles. This would provide training data for the ultrasonic SNNs.

The sources of errors in the training data, i.e., human errors, inconsistencies in the motor control and noisy functioning of the ultrasonic sensor, were reduced as much as possible, for example, by repeating all physical measurements twice. Nonetheless, a considerable amount of noise remained which could potentially make the effective training of the SNNs challenging.

VII. Network Training

The SNNs were trained using a Genetic Algorithm (GA), although other optimisation techniques may also have been used. Each individual in the GA encoded potential weight values for a candidate SNN directly. A population of 100 randomly initialized chromosomes were used. A tournament among 33 individuals were used to select offspring. Simulated Binary Crossover [33] occurred with a probability of 80%, while mutations (normally distributed random values were added to each component) occurred with a probability of 5%. Training was halted when an improvement of less than 0.1% was found in the inverse of the mean-squared error of the fittest individual in the population over a period of 300 generations of the GA.

A Feed-Forward Neural Network (FFNN) with a single hidden layer was used as topology for the SNNs. Appropriate values for the number on neurons in each hidden layer were experimentally determined [34]. Each hidden layer contained an equal number of summation and product units, while output layers contained only summation units. Table I lists the activation functions used by each SNN, along with the number of hidden neurons that were employed.

Table I
Details of each SNN

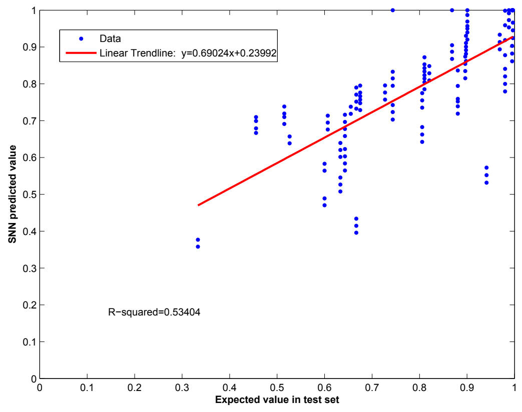
	Activation Function		
SNN	Hidden Layer	Output Layer	Number Hidden
Orientation Angle	Linear	Linear	4
X-coordinate	Linear	Linear	20
Y-coordinate	Linear	Linear	20
Ultrasonic Probability	Linear	Ramp	10
Ultrasonic Value	Linear	Linear	40

VIII. SNN Training Accuracy

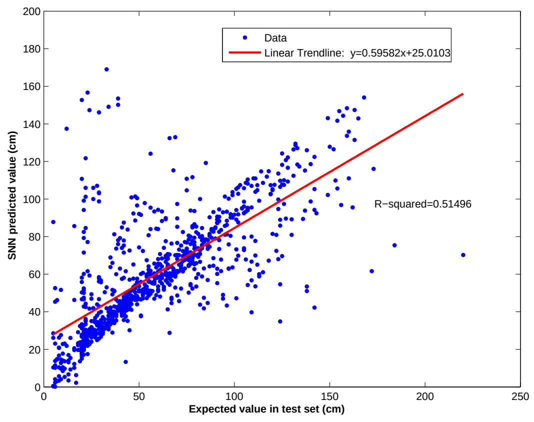
The training accuracy of the ultrasonic SNNs is illustrated in Figure 2. Each graph gives the output of the SNN plotted against empirically obtained values from the real world. The empirically obtained values used to perform this analysis were not used during the network training phase and were thus previously unseen by the SNNs. Each graph contains a trendline with its associated equation and R^2-value.

Figure 3 gives three-dimensional plots of the SNN outputs based on the distance and orientation of the sensor with respect to the obstacle. Two views of each surface are shown from different viewing angles. The plots also contain data from the test set for comparison.

Figures 2 and 3 illustrate that the SNNs generally trained relatively well (although the R^2-values are relatively low).



(a) Ultrasonic probability

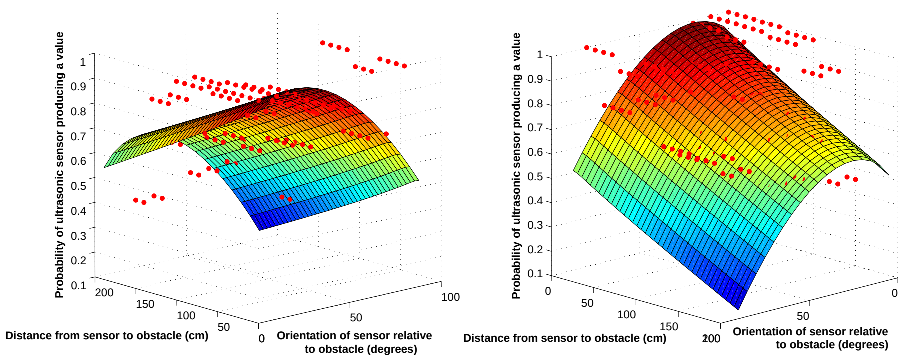


(b) Ultrasonic value

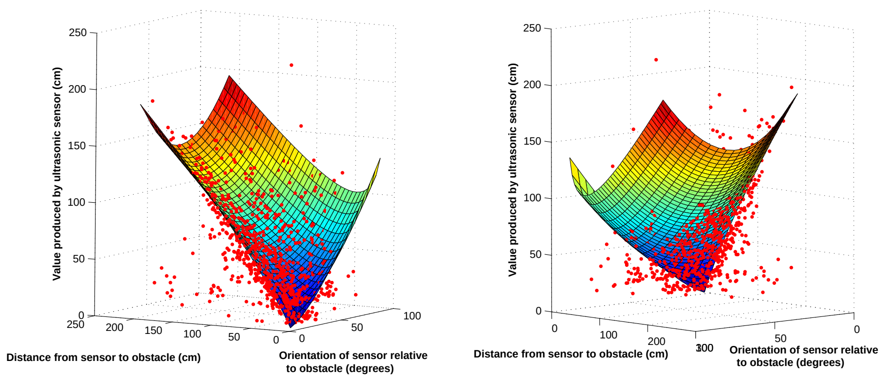
Figure 2. Comparison of expected and SNN predicted values for ultrasonic sensor SNNs of the OAR

The noise in the ultrasonic sensor value SNN test data can easily be seen in Figure 3(b) (see discussion on this figure later in this section). Visible in Figure 2(b) is a vertical line where the expected value is roughly 20cm. This can be attributed to the fact that the ultrasonic sensor was sometimes seen to produce an erroneous value of roughly 20cm regardless of the actual distance of the sensor from an obstacle. Taking the noise levels present in training data for the ultrasonic sensor SNNs into account, the accuracy obtained by these SNNs is reasonable.

The three-dimensional plots shown in Figure 3(a) indicate that a relatively good fit was obtained for the ultrasonic probability SNN. The general trend predicted by this SNN is as would be expected: The probability of the ultrasonic sensor firing successfully (that is, the emitted pulse returning to the receiver of the ultrasonic sensor) is highest when the sensor is close to an obstacle and facing it head-on (that is, it has a small orientation angle). This probability decreases as the obstacle gets further away and the relative angle between



(a) Ultrasonic probability SNN



(b) Ultrasonic value SNN

Figure 3. Outputs for the SNNs for various values of distance and orientation (test data included)

the robot and the obstacle becomes bigger, since the emitted pulse would have a low probability of reaching the receiver of the ultrasonic sensor under these circumstances due to it being scattered.

High levels of noise can be seen in the test data for the ultrasonic sensor value SNN in Figure 3(b). These noise levels were almost certainly also present in the training data for this SNN, and are probably caused by inaccuracies in the sensor itself. The surface produced by the SNN does not match the test data very accurately, but this is probably due to the erroneous test data. A trend can clearly be seen in the noise level present in the test data. When the orientation angle is small (that is, the ultrasonic sensor faces the obstacle almost head-on) the noise levels are low and the value produced by the ultrasonic sensor is roughly equal to the real-world distance between the sensor and the obstacle as determined through motion tracking. As this orientation angle increases, however, more noise is gradually introduced in the test data with the value produced by the ultrasonic sensor diverging more and more from the distance determined by motion tracking. These large levels of noise from the training data of the ultrasonic value SNN almost certainly impacted negatively on the training accuracy of said SNN.

IX. Controller Evolution

The actual effectiveness of the SNNs in the ER process could, however, only be demonstrated by evolving a controller in simulation using the SNNs and consequently successfully transferring the controller to the real world. A simple array-based obstacle avoidance controller was consequently evolved in simulation using the SNNs. Four different obstacle avoidance tasks where set by choosing four different initial positions and orientations for the OAR in a scene containing four stationary obstacles (refer to Figure 4). The goal of each of the four evolved controllers was to guide the OAR through the scene to a predetermined target point by using inputs from the ultrasonic sensor.

The obstacle avoidance task was previously solved using an open-loop time-based controller which did not take any sensory inputs into account [11]. However, in the current study the controller was not allowed to transition to new motor speeds based on time and was thus forced to use input from the ultrasonic sensor. Ultimately, the effectiveness of the evolved controller would be entirely dependent on the accuracy of the real-world sensor and of the developed SNNs.

A command set containing 4-tuples of the form (mot_1, mot_2, $bigorsmall$, $ultval$) was evolved. A given pair of motor speeds (mot_1 and mot_2) was maintained until the ultrasonic sensor produced a value (the sensor would sometimes not produce a value at all) and this value was larger than or smaller than (as determined by a boolean value *bigorsmall*) a threshold value ($ultval$). The next pair of motor speeds in the next tuple would then be executed. This process was continued until the end of the command set was reached. Each controller contained six of these 4-tuples.

A. Evolution Procedure

A total of four controllers were evolved to guide the robot to the target position from each of the four positions from which the robot was started. These controllers were evolved entirely in simulation, using the motion and ultrasonic sensor SNNs. The different tuples of the controllers were directly encoded as individuals in the algorithm. Identical parameters were employed in the ER process as was used in the GA described in Section VII.

The fitness function (F_{obs}), used to quantify the quality of each potential solution, is given in Equation (3) [11].

$$F_{obs} = \begin{cases} \frac{0.5}{dist_{crash}} & \text{if the robot crashed} \\ \frac{1}{dist_{final}} & \text{otherwise} \end{cases} \tag{3}$$

The value $dist_{final}$ is calculated as the Euclidean distance from target point to the robot's final position. The value $dist_{crash}$, which is used only when the robot crashed into an obstacle, is the Euclidean distance from the target point to the impact point.

The fitness function thus assigned high fitness values when the robot stopped close to the target position without colliding with obstacles. Controllers that caused collisions close to the target position were favoured over those that caused collisions far from the target position.

The SNNs were used to produce a simulated path for each controller in the ER population. Each controller was assigned a fitness based on the simulated path, using equation (3). Evolution was terminated after 1000 generations.

Randomness was incorporated into the ultrasonic sensor SNNs in that the ultrasonic probability SNN predicts the probability of the ultrasonic sensor producing a value. As a result of the randomness present in the ultrasonic sensor simulation, any given controller in the ER population would thus produce different behaviours in simulation when run multiple times. In order to thus produce a robust controller which would take into account the firing and non-firing of the ultrasonic sensor, the fitness of each controller in the ER population was determined by running each controller five times in simulation and summing the fitnesses produced in each of these five runs. The task to be performed by a controller evolved in this way would thus be not only to reach the target position as accurately as possible, but also to do this in spite of the noise present in the ultrasonic sensor.

B. Results and Discussion

The quality of the evolved controllers were investigated by executing the controllers on the real-world robot. The successful execution of the obstacle avoidance task in the real world is of paramount importance, as this determines whether SNNs can be successfully used in the ER process.

Figure 4 compares the paths followed by the simulated robot and the real-world robot for each of the four starting points considered. Three paths are illustrated in each case. Motion tracking was used to determine the real-world paths of the robot (indicated by solid lines in Figure 4).

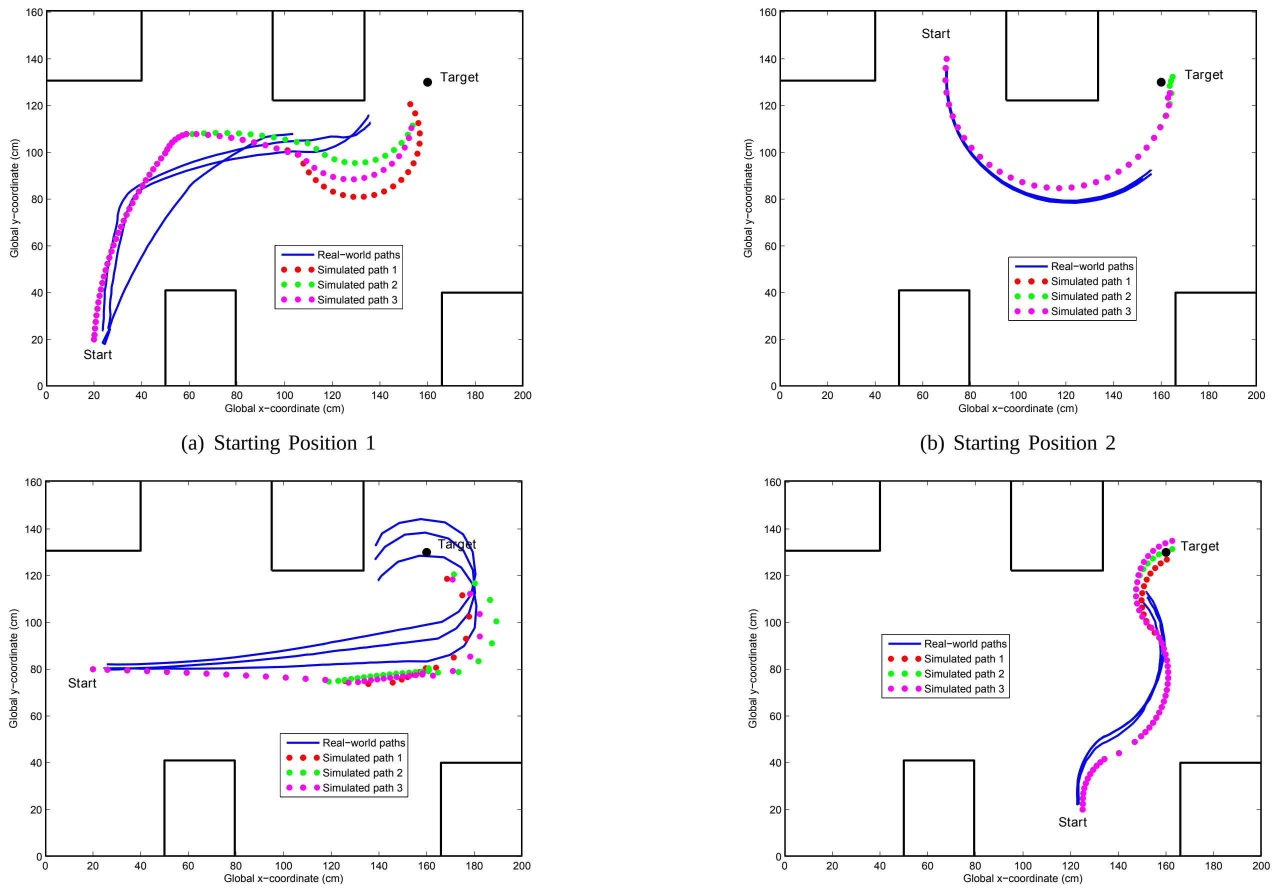
Results obtained for the ultrasonic sensor-based controllers show a relatively good correspondence between the simulated and real-world behaviours, although some discrepancies are evident. Notably, the simulated paths shown in Figure 4(a) can be seen to differ considerably from the real-world paths. This resulted from the robot crashing into an obstacle. In this case the SNNs thus failed to accurately model the robot's behaviour, although more accurate results were seen for the remaining three starting points.

Large amounts of noise present in the functioning of the ultrasonic sensor could have contributed to the differences between simulation and the real world. The fact that relatively consistent paths were observed for the evolved controllers in the real world from each starting position (roughly the same path was taken in all three real-world runs) indicates that the ER process succeeded in evolving controllers which could perform their task adequately. This is in spite of the ultrasonic sensor sometimes not producing a value and readings from this sensor generally containing large amounts of noise. The results thus indicate that the ultrasonic sensor SNNs (especially the ultrasonic probability SNN) represented the functioning of the ultrasonic sensor with reasonable accuracy.

X. Conclusion and Future Work

The main goal of this study was to determine whether SNNs can be successfully used to simulate an ultrasonic sensor in the ER process. The results of this study indicate that this could indeed be achieved, and provide evidence that the basic technique proposed in [9][10][11] can be extended to create simulators for more complex sensors.

Controllers were successfully evolved using the created SNNs. This suggests that, despite the limited and noisy training data, the SNNs managed to generalise. Readings from sensors will inevitably contain noise which results in unpredictable real-world behaviour. This is evident, for example, in the operation of the ultrasonic sensor. By creating the ultrasonic sensor probability SNN, a simple method has thus been suggested to model such unreliable sensors. Reasonable results were obtained using this method,



(a) Starting Position 1

(b) Starting Position 2

(c) Starting Position 3

(d) Starting Position 4

Figure 4. SNN predicted and real-world paths for the ultrasonic sensor-based controllers

in that controllers evolved in simulation using the ultrasonic probability SNN performed reasonably when executed on the real-world OAR. SNNs thus provide a method for modeling unreliable sensors.

A benefit of using SNNs as simulators is that their use do not require an in-depth analysis and understanding of the environment to be modeled and the underlying mechanics. This is a considerable advantage of the applied approach in comparison to more traditional approaches. Furthermore, the investigated approach has the advantage of implicitly incorporating the flaws and imperfections of the robotic hardware. It is not certain how well the applied approach will scale to other more complex robot systems. Future investigations can therefore explore the potential usage of SNNs to model other robotic sensors with large quantities of noise in their readings, or robots which need to function in highly dynamic environments.

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] S. Nolfi and D. Parisi, “Evolving non-trivial behaviors on real robots: An autonomous robot that picks up objects,” in *Proceedings of the 4th Congress of the Italian Association for Artificial Intelligence on Topics in Artificial Intelligence*. London: Springer Verlag, 1995, pp. 243–254.

[3] I. Harvey, P. Husbands, and D. Cliff, “Issues in evolutionary robotics,” in *Proceedings of the Second International Conference on Simulation of Adaptive Behavior: From Animals to Animats 2*. Cambridge: MIT Press, 1993, pp. 364–373.

[4] R. A. Brooks, “New approaches to robotics,” *Science*, vol. 253, no. 5025, pp. 1227–1232, Sep 1991.

[5] W. Van de Velde, “Toward learning robots,” *Robotics and Autonomous Systems*, vol. 8, no. 1-2, pp. 1–6, 1991.

[6] L. A. Meeden and D. Kumar, “Trends in evolutionary robotics,” in *Soft Computing for Intelligent Robotic Systems*, L. Jain and T. Fukuda, Eds. New York: Physica-Verlag, 1998, pp. 215–233.

[7] D. Floreano, P. Husbands, and S. Nolfi, "Evolutionary Robotics," in *Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Berlin: Springer Verlag, 2008.

[8] D. Floreano and F. Mondada, "Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot," in *Proceedings of the Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3*. Cambridge: MIT Press, 1994, pp. 421–430.

[9] 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*. New York: ACM, 2009, pp. 170–178.

[10] ——, "A neural network-based kinematic and light-perception simulator for simple robotic evolution," in *IEEE Congress on Evolutionary Computation*, 2010, pp. 1–8.

[11] ——, "Simulating robots without conventional physics: A neural network approach," *submitted to Journal of Intelligent and Robotic Systems*, 2012.

[12] H. H. Lund and O. Miglino, "From simulated to real robots," in *Proceedings of IEEE Third International Conference on Evolutionary Computation*, 1996.

[13] N. Jacobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *Proceedings of the Third European Conference on Advances in Artificial Life*. London: Springer Verlag, 1995, pp. 704–720.

[14] O. Miglino, H. H. Lund, and S. Nolfi, "Evolving mobile robots in simulated and real environments," *Artificial Life*, vol. 2, pp. 417–434, 1996.

[15] O. Miglino, K. Nafasi, and C. E. Taylor, "Selection for wandering behavior in a small robot," *Artificial Life*, vol. 2, no. 1, pp. 101–116, 1995.

[16] R. A. Brooks, "Artificial life and real robots," in *Proceedings of the First European Conference on Artificial Life*. Cambridge: MIT Press, 1992, pp. 3–10.

[17] J. Teo, "Robustness of artificially evolved robots: What's beyond the evolutionary window?" in *Proceedings of the Second International Conference on Artificial Intelligence in Engineering and Technology*, Kota Kinabalu, Sabah, Malaysia, 2004, pp. 14–20.

[18] V. Tikhanoff, A. Cangelosi, P. Fitzpatrick, G. Metta, L. Natale, and F. Nori, "An open-source simulator for cognitive robotics research: The prototype of the iCub humanoid robot simulator," in *Performance Metrics for Intelligent Systems Workshop, National Institute of Standards and Technology*, 2008.

[19] K. M. A. Chai, C. K. I. Williams, S. Klanke, and S. Vijayakumar, "Multi-task Gaussian process learning of robot inverse dynamics," in *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems*, 2008, pp. 265–272.

[20] T. Kyriacou, U. Nehmzow, R. Iglesias, and S. A. Billings, "Accurate robot simulation through system identification," *Robotics and Autonomous Systems*, vol. 56, no. 12, pp. 1082–1093, 2008.

[21] S. Carpin, T. Stoyanov, and Y. Nevatia, "Quantitative assessments of USARSim accuracy," in *Proceedings of Performance Metrics for Intelligent Systems Workshop*, 2006.

[22] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *In IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004, pp. 2149–2154.

[23] J. Rieffel, F. Saunders, S. Nadimpalli, H. Zhou, S. Hassoun, J. Rife, and B. Trimmer, "Evolving soft robotic locomotion in PhysX," in *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference*. New York: ACM, 2009, pp. 2499–2504.

[24] I. A. Basheer and M. Hajmeer, "Artificial neural networks: Fundamentals, computing, design, and application," *Journal of Microbiological Methods*, vol. 43, no. 1, pp. 3–31, 2000.

[25] S. I. Gallant, *Neural Network Learning and Expert Systems*. Cambridge: MIT Press, 1993.

[26] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. New Jersey: Prentice Hall, 2008.

[27] R. Grzeszczuk, D. Terzopoulos, and G. Hinton, "Neuroanimator: Fast neural network emulation and control of physics-based models," in *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. New York: ACM, 1998, pp. 9–20.

[28] P. O. Moreno, S. I. Hernandez Ruiz, and J. C. R. Valenzuela, "Simulation and animation of a 2 degree of freedom planar robot arm based on neural networks," in *Proceedings of the Electronics, Robotics and Automotive Mechanics Conference*. Washington, DC: IEEE Computer Society, 2007, pp. 488–493.

[29] 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.

[30] P. Vadakkepat, K. C. Tan, and W. Ming-Liang, "Evolutionary artificial potential fields and their application in real time robot path planning," in *Proceedings of the 2000 Congress on Evolutionary Computation*, 2000, pp. 256–263.

[31] J. Kodjabachian and J. Meyer, "Evolution and development of neural controllers for locomotion, gradient-following, and obstacle-avoidance in artificial insects," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 796–812, 1998.

[32] "LEGO.com MINDSTORMS : Home," retrieved: August, 2012. [Online]. Available: mindstorms.lego.com

[33] K. Deb and R. Agrawal, "Simulated binary crossover for continuous space," in *Complex Systems*, 1995, pp. 115–148.

[34] C. J. Pretorius, "Artificial neural networks as simulators for behavioural evolution in evolutionary robotics," Master's thesis, Nelson Mandela Metropolitan University, 2010.