# Towards an Artificial Neural Network-based Simulator for Behavioural Evolution in Evolutionary Robotics

Christiaan J. Pretorius
christiaanjohannes.pretorius
@nmmu.ac.za

Mathys C. du Plessis
mc.duplessis
@nmmu.ac.za

Charmain B. Cilliers
charmain.cilliers
@nmmu.ac.za

Department of Computer Science and Information Systems
Nelson Mandela Metropolitan University
PO Box 77000
Port Elizabeth, 6031

## ABSTRACT

It is not apparent to employ an Artificial Neural Network (ANN) as simulator structure during the Evolutionary Robotics (ER) process. Consequently, the potential for the use of an ANN in this regard has been investigated in this paper. This simulator Neural Network (NN) was trained to predict changes in the position and orientation resulting from arbitrary motor commands sent to a Lego Mindstorms NXT robot moving in a two-dimensional plane. The proposed NN simulator was employed as an alternative to conventional physics-engine based simulators to evolve simple navigation behaviour in said robot and encouraging results were obtained.

## Categories and Subject Descriptors

I.2.9 [**Artificial Intelligence**]: Robotics; I.6.5 [**Simulation and Modeling**]: Model Development; F.1.1 [**Computation by Abstract Devices**]: Models of Computation

## General Terms

Algorithms, Design, Experimentation

## Keywords

Evolutionary Robotics, Simulator Neural Networks, Lego Mindstorms

## 1. INTRODUCTION

The field of Evolutionary Robotics (ER) allows for the automatic artificial evolution of optimized task-execution behaviours for autonomous robots [7]. ER was born from the need to develop accurate yet robust control structures for modern robotic systems which are required to perform well in unstructured, noisy and dynamic real-world environments [4]. As a result of the improved onboard computational power of modern robots, the possibility exists for them to perform increasingly complex tasks, the explicit programming of which becomes almost unfeasibly complex [9]. The aim of ER is thus to allow for the evolution of robot behaviours to accomplish complex tasks with minimal need for human intervention.

As will be discussed in more detail in Section 2, the evolution of robotic control structures using ER can be drastically accelerated by making use of some form of robot simulator. Artificial Neural Networks (ANNs) are not generally used to implement these simulator structures. However, since ANNs have many features which can be employed to accurately train a robot simulator, such as noise tolerance and generalization ability, constructing a simulator using an ANN seems to be a possibility worthy of investigation and this is the endeavour of this work.

The rest of this paper is structured as follows: An overview of the field of ER is given in Section 2, from where should become apparent the potential for the novel approach proposed in this work (Section 3). In order to test the viability of the simulator Neural Network (NN) proposed, behaviours evolved using this approach were tested on a real-world robot (Section 4). Decisions that were made in terms of the structure of the NN to utilize and other implementation details are discussed (Section 5), as well as the experimental procedure employed (Section 6). Encouraging results were obtained (Section 7), which are discussed in Section 8. Finally, conclusions are drawn and possible future work is discussed in Section 9.

## 2. BACKGROUND

ER evolves robotic control structures (most popularly controller NNs) through a process analogous to the Genetic Algorithm (GA) [10]. Typically these control structures are employed to generate appropriate robotic behaviours in reaction to some stimulus from the robot's external environment (normally detected by means of robotic sensors).

In order to evolve control structures using the ER process, a population of candidate chromosomes is created, each encoding, for example, the potential weights to be used in a controller NN. Following this, a controller NN is constructed using the weights encoded by each chromosome in turn. The efficacy of each of the constructed controller NNs in allowing the experimental robot to execute a certain task is then gauged and each chromosome is assigned a fitness value proportional to the relative performance of the control structure encoded by that chromosome.

A new generation of chromosomes then replaces the previous generation. The chromosomes in this new generation are produced by reproduction (crossover and mutation) of chromosomes in the previous generation, with high-fitness chromosomes having a higher probability of being chosen to partake in producing offspring chromosomes. During the crossover process genetic material (or knowledge related to which control structures work effectively and which do not) is passed from the previous generation to the next one. Mutation then causes small random changes in this genetic material which allows for different, and perhaps better, control structures to be represented in the next generation. This process is repeated for a large amount of generations, evolving chromosomes with steadily-increasing fitness. At the end of the ER process an optimized robotic control structure is thus produced with minimal human input.

A major issue in ER is found in gauging the performance of the candidate control structure encoded by each chromosome in the population, in order to assign a fitness value to said chromosome. Typically many thousands of these evaluations are needed during an ER exercise, meaning that performance evaluation by uploading the control structure to the real-world robot is time-consuming and financially costly.

## 2.1 Evolution in Simulation

In order to address the difficulties of real-world controller evaluation mentioned above, controller evolution in simulation has been used by several researchers [8, 12, 14, 16, 19, 25]. In this approach, the candidate control structure encoded by each chromosome in the population is employed to control the behaviour of a simulated robot operating in a simulated environment. Such simulated evolution dramatically accelerates the ER process, saves on financial cost and prevents wear on the real-world experimental robot from repeated controller-performance evaluations.

The majority of simulator structures used for simulated evolution are implemented by means of physics-engines [23, 24]. Although these structures have been shown to be quite accurate and work effectively as simulators during ER, their development can be complex, time-consuming and costly and their execution is often computationally very demanding. The complexity of these simulators follows from the fact that in order to represent all the intricacies involved in the interaction between the robot and its environment properly in a physics-engine, many physics principles need to be incorporated in the design. These principles include concepts such as static and kinetic friction, moments of inertia and mass distribution.

Efforts to reduce the complexities involved in the design of certain simulator structures have been reported [13, 21], but not directly with reference to the field of ER. It is presumed that most researchers in ER favour the accuracy offered by complex simulators above the development time and expense of these simulators. In addition to this, not many alternative simulators to physics-engines exist for robotics research, meaning that most researchers simply employ the simulators which are available.

## 3. PROPOSED SIMULATOR

This work proposes an ANN as an alternative simulation technique for use in a simulated ER process. It is argued that the construction of such a simulator structure could be simpler than that of a physics-engine simulator and that less computational power could be required by said simulator.

The specific NN simulator proposed in this work was used to predict the change in the state of a robot's position and orientation in a two-dimensional plane while performing arbitrary motor commands, with no additional sensor inputs (Section 5.3). The motivation for employing an NN to this end, rather than certain other techniques which could also be used to create simulator structures (for example, regression techniques such as Gaussian Processes), is the ability of NNs to take on many different topologies. NNs can be used to construct open-loop simulators (as will be done in this study by employing a Feed-Forward Neural Network, Section 5.3) or closed-loop simulators (by employing, for example, a Recurrent Neural Network). The distinction between an open-loop and closed-loop prediction schema lies in the fact that an open-loop structure makes predictions based purely on information directly available from the system about which said schema makes the prediction [22]. The NN employed in this study is thus an open-loop structure, since it employs only motor speeds of the robot to predict changes in its position/orientation. Conversely, a closed-loop structure makes predictions based not only on direct information from the system, but also on indirect information, such as the accuracy of previous predictions. If needed, closed-loop structures can also fine-tune their predictions based on sensory information available from the system [22]. Although it could thus be argued that a simple regression technique could be employed in this study with similar or indeed superior accuracy to the NN employed, research into the use of an NN simulator was warranted as a result of the extendability of such structures for more complex closed-loop simulators in future research. In fact, it has been shown [11] that any algebraically computable function can be expressed as a Recurrent Neural Network, thus meaning that such NNs are ideal for use as robotic simulators since these NNs should, in principle, (assuming that enough experimental data can be obtained for training) be able to approximate any relationship between robotic actions and the effects of these actions.

In order to construct a physics-engine simulator to predict the position/orientation state changes of the robot, many mechanics principles would have to be included, such as the kinetic and static coefficients of friction between the robot wheels and its operating surface, the moment of inertia of various components of the robot and linear and rotational equations of motion. None of these factors need to be explicitly considered in developing an ANN simulator. It can be argued that by making certain assumptions, many of these mechanics principles could be ignored, while still arriving at simple equations which could be used to explicitly relate the change in position and orientation of the experimental robot to its wheel speeds. Such equations would likely lead to relatively accurate predictions of position/orientation changes. However, over-assumptions which would need to be made in arriving at these equations (such as ignoring, for example, the brief period of acceleration/deceleration which the robot undergoes when changing motor speeds) would inevitably lead to errors in the prediction ability of these equations. It was specifically decided to employ an NN in the study presented here, since it would cater for such effects. Furthermore, the need for explicit derivation of equations of motion predicting the changes in position/orientation of the robot was eliminated by employing an NN simulator structure. Sufficiently accurate predictions of robotic motion obtained from the proposed simulator would thus justify the notion that the use of an ANN as simulator can (at least from a mathematical point of view) simplify simulator design and, as a result, the ER process as a whole.

## 4. EXPERIMENTAL ROBOT

Use was made of the compact, modular end-user robotics range from Lego, called the Mindstorms NXT [3] as the experimental robot. The use of Lego robots in the field of ER has also been reported by several other researchers [15, 17, 18].

The reason that this robot was decided upon, is that it is very simple to configure and program. Due to the plug-and-play nature of the sensors and motors used in the Mindstorms robot, no prior engineering knowledge was needed to configure these devices to work properly. The morphology of this robot is also completely customizable. For the work outlined here, the configuration employed was as shown in Figure 1.

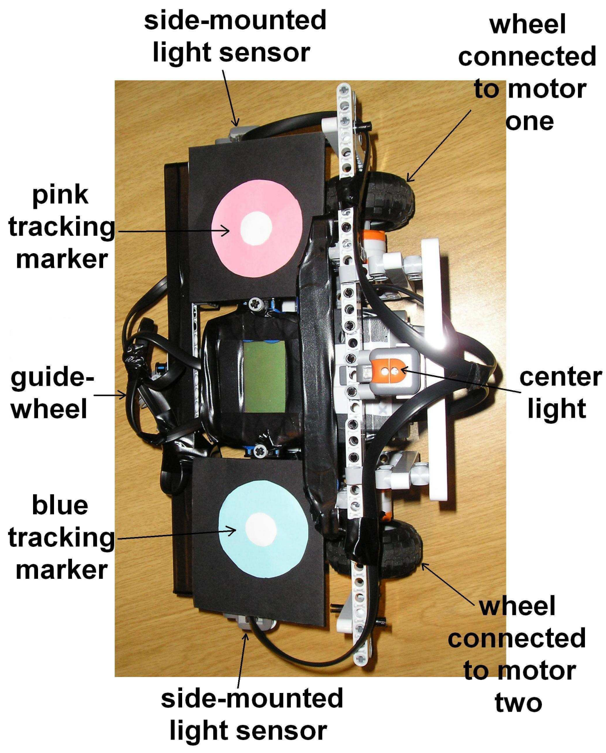
side-mounted light sensor
wheel connected to motor one
pink tracking marker
guide-wheel
center light
blue tracking marker
wheel connected to motor two
side-mounted light sensor

**Figure 1: Robot Morphology**

As can be seen from Figure 1, a very simple shape was decided upon, incorporating two wheels, each controlled by a separate servo motor, and two light sensors. At the back of the robot a small guide-wheel was attached, used simply to provide a third contact point with the working surface (apart from the two large wheels) to prevent excessive friction. The two large rubber wheels were employed in order to allow the robot to execute simple motion through differential steering. Two side-mounted light sensors were used to detect lines drawn around the perimeter of the working surface, to prevent the robot from leaving the surface during data acquisition. The center-mounted light was used as a synchronizing device for video analysis, while the blue and pink markers were used to track the position and orientation of the robot in the video footage (Section 5.2).

Due to the reasonably slow onboard processor of the Mindstorms robot and its limited onboard memory, it was decided to execute code on a "server"-like personal computer and relay commands to the robot via a Bluetooth connection, using the robot's onboard Bluetooth device. This was achieved by making use of the Java-based API, ICommand [1], which provides high-level functionality for establishing a Bluetooth connection with the robot, as well as sending commands and receiving feedback from the robot over this connection.

The Lego Mindstorms NXT robot operates on six AA batteries. For this work, use was made of rechargeable batteries which could be charged when not in use to allow for long periods of testing. It was discovered that the performance of the robot's motors depends rather largely on the charge-level of the batteries, and it was thus decided to employ the motor speed regulation capability of the ICommand package [1] and to ensure that batteries were charged above a certain level during all tests. The maximum motor speeds sent to the robot during testing were also limited to half of the full speed of which the motors are capable in order to minimize the battery-dependence factor and reduce a small degree of slipping of the robot's wheels which was observed on the working surface at higher speeds.

## 5. IMPLEMENTATION DETAILS

In order to ensure that the final simulator NN was as accurate as possible, much effort was expended in deciding upon the implementation details for various factors related to the data acquisition and NN training processes in this study. These details are discussed below.

### 5.1 Robot Commands

Each of the commands given to the robot in this work consisted of three elements:

- Motor speed for motor one
- Motor speed for motor two
- Execution time of command (in milliseconds)

The motor speeds for each command consisted of a randomly generated magnitude (in the range 0 to 50% of maximum motor speed) as well as a randomly generated direction (forwards or backwards). Furthermore, a random execution time in the range 0.5 to 3 seconds was used for each command. Although commands were randomly generated, the generation was seeded to be biased towards equal motor speeds in the same direction (which would cause the robot to move in an approximately straight line) and for both motor speeds to equal zero (which would cause the robot to stop), since it was envisaged that these commands would be required regularly and thus needed to be represented well in the training data for the simulator NN. This seeding of commands was achieved by ensuring that 50% of the commands consisted of totally random motor speeds, 30% of equal motor speeds in the same direction and setting both motor speeds to zero the remaining 20% of the time.

## 5.2 Motion Tracking

It was decided to make use of a roof-mounted camera to track the state changes (i.e. changes in position and orientation) of the robot in response to commands. The motivation for the use of this technique is that it would dramatically accelerate the data acquisition process and also lead to more accurate data acquisition than would be the case if use was made of manual measurements, since human error was eliminated as a possible source of error.

In order to track the robot, use was made of the pink and blue markers shown in Figure 1. Each of these markers contains a small white dot in its center. The effect of these dots in the images obtained from the camera, was a gradual fade from blue (or pink) to white from the circumference of the marker to its center. By carefully using this colour gradient and comparing the colour value of pixels near the center of the marker with that of their neighbouring pixels, the center of each marker could be determined in each frame of the video footage with reasonable accuracy. Each of the markers was placed on a black background and some other features of the robot were blackened out (Figure 1) in order to improve the tracking accuracy by eliminating colours which could match those of the markers.

After tracking the positions of the marker centers in each frame, the pixel coordinates of the markers in the image were converted to real-world coordinates on the operating surface of the robot by employing the JCamCalib Java Camera Calibration Package [2]. This package allowed for the calculation of the camera's intrinsic parameters which could be used to undistort the images obtained from the camera. It also provided the functionality to construct a homography matrix to convert from pixel coordinates to real-world coordinates.

Although certain inaccuracies were inevitably present in the motion tracking process, satisfactory results were generally obtained, with the position of each of the two marker centers being tracked to within approximately 2cm of their actual positions.

## 5.3 Simulator Neural Networks

Three separate simulator NNs were employed in this study, one each for the prediction of the change in the x-coordinate, y-coordinate and orientation angle of the robot respectively in reaction to an arbitrary motor command. The change in the orientation angle of the robot was measured relative to a fixed, globally-defined vector, while a simple Cartesian coordinate frame was used to measure the change in the position of the robot. This coordinate frame was locally defined to move with the robot during testing.

Given as input to each of the simulator NNs were the two motor speeds maintained by the robot before receiving the current command, the two motor speeds given as part of the current command and the time the current command was allowed to be executed. In response to this input data, the networks were expected to predict the change in the x-coordinate, y-coordinate and orientation angle of the robot.

The motivation for presenting each of the NNs with the two motor speeds maintained before receiving a certain command, is that it was anticipated that the robot would undergo a certain amount of acceleration/deceleration before stabilizing on the motor speeds given as part of the current command. This acceleration/deceleration would depend on the previously maintained motor speeds and this thus warranted giving this information as input to the NNs. The fact that the robot would take a certain amount of time to stabilize on constant motor speeds after receiving a new command was also the reason for imposing a minimum on the duration time of each command (Section 5.1).

In Figure 2 is an illustration of each of the three NNs used:

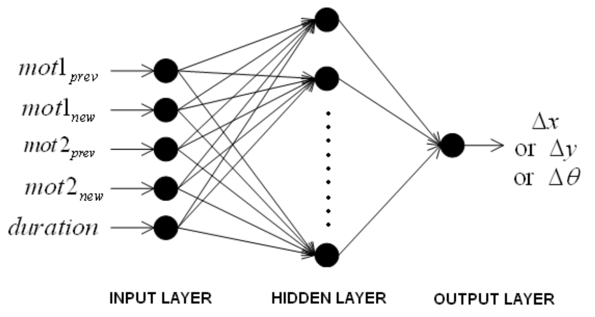

**Figure 2: Neural Network Structure**

A linear activation function was used for all neurons in the simulator NNs. In the hidden layer of all three these networks, 20 hidden neurons were employed of which 10 were implemented as product units and the remaining 10 as summation units.

After experimentally acquiring training data, each NN was trained by making use of a Genetic Algorithm in which each chromosome encoded potential weight values of the network. The decision was made to evolve weight values for the NNs using a Genetic Algorithm rather than training the NNs using more conventional training techniques (such as back-propagation), since the topology of the error landscapes for the NNs was not known before starting training. These landscapes could possibly be irregular with local minima in which conventional training techniques would get caught, leading to premature convergence. Local minima could be a result of the physical problem being solved: A certain combination of weight values in an NN could lead to relatively good predictions by the NN, although this combination of weights is still not the ideal combination, since better predictions could be obtained when employing another (superior) combination of weights. First-mentioned combination of weights would thus form a local minimum in the error landscape where conventional training techniques could be trapped. Another possibility is that local minima were introduced in the training data as a result of errors involved in the motion tracking or other stages of data collection. Employing a Genetic Algorithm would circumvent difficulties introduced by such local minima, since Genetic Algorithms have been shown to be efficient and accurate for optimization over error landscapes which are discontinuous, non-differentiable or noisy [6]. Parameters used during the NN training process were as shown in Table 1.

The fitness function employed during this training process was simply the inverse of the Mean Squared Error (MSE)

**Table 1: Genetic Algorithm parameters used during Neural Network training**

| | |
|---|---|
| Population size: | 250 |
| Crossover probability: | 80% |
| Crossover method: | Simulated Binary Crossover (SBX) |
| Mutation probability: | 5% |
| Selection method: | Tournament(50 individuals) |

produced by the relevant NN when constructed from the weights encoded by a certain individual. The MSE is defined as:

$$MSE = \frac{1}{N}\sum_{p=1}^{N}\sum_{i=1}^{O}(t_{pi} - o_{pi})^2, \qquad (1)$$

where $N$ is the number of training patterns in the training set, $O$ is the number of outputs produced by the NN and $t_{pi}$ and $o_{pi}$ are the expected (correct) output and output produced by the NN at the $i$-th output neuron when presented with the $p$-th training pattern in the training set.

Training for each NN was carried out for 15000 generations to ensure that the NNs were as accurate as possible after training. This process took approximately 12 hours per NN.

## 6. EXPERIMENTAL PROCEDURE

The procedure followed in this study was firstly to acquire data with which the simulator NNs were trained. Following this, a validation experiment was conducted in which a very simple robotic control structure was evolved using the developed simulator NNs.

### 6.1 Simulator Training

In order to obtain training data for the simulator NNs, a series of 5000 random commands was sent to the robot. Video footage of the reaction of the robot to each of these commands was recorded by a roof-mounted camera. The robot was operated on a working surface of dimensions 2.76m by 1.84m positioned directly under the camera. The material for this surface was chosen to be slip-resistant, to reduce slippage of the robot's wheels during testing.

Each of the commands sent to the robot over the Bluetooth connection as well as the execution duration of each were carefully logged, along with the exact system time of the computer from which these commands were sent. At the beginning of the video clip, before sending any commands, the center-mounted light on the robot was briefly flashed into the camera. The frame in the video where this flash was seen was located and the system time on the computer when the command to flash the light was sent was logged. Using this information, along with the known frame rate at which the video was recorded, a direct mapping between the frames in the video and the system time of the computer could be determined, and this mapping was used to determine the relevant frame in the video when each subsequent command was received by the robot.

After recording the relevant video footage, the video was post-processed by extracting each frame of the video as an image file. By employing the tracking software discussed earlier, the position of the robot's markers was determined, which in turn was used to determine the position and orientation of the robot in each frame.

By making use of the robot position and orientation for each frame as well as the synchronization information between the system clock and frame sequence of the video, data could be extracted relating robotic response to each command. This data was then used to train the simulator NN using a Genetic Algorithm (as discussed in Section 5.3).

### 6.2 Validation Experiment

After training was completed, the ER process was employed to evolve a simple navigation controller for the robot in simulation, using the simulator NNs to monitor the motion of the robot in its simulated environment. The behaviour to be evolved during this process was deliberately chosen to be very simple, since this validation experiment was intended only to gauge the accuracy of the simulator NNs. The evolution of similarly uncomplicated behaviours has been reported [5, 12, 15, 19, 20].

In Figure 6, the required behaviour of the robot is shown. The operating surface of the robot was divided into nine rectangles, each 80cm x 50cm in size. The robot was initially placed in the center of the center left rectangle and orientated to face directly towards the rectangle numbered 1. The task to be completed by the robot was to traverse the board in a clockwise direction, passing through each of the numbered rectangles in the sequence that they are numbered, while avoiding the center rectangle (Forbidden Zone in Figure 6) and the area surrounding the nine-rectangle grid. Finally, the robot was to return to its initial rectangle and stop.

It was decided to evolve a command set simply specifying a sequence of commands and the execution duration for each needed to achieve the desired behaviour. Each chromosome in the population of this ER process thus encoded a potential set of commands. By making use of the simulator NNs the outcome of the execution of the command set encoded by each chromosome could be determined in simulation. Based on this outcome, a fitness value was assigned to each chromosome in the population using the following fitness function:

$$F = N_{max}^2 - \frac{C_{nogain}}{10}, \qquad (2)$$

where $N_{max}$ is the number of the furthest rectangle reached while passing through all previous rectangles and not passing through the forbidden zone and $C_{nogain}$ is the number of commands in the command set of the relevant chromosome which did not cause the robot to move from one rectangle to the next (i.e. the robot moved to the previous rectangle or stayed in the current rectangle).

If it was discovered that a certain command set sent the robot on a path through the forbidden zone or the surrounding area of the nine-rectangle region, or that a certain block was not visited in arriving at a subsequent block, the relevant chromosome's fitness was immediately reduced heavily.

For the crossover process, it was decided to introduce certain rules governing which commands in the command set encoded by each of the parent chromosomes would be paired up and crossed over to produce a new command for offspring chromosomes. The crossover point would be selected to cross over commands which correspond to each other in that the simulated robot was at approximately the same position in the simulated environment when receiving each of these commands (i.e. in the same block). It was anticipated that this selection criteria would accelerate the evolution process, since each of the commands selected for crossover would need to be similar (each command would aim to take the robot in a similar direction).

In order to keep the crossover process as simple as possible, constant-length chromosomes (each encoding a fixed number of commands in its command set) were used during evolution. However, the possibility was introduced for a command in the evolved command set to be a null command, which meant that no command was given to the robot when arriving at said command in the command set. This thus allowed for command sets effectively consisting of differing quantities of commands to be evolved, while ensuring an uncomplicated crossover process.

The remaining parameters employed during this ER process were the same as were used for the Genetic Algorithm to evolve NN weights for the simulator NNs (Table 1). Evolution was allowed to proceed for 1000 generations and was repeated multiple times to produce different optimized command sets. After evolution was completed, each of the evolved command sets were sent to the real-world robot and their performance in executing the task in the real-world was examined.

## 7. RESULTS

In order to evaluate the accuracy with which the simulator NNs were trained, the final value of the MSE (Equation 1) produced by each NN after training was considered when each NN was presented with data from the validation set (see Section 7.1 for an explanation of this term). Furthermore, the values predicted by each of the three simulator NNs (in terms of the change in the position/orientation of the robot) were compared with expected values for certain commands present in the validation set.

These results are shown in this section, along with the results obtained in the validation experiment, where the robotic behaviours evolved in simulation using the simulator NNs are compared with the behaviours observed when transferring the evolved behaviours to the real-world robot.

### 7.1 Simulator Accuracy

In Table 2, a summary of the accuracy achieved by each simulator NN after training is shown. In this Table, the MSE value produced by each NN when presented with data from the validation set is shown, along with the average error present in the prediction made by each NN for the change in position/orientation of the robot for commands in the validation set. The validation set is a set of data which was collected during data acquisition, but not presented during the training phase of the NNs and is used to gauge the accuracy and generalization ability of the NNs.

**Table 2: Summary of NN simulator accuracy**

| NN Simulator | Final MSE | Average absolute error |
|---|---|---|
| change in angle | 26.412 | 3.585 degrees |
| change in y-coordinate | 12.909 | 2.143 cm |
| change in x-coordinate | 18.559 | 2.782 cm |

Figures 3, 4 and 5 show plots of the expected value and value predicted by each of the three trained simulator NNs for the change in the orientation angle, y-coordinate and x-coordinate respectively resulting from 50 commands in the validation set.

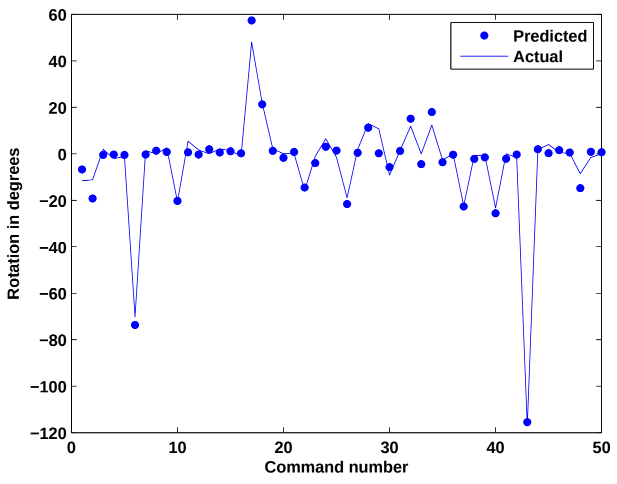

**Figure 3: Comparison of predicted and actual values for change in angle**

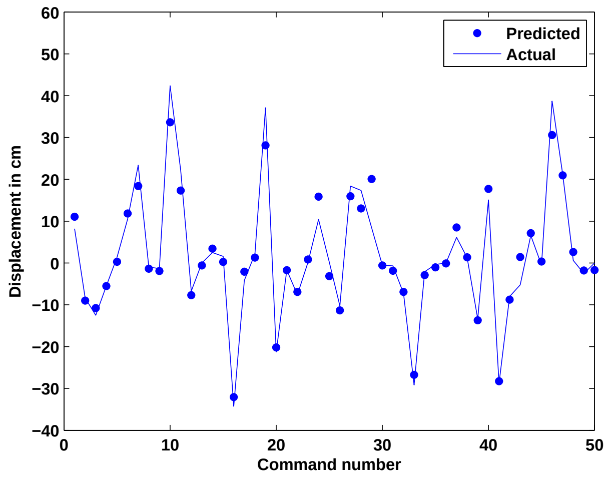

**Figure 4: Comparison of predicted and actual values for change in y-coordinate**

As can be seen from Table 2 and Figures 3, 4 and 5 all three the NNs trained reasonably well and the values predicted by the NNs are generally quite close to the expected values. If it is taken into consideration that the change in the y-coordinate of the robot as a result of a given command was on average in the order of 25cm, the errors present in the predictions of these changes are relatively minor (Table 2). The same holds for the prediction of the orientation angle and x-coordinate changes. The x-coordinate NN is perhaps the slight exception to the high degree of accuracy obtained, as it can be seen that the predictions from this network do

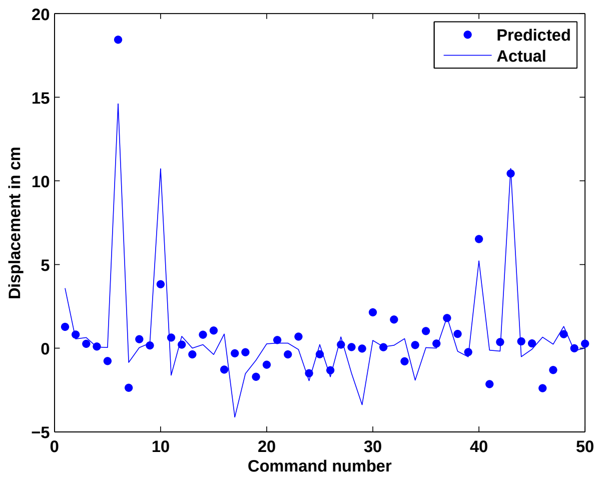

**Figure 5: Comparison of predicted and actual values for change in x-coordinate**

differ occasionally from the expected values, and are on average less accurate than the predictions of the change in y-coordinate.

The suspected reason for this is that the changes in the x-coordinate of the robot observed during data acquisition were generally much smaller than the changes in its y-coordinate. This could thus cause the errors introduced by the motion tracking to be much larger relative to the change in the x-coordinate than was the case for the y-coordinate. These relatively large errors in the x-coordinate data presented to the relevant NN could have contributed strongly to the sub-optimal results obtained for this network.

### 7.2 Validation Results

The simulation-evolved ideal paths for performing the task explained in Section 6.2 as well as those observed when transferring the simulation-evolved control sets to the real-world robot are shown in Figures 6 and 7 for the best behaviours which could be evolved using 10 and 13 commands respectively.

Firstly, it can be seen from the figures that the behaviour evolved in simulation does indeed accomplish the required task. What has thus been achieved is the accurate execution of a specified task with no need for manual robotic programming, which is the major advantage of the ER process.

Secondly, it can be seen that the predicted and real-world paths are reasonably close to one another, indicating that the simulator NNs achieved a reasonably high level of accuracy in predicting state changes in the robot as it traversed the working surface.

It can also be seen from the figures that the real-world and predicted paths steadily diverge further from each other as is moved further along each path (especially in Figure 6). This was to be expected, since the errors that are present in the simulator NNs' predictions of the state changes of the robot accumulate, i.e. the error made in the prediction of the state change resulting from the first command in the command set is effectively added to the error in predicting the second state change and so on.

## 8. DISCUSSION

The experimental results presented suggest that the use of an ANN as simulator structure during a simulated ER process is indeed feasible for evolving robotic behaviours to accomplish simple tasks. Although a direct comparison with the use of a physics-engine simulator to evolve the behaviour required in this study is impossible, it can be stated that the NN simulator employed here is, if perhaps not more accurate than, at least an alternative to a physics-engine simulator.

The reasonable accuracy of the results obtained becomes more noteworthy when all the sources of possible errors during the data acquisition, NN training and behavioural evolution processes are taken into account. Possible sources of errors include those introduced by the motion tracking software, synchronization between the video footage and command logs, undistortion of images and pixel-coordinates to real-world coordinates mapping. Small errors are also suspected to have been introduced by the physical robot itself from inaccuracies in its motor control capabilities, as well as slipping of the robot on its operating surface. It is believed that the noise-tolerance abilities of the simulator NNs employed contributed strongly towards minimizing the effect of errors in training data on final NN accuracy.

The significance of the simulator NNs developed in this work should not be underestimated. What effectively has been created here is a simulator structure which enables the robot to navigate blindly in its environment, with no sensor inputs whatsoever. The only information needed in order to establish the state change in the robot's position/orientation is that regarding the motor movements performed by the robot. Although the control structure developed here is inherently open-loop in that it does not employ any sensor readings, the reasonably accurate results obtained in this study mean that when a closed-loop control structure is considered (for example, a Recurrent Neural Network as robotic controller), even more accurate results will hopefully be obtained.

The rather simple nature of the system for which the employed simulator NNs were used to predict state changes could be criticized. It is uncertain whether the technique employed in this study will scale well to predict state changes for more complicated systems, and this will indeed be the endeavour of future research. Given the noise-tolerant and generalization abilities of NNs and the encouraging results obtained in this study, it is envisaged that, if enough training data can be obtained, NNs can indeed by employed as simulators in more complex systems.

## 9. CONCLUSIONS AND FUTURE WORK

As a first attempt at employing an NN simulator structure for behavioural evolution in ER, the results obtained in this study are very encouraging and suggest that further investigation into this possibility is warranted.

Many possibilities for future improvements have been identified. Future research could include:

- possible improvements to the motion tracking software
- addition of sensors to the robot and integration of sensor readings into the simulator NNs to improve prediction accuracy
- simulator construction for more complex systems, i.e. systems with more degrees of freedom for which it is

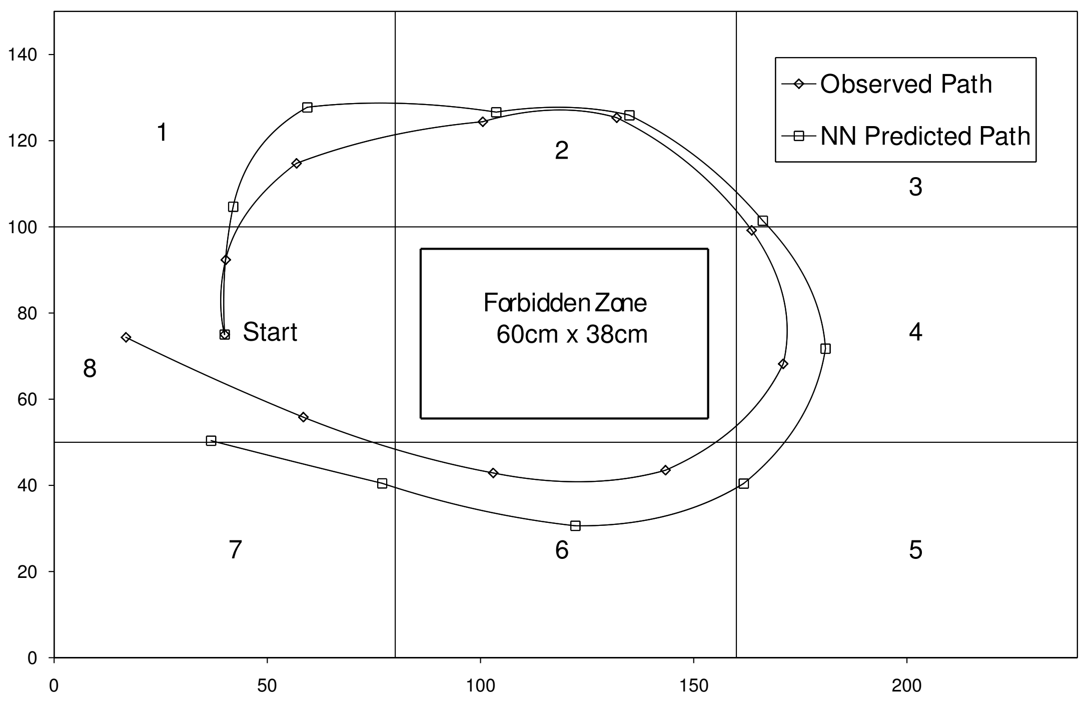

**Figure 6: Predicted and actual robot paths for command set of size 10**

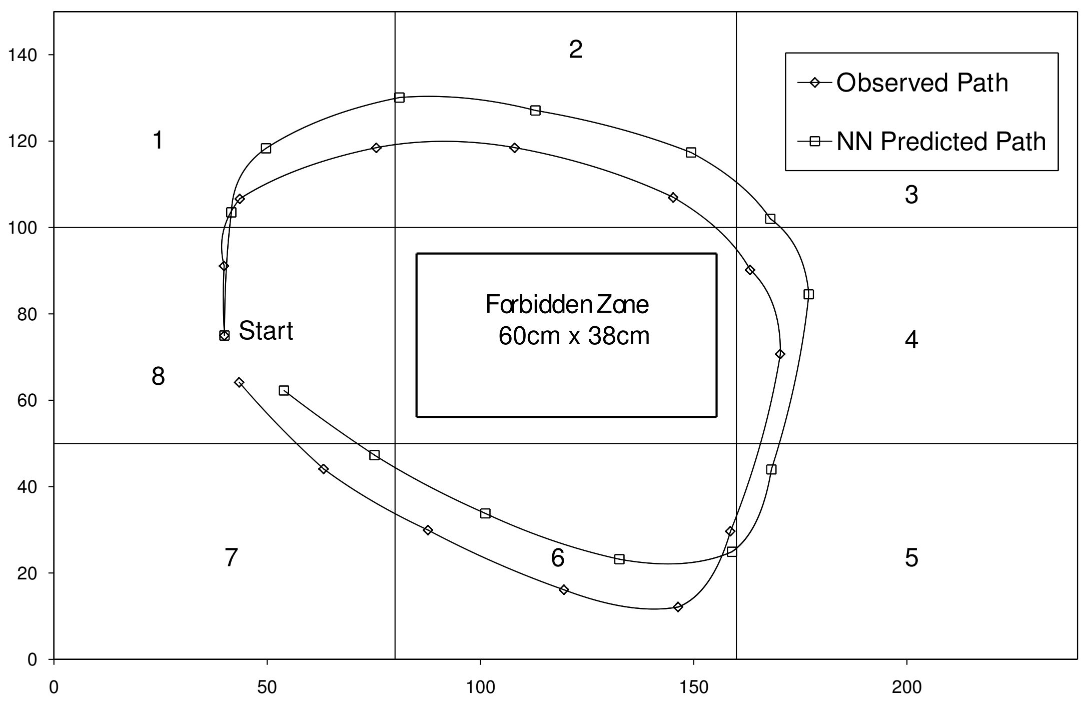

**Figure 7: Predicted and actual robot paths for command set of size 13**

more difficult to find relationships between robotic actions and the effects that these actions have on the robot-environment system

- changes in the morphology of simulator NNs (i.e. number of hidden neurons, activation functions employed etc.) that can possibly improve simulator accuracy and generalization ability
- evolution of more complex robotic control structures, i.e. controller NNs

Although the research presented here is still largely in the developmental phase, it has shown a definite potential for the use of ANN simulators in ER. It is envisaged that implementing a simulator in this way could simplify the process of simulator development for the evolution of more complex behaviours than that evolved in this study. If this is indeed the case, research into currently unexplored applications of ER could become much more achievable through the use of ANN simulators. Investigation into the use of ANN simulators is therefore considered to be of fundamental importance as a result of the possible contributions that such research can make to the advancement of the field of ER.

## 10. REFERENCES

[1] Icommand technology. Website, accessed March 2009. `http://lejos.sourceforge.net/p_technologies/nxt/icommand/icommand.php`.

[2] JCamCalib: A Camera Calibration Utility. v0.7. Website, accessed March 2009. `http://ftp.heanet.ie/disk1/sourceforge/j/jc/jcamcalib/`.

[3] LEGO.com MINDSTORMS NXT Home. Website, accessed March 2009. `http://mindstorms.lego.com`.

[4] R. A. Brooks. New Approaches to Robotics. *Science*, 253(5025):1227–1232, Sep 1991.

[5] J.-F. Dupuis and M. Parizeau. Evolving a Vision-Based Line-Following Robot Controller. In *CRV '06: Proceedings of The 3rd Canadian Conference on Computer and Robot Vision*, page 75, Washington, DC, USA, 2006. IEEE Computer Society.

[6] A. P. Engelbrecht. *Computational Intelligence: An Introduction*. John Wiley & Sons, West Sussex, second edition, 2007.

[7] D. Floreano, P. Husbands, and S. Nolfi. Evolutionary Robotics. In *Handbook of Robotics*. Springer Verlag, Berlin, 2008.

[8] C. Hartland and N. Bredeche. Evolutionary Robotics: From Simulation to the Real World using Anticipation. In *ABIALS*, Rome/Italie, 2006.

[9] 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*, pages 364–373, Cambridge, MA, USA, 1993. MIT Press.

[10] J. H. Holland. *Adaptation in natural and artificial systems*. MIT Press Cambridge, MA, USA, 1992.

[11] H. Hyötyniemi. Turing Machines are Recurrent Neural Networks. In *Proceedings of STeP'96*, pages 13–24. Finnish Artificial Intelligence Society, 1996.

[12] 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*, pages 704–720, London, UK, 1995. Springer-Verlag.

[13] 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*, pages 465–469. SCS Europe, 1998.

[14] H. Lipson, J. C. Bongard, V. Zykov, and E. Malone. Evolutionary Robotics for Legged Machines: From Simulation to Physical Reality. In *Intelligent Autonomous Systems*, pages 11–18, 2006.

[15] H. H. Lund. Co-evolving Control and Morphology with LEGO Robots. In *Morpho-functional Machines*. Springer-Verlag, 2001.

[16] H. H. Lund and O. Miglino. From simulated to real robots. In *Proceedings of IEEE Third International Conference on Evolutionary Computation, NJ*. IEEE Press, 1996.

[17] H. H. Lund, O. Miglino, L. Pagliarini, A. Billard, and A. Ijspeert. Evolutionary Robotics — A Children's Game. In *Proceedings of IEEE 5th International Conference on Evolutionary Computation*, pages 154–158. IEEE Press, 1998.

[18] O. Miglino, O. Gigliotta, M. Ponticorvo, and S. Nolfi. Breedbot: An Edutainment Robotics System to Link Digital and Real World. In *KES '07: Knowledge-Based Intelligent Information and Engineering Systems and the XVII Italian Workshop on Neural Networks on Proceedings of the 11th International Conference*, pages 74–81, Berlin, Heidelberg, 2007. Springer-Verlag.

[19] O. Miglino, H. H. Lund, and S. Nolfi. Evolving Mobile Robots in Simulated and Real Environments. *Artificial Life*, 2:417–434, 1996.

[20] O. Miglino, K. Nafasi, and C. E. Taylor. Selection for wandering behavior in a small robot. *Artif. Life*, 2(1):101–116, 1995.

[21] 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 *CERMA '07: Proceedings of the Electronics, Robotics and Automotive Mechanics Conference*, pages 488–493, Washington, DC, USA, 2007. IEEE Computer Society.

[22] E. D. Sontag. *Mathematical Control Theory: Deterministic Finite Dimensional Systems*. Textbooks in Applied Mathematics. Springer, New York, second edition, 1998.

[23] 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 (ICAIET 2004)*, pages 14–20, Kota Kinabalu, Sabah, Malaysia, 2004.

[24] 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, August 2008.

[25] J. C. Zagal and J. Ruiz-Del-Solar. Combining Simulation and Reality in Evolutionary Robotics. *J. Intell. Robotics Syst.*, 50(1):19–39, 2007.