Robotica: (2020) volume 38, pp. 886–902. © Cambridge University Press 2019
doi:10.1017/S0263574719001140

Complex Morphology Neural Network Simulation in Evolutionary Robotics

Grant W. Woodford* and Mathys C. du Plessis

Department of Computing Sciences, Nelson Mandela University South Campus, Port Elizabeth, South Africa
E-mail: mc.duplessis@mandela.ac.za

(Accepted June 24, 2019. First published online: July 22, 2019)

SUMMARY
This paper investigates artificial neural network (ANN)-based simulators as an alternative to physics-based approaches for evolving controllers in simulation for a complex snake-like robot. Prior research has been limited to robots or controllers that are relatively simple. Benchmarks are performed in order to identify effective simulator topologies. Additionally, various controller evolution strategies are proposed, investigated and compared. Using ANN-based simulators for controller fitness estimation during controller evolution is demonstrated to be a viable approach for the high-dimensional problem specified in this work.

KEYWORDS: Automation; Control of robotic systems; Motion planning; Robot dynamics; Serial manipulator design and kinematics.

1. Introduction

Evolutionary robotics (ER) is a biologically inspired approach for the automatic development of autonomous robotic systems and is a branch of the evolutionary computing field.[1,2] The ER process has been successfully applied to many problems, such as obstacle avoidance, light following, trajectory planning, inverted pendulum stabilisation, evolutionary-added design and many others.[3–6]

A large number of controller evaluations are required during the ER process in order to rank relative fitnesses of controllers. Evaluating a large number of controllers on a real-world robot is often time-consuming and may damage hardware through mechanical wear.[7] In order to overcome these issues, simulation techniques have been used to approximate the performance of controllers.[8]

Creating a physics-based simulator capable of effectively simulating arbitrary robot morphologies is an extremely difficult task.[1] Selecting an existing simulation framework may depend on multiple criteria and no perfect simulator is likely to exist for any given robotic system.[9] Many ER approaches rely on built-in compensation techniques that try to overcome the inherent difficulties in developing effective simulators in ER.[9–14]

Physics-based simulators are an imperfect approximation of real-world systems.[1,7] This imperfection is due to the difficulty in accounting for all the various physical components like weight, friction, inertia and many other aspects. Highly accurate simulators can be developed but may require specialised knowledge on the modelling of complex systems. The development of highly accurate simulators can be financially costly due to the hiring of experts or due to the time spent acquiring such expertise.[7]

The building of physics-based models can be automated through approaches such as the Estimation-Exploration Algorithm.[11] Robot model parameters can be automatically tuned using approaches such as Anytime Learning[12] or the Back-to-Reality Algorithm.[13] The Transferability Approach[15] models weaknesses in an existing simulator so that unreliably simulated behaviours can

* Corresponding author. E-mail: s205014224@mandela.ac.za

https://doi.org/10.1017/S0263574719001140 Published online by Cambridge University Press

be avoided. Simulators can be augmented by mapping behaviours seen in simulation with real-world performance using Intelligent Trial-and-Error Learning.[14] A disadvantage of all these approaches is that they require physics-based simulators. In comparison, model-fitting approaches based on empirical data require little specialised knowledge and are relatively simple to develop.[16,17]

Physic-based simulators model the various underlying features between the robot and the environment. Alternatively, empirical models can be constructed and optimised from empirically collected data. Physics-based simulator development is often complex and time-consuming to construct, and requires specialised knowledge of the robotic system being simulated.[5] Research into possible alternative simulation techniques such as artificial neural network (ANN)-based simulators that are simple to construct and require little specialised knowledge could therefore provide a significant contribution in the ER field.[3–5,10,16,18]

ANN-based simulators shall be referred to as simulator neural networks (SNNs). SNNs are known to enjoy good generalisation abilities, computational efficiency and noise tolerances; require little specialised knowledge to construct; and can result in good transferability of robot behaviours from simulation to reality.[5,10,16]

Prior work investigating the use of SNNs as an alternative to physics-based simulators in ER focused on simple, low-dimensional problems. No known prior research has investigated the effectiveness of SNNs on complex, high-dimensional problems in ER. This paper successfully scales up SNNs for use on a complex, high-dimensional robotics problem.

1.1. Simulator neural networks

SNNs have been successfully utilised in the development of controllers for simple differentially steered mobile robots.[4,5,18] Problems solved include path planning, obstacle avoidance and light-approaching behaviours. The use of SNNs for simulating the behaviours of an inverted pendulum robot has been successfully demonstrated.[4,16] SNNs can serve as a fast surrogate model to a more high-fidelity simulation where evaluations can be time-consuming.[19]

Prior investigations into the use of SNNs for simulating the behaviours of a complex snake-like robot required structured changes to the joint angles.[3,10] Changes to the robot's joint angles were generated based on a sinusoidal mathematical equation.[20] The SNNs predicted the robot's behaviour based on the parameter settings used to generate the changes in the joint angles. Consequently, these SNNs are specific to a single controller design and do not generalise to multiple controller designs. This paper investigates the first generalised SNN designed for a snake-like robot and is capable of simulating behaviours caused by arbitrary changes applied to a robot's joint angles.

1.2. Snake robot control and simulation

Biological snakes are capable of many modalities, such as climbing, swimming, jumping, crawling and many others. Many different snake robot platform configurations exist. Examples of these differences include robots having force sensors, cameras, orientation sensors, passive or active wheels, link length, number joints and many other. These variations make comparisons between snake robot platforms and approaches difficult.

Commonly used controller designs include central pattern generators, curve fitting procedures and parametrised gait generating functions. Useful motions that can be generated include lateral undulation, sidewinding, rolling, shuffling, climbing and various others. However, these locomotion modes are relatively simple and not suited for all situations. A more diverse set of behaviours can be generated using scripted gaits. Scripted gait controllers allow a snake robot to change from any given set of joint angles to any other depending on the range of movement. Certain problems such as climbing stairs could require a carefully crafted sequence of scripted commands in order to be solvable. However, scripted gaits are arguably the most difficult controller design to accurately simulate due to the size of the behavioural search space. There is little known work investigating scripted gait approaches in ER. Scripted gait controller approaches have been proposed in prior work; however, controller generation is constrained using a mathematical function.[21] A Genetic Algorithm has been utilised to evolve scripted gaits in order to transition effectively between snake robot locomotion modes but the study restricted movement to a planar dimension.[22]

In order to simplify the simulator development process or speed up calculations, certain locomotion modes can be estimated through modelling simplifications or heuristics, or surrogate models can be developed. A mapping between the parameters used by a pattern-generating function and the behavioural performance of the robot can be developed using ANNs,[10] Gaussian processes[19] or other

https://doi.org/10.1017/S0263574719001140 Published online by Cambridge University Press

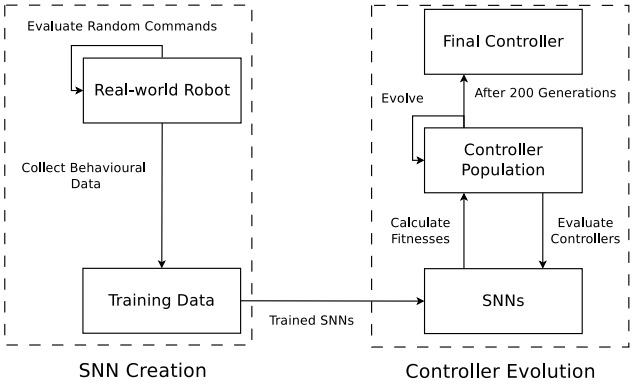


Fig. 1. SNN creation and integration into the ER process.

regression techniques. However, the parameter mapping approach does not model behaviours related to changing parameter values or switching between controller designs. Much research is currently focused on controller designs based on expert knowledge. This paper focuses on scripted gaits with the only limitation applied being on the range of movement in joint angles.

This paper is structured as follows: The development of SNNs and their integration into the ER process is described in Section 2. Section 3 addresses the motivations of this research. Section 4 gives a high-level overview of the experiments and hardware configurations. Section 5 discusses the simulator topology benchmarks. Section 6 covers the controller evolution and real-world performance validations of solutions. Finally, conclusions are discussed in Section 7.

2. SNNs in ER

A high-level overview of the approach in this paper is illustrated in Fig. 1. The process consists of two phases, the SNN creation phase and the Controller Evolution phase. Firstly, randomly generated controllers are evaluated on the real-world robot in order to collect a sufficient amount of training data (SNN creation phase). Once enough training patterns have been collected, the data are used to train SNNs. The SNNs are used to simulate robot behaviours and evaluate the fitnesses of controllers (Controller Evolution phase). After 200 controller evolution generations, the fittest controller in the last generation is declared the final controller solution.

3. Motivation

The use of SNNs as an alternative to physics-based simulation approaches is a paradigm shift from traditional ER thinking. SNN research is in the early stages of automating parts of the ER process that involve specialised human knowledge and intuition. The development of simulators that generalise well to different environments and robots without human intervention is difficult. For certain problems, SNNs may be faster, more accurate and easier to construct than physics-based approaches.[16,23] This is due to SNNs being relatively simple and behaviourally focused rather than built from specialised assumptions. SNNs purposely avoid many of the mathematical complexities required in ER when designing simulators.

Recent Deep Learning techniques can be used to scale up SNNs for more complex, high-dimensional problems. Deep ANNs differ considerably compared to classical ANNs and have significant improvements. Multiple hidden layers often results in superior performance compared to a single hidden layer. The rectified linear unit (ReLU) activation function is more efficient compared to alternate activation functions. Stochastic gradient descent is more efficient at optimising multi-layer ANNs compared to the standard gradient descent approaches. Dropout or other techniques can be used for regularisation.

A goal of this research is to investigate if ANNs can be scaled up to effectively simulate more complex robots. The increased complexity naturally leads to larger ANN topologies, more input features and greater difficulties in training. Deep Learning-based ANNs are investigated in order to eliminate scaling problems associated with more classical ANN approaches. Controller evolution strategies are investigated in order to overcome problems associated with simulating complex, high-dimensional robotic controllers.

https://doi.org/10.1017/S0263574719001140 Published online by Cambridge University Press

This paper addresses several weaknesses experienced in prior research involving SNNs.[3] The following points highlight the main novel contributions of this work that have not been investigated for SNNs and a complex robot:

- Scale up SNNs for use on a complex, high-dimensional problem.
- Investigate simulating a robotic sensor (orientation sensor).
- Develop a generalised simulator capable of handling many different controller designs.
- Investigate the use of uncertainty estimation for avoiding poorly simulated behaviours during controller evolution.
- Investigate the real-world performance impact and properties of several proposed controller evolution approaches.

4. Research Overview

The ideal SNN topologies are chosen based on a comparison of many possible configurations (Section 4.1). Once the final SNN topology configurations are chosen, those SNNs are used to evolve controllers for a particular task. Factors that can affect the real-world performance of evolved controllers are investigated and compared (Section 4.2). The robotic hardware and behaviour-tracking system are discussed in Section 4.3.

4.1. SNN topology benchmarks

Before any controllers can be evolved in simulation, a suitable simulator is built. The ideal topology of the SNNs cannot be known beforehand; therefore, multiple SNN topologies are tested and compared. The number of hidden layers are varied and different layer sizes are benchmarked. The best performing SNN topologies are selected for use in the follow-up experiments investigating controller evolution strategies.

4.2. SNN approaches investigated

Certain factors can influence the real-world performance of evolved controllers, namely, the fitness estimate calculation, the distribution of noise injected into the simulator, boundary limit penalisations in the fitness function and the estimation of uncertainty in predictions. These factors are tested and compared in different experimental configurations.

Controllers are evaluated multiple times in simulation with noise. By default, the average fitness is used to estimate performance of controllers. If the fitness deviations in simulation is considered then the average fitness plus the standard deviation of the fitnesses could be used as the adjusted fitness of controllers.

Adaptive noise is when the noise injected into the simulator is Gaussian with a mean of zero but the standard deviation is proportional to the size of the prediction. For adaptive noise, larger predictions result in more noise being injected into the simulator. The purpose of adaptive noise is to more accurately simulate noise found in reality.

Reducing the simulator and controller search space is hypothesised to increase accuracy in predictions. Improved accuracy improves the ability of behaviours developed in simulation to transferring into reality. The simulator search space for the chosen snake-like robot may be too large to thoroughly explore through randomly generated controllers evaluated in reality. Using dropout to estimate prediction uncertainties has been proposed for ANNs[24] and these uncertainties are integrated into the ER process.

4.3. Hardware and data capture

All experimental work is performed using a custom snake-like robot (Fig. 2) based on similar designs by other researchers.[25,26] The robot consists of an array of 12 Dynamixel AX-12 servo motors. The length of the robot is 114 cm and has a width/height of approximately 5 cm. The joints alternate between moving laterally and vertically. A tethered power source is used in order to avoid power instabilities.

No wheels or snake-like skins are added to the robot. Camera-based colour tracking is used to capture the robot's movement behaviours and an orientation sensor is used to measure the head orientation. Two coloured tracking markers on the head allows the robot to be tracked by a camera.

https://doi.org/10.1017/S0263574719001140 Published online by Cambridge University Press

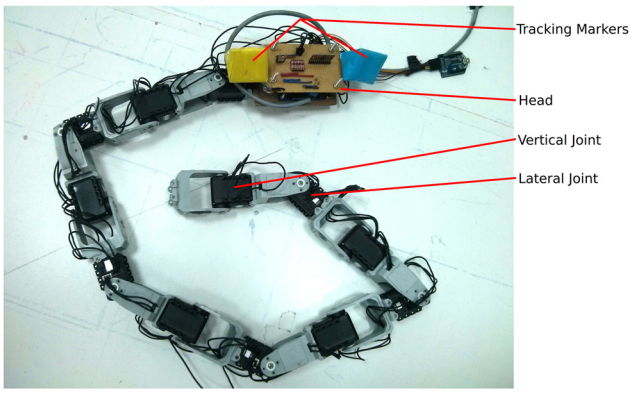


Fig. 2. Snake-robot morphology.

5. SNN Training and Optimisation

This section investigates the performance of different simulator topologies. The benchmarking methodology followed is discussed in Section 5.1. A detailed explanation of the simulator design is given in Section 5.2. Results of the benchmarking experiments are presented (Section 5.3).

5.1. Benchmarking methodology

Benchmarking experiments conducted tested between one and five hidden layers. The number of neurons for any given hidden layer could be either 25, 100, 400 or 800. All possible permutations of hidden layer sizes were investigated. For each behavioural component simulated, the total number of possible SNN configurations is 1364 permutations. Each SNN permutation is trained 30 times independently in order to perform meaningful statistical comparisons between configurations. The final performance outcome of a given SNN configuration is calculated as the average error of the unseen test set. The best SNN topology, statistically better than all other topologies, is chosen for the real-world experiments described in the controller evolution experiments (Section 6).

5.2. The simulator

An example of a controller being evaluated in simulation is illustrated in Fig. 3. Joints on the robot are numbered. The i^{th} command's j^{th} joint has starting and transition angles represented by α_{ij} and β_{ij}, respectively. The i^{th} command contains the robot's current joint angles (α_{ij}) and the transition joint angles (β_{ij}) obtained once the command has finished execution. The starting head orientation before the i^{th} command is executed is also captured (θ_i). Commands executed in simulation produce the behavioural components Δx_i, Δy_i, Δa_i and Δo_i. These components are the relative changes in the position of the head of the Snake robot. The behavioural components for many commands can be pieced together to produce a trajectory.

Commands are randomly generated during the data collection process such that values for α_{ij} and β_{ij} are sourced from a uniform distribution. Commands executed in reality produce the behavioural components Δx_i, Δy_i, Δa_i and Δo_i. During the behavioural data collection process, the SNNs in Fig. 3 are replaced by a real-world robot and the command inputs and outputs produced are captured.

Each SNN is implemented as a standard feedforward neural network (FFNN) with hidden and dropout layers. The simulator consists of four separate FFNNs and is illustrated in Fig. 4. Each FFNN takes the same inputs but predicts a different single output.

The predicted outputs are Δx, Δy, Δa and Δo. All tracked positions and predicted outputs are calculated as the midpoint between the coloured markers on the head of the robot. These outputs estimate the relative changes in the robot's position and head orientation as a result of the giving inputs (Fig. 5). The change in the robot's position relative to the perpendicular direction of the robot's heading is represented by Δx. The heading of the robot is the direction at which the head points, from the yellow to the blue coloured marker (Fig. 2). The change in y displacement (Δy) is the change in the robot's position parallel to the heading. The a displacement (Δa) is the change in the robot's heading relative to the heading before the command has been executed. The head orientation is the angle between the normal of the working surface and the normal of the head of the robot. The change

https://doi.org/10.1017/S0263574719001140 Published online by Cambridge University Press

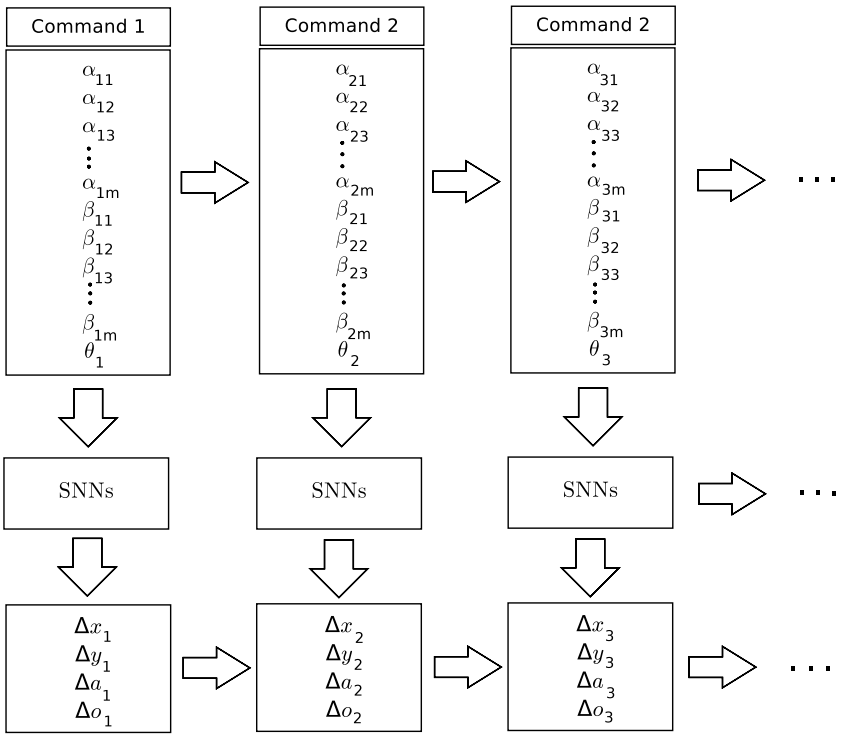


Fig. 3. Evaluating a controller using SNNs.

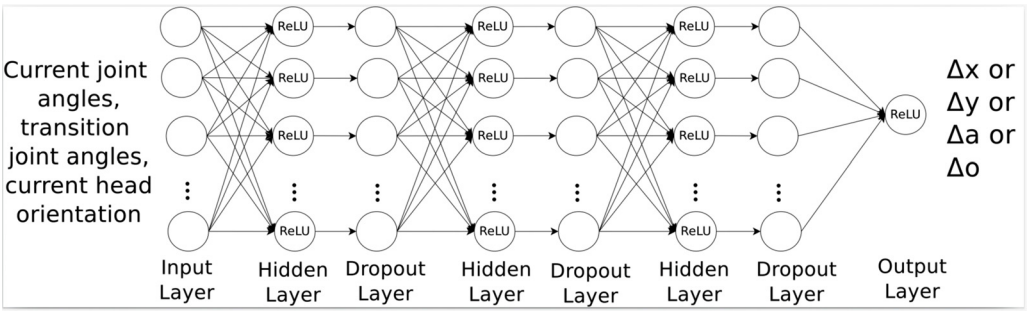


Fig. 4. SNNs of the snake robot.

in the robot's head orientation is represented by Δo. A single command consists of 12 joint angle changes. The inputs to each of the FFNNs are the current joint angles of the robot, the next set of joint angles and the current orientation of the head.

Training data are captured through the real-world evaluation of randomly generated commands (Section 6). A total of 4990 patterns are captured during these evaluations, and the patterns are split into a training, validation and test sets. The data collection phase took approximately 20 h in total, not including rest periods. Data collection was spread over many days with frequent breaks in order to avoid overheating. Eighty per cent of the total patterns were used as a training set, 10% formed the validation set and another 10% was used as a test set. The validation set is used to prevent overfitting during training, and the test set is used to gauge the final performance. The training data are standardised to be a Gaussian distribution with mean zero and a standard deviation of one. All SNNs are trained on a mid-range computer with a graphics card within 5 min.

The FFNNs are implemented and trained using the deep learning libraries called Keras[27] and Tensorflow.[28] FFNNs are trained using the standard optimiser in Keras called adaptive moment estimation with the default parameter settings.[29] Training is performed using the entire training data set, and batch-based training is not utilised. FFNNs are trained for 4000 iterations which would result in

https://doi.org/10.1017/S0263574719001140 Published online by Cambridge University Press

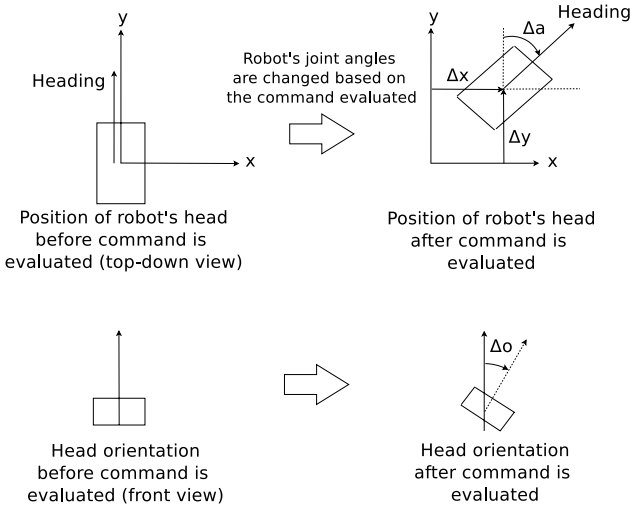


Fig. 5. Behavioural components of the robot's head.

overfitting but checkpointing is used to prevent overtraining. In each iteration, checkpointing saves the FFNN weights whenever the validation error improves. Therefore, only the weights with the lowest validation errors are used in the final models.

A dropout layer is added after every hidden layer. Dropout layers are enabled during training and certain experimental configurations. If dropout layers are enabled, hidden layer nodes are randomly ignored during forward pass calculations. A dropout rate of 50% is used for each hidden layer node. All hidden and output layer nodes make use of the ReLU activation function. During predictions, inputs are specified and the standard FFNN forward pass calculations are applied in order to produce some output value. The output value is integrated into the overall simulated result.

5.3. SNN benchmarking results

The number of permutations for a single hidden layer configuration is 4, namely a hidden layer with either 25, 100, 400 or 800 neurons. Similarly, the number permutations for 2 hidden layers is 16, 3 hidden layers is 64, 4 hidden layer is 256 and 5 hidden layers has a total of 1024 different SNN configurations.

The top performing model configurations (Table I) are chosen based on a statistical comparison of all the possible SNN configurations. Out of the FFNN configurations tested, the ideal number of hidden layers was found to be three. The Mann–Whitney U test is used to determine if two different model configurations are equal in terms of the real-world performance. The significance level chosen for the Mann–Whitney U test is 5%. For each simulation component, the model with the lowest test error is compared for equivalence with the test error of every other model configuration. If there are multiple model configurations with a statistically equal performance, the model with the least number of hidden layers or least number of artificial neurons is chosen. For the x and o displacement components, the models with the lowest test errors were found to perform significantly better than all other model configurations.

The model with the lowest test error for the y displacement component performed significantly better than all other model configurations except two. These two exceptions also had 800 artificial neurons for the first two hidden layers but the third layer of the one model has 100 neurons versus 25 neurons for the top performer. The second exception has a total of four hidden layers and was therefore disqualified. The top performer with three hidden layers is more computationally efficient than the configuration with four hidden layers.

The model with the lowest test error for the a displacement component performed significantly better than all model configurations except one. Both these model configurations have only three hidden layers and 800 artificial neurons for the first two layers. However, the last layer of the one model has 400 neurons while the best configuration only has 100 neurons.

The predicted versus actual displacements of the top performers using the test data set are illustrated in Fig. 6. Linear regression lines are fitted, and coefficient of determinations (R^2) are calculated.

https://doi.org/10.1017/S0263574719001140 Published online by Cambridge University Press

Table I. FFNN topologies.

FFNN component	First hidden layer	Second hidden layer	Third hidden layer
Δx	800	800	100
Δy	800	800	25
Δa	800	800	100
Δo	800	800	25

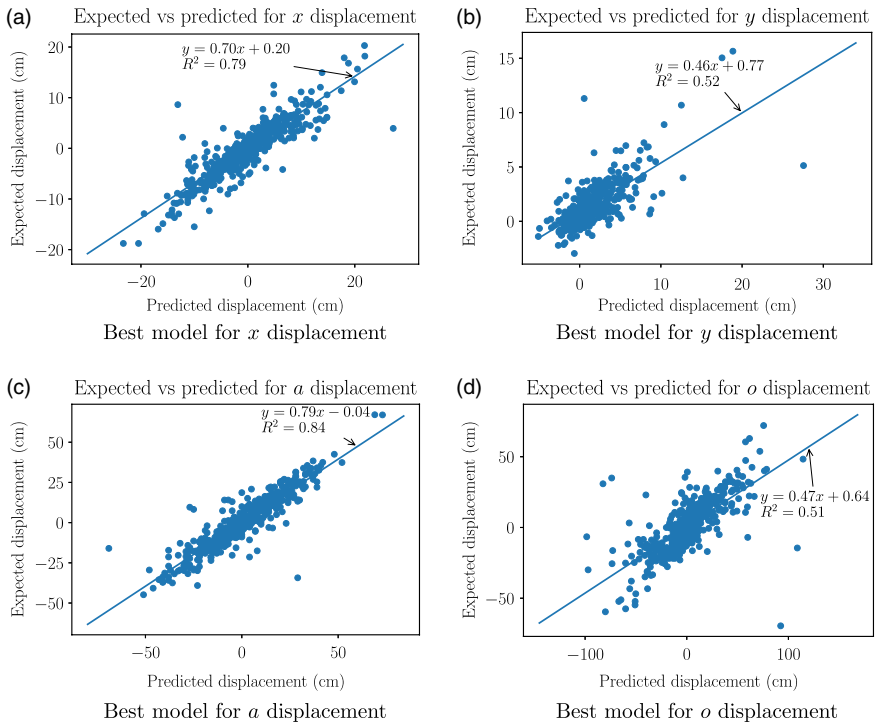


Fig. 6. Top performing models.

The y-intercepts of all regression lines were near zero and obvious trends in the figures can be observed. Based on the slopes of all the linear regression lines, it can be said that the models tend to overestimate the real-world displacements.

The R-squared for the x, y, a and o displacements are 0.79, 0.52, 0.84 and 0.51, respectively. Similarly, the slopes for the x, y, a and o displacements are 0.70, 0.46, 0.79 and 0.47, respectively. There are a few outliers for all displacements, but the predictions are sufficiently accurate to capture the dynamics involved.

The number of neurons required for the first two hidden layers of the optimal configurations were all 800, which is the maximum number of neurons any single layer could posses in the tested configurations. This could indicate that increasing the maximum number of neurons per layer could achieve even better results. The maximum number of neurons is capped in order to keep the computational expense relatively low and runnable on low-end hardware. The last hidden layer for each component favoured a relatively small number of neurons.

6. Controller Evolution

The controller design is discussed in Section 6.1. An overview of the methodology is discussed in Section 6.2. Aspects regarding controller evolution and the different approaches tested are given in Section 6.3. The real-world performance of the controller evolution investigations are presented in Section 6.4 and a detailed look at notable controllers is presented (Section 6.5). Lastly, the controller evolution results are discussed in Section 6.6.

https://doi.org/10.1017/S0263574719001140 Published online by Cambridge University Press

6.1. Controllers

Behavioural tracking using coloured markers mounted on the head meant that the robot could not be tracked when the robot turned upside down during controller evaluations. Controllers were evolved to remain relatively upright and trackable. Controllers that resulted in the robot being unable to be tracked for more than two commands are considered a failure.

The task chosen for the experimental work is to maximise the distance travelled by the robot. At the start of any given controller evaluation, the robot is completely straight and upright. A controller consists of a set of 13 commands. These 13 commands are repeated 3 times in sequence when evaluated to produce a cyclical gait. Any given command consists of a list of 12 joint angle changes to be applied to each of the robot's respective joints. When a command is evaluated, all joint angles are simultaneously adjusted according to the change specified in the command. Once all the specified joint angles are reached, the next command can be executed. The joint resolution of motors is approximately 0.3°. A change in a joint angle is encoded as an integer between −100 and 100 (−30° to 30°).

The change in the angle of any given joint per command is between −30° and 30°. The range of possible joint angles for lateral joints are between −60° and 60° from the starting position. Similarly, vertical joint angles are limited to be between −30° and 30° from the starting position. In order to reduce the number of failures due to the robot turning upside down, the lateral joint limits are larger than the vertical joint limits. Each command is executed in approximately 15 s with slight variations in execution times depending on the torques applied. A 3-s break is present between the execution of commands in order to ensure position stability.

Algorithm 1 is used to estimate the fitness value of every controller in simulation. The algorithm took as input the simulated path positions of the head of the robot associated with the commands evaluated. A list of controller commands evaluated is also included. The fitness is based on two components. The first component is the Euclidean distance between the starting and final positions of the robot's head in simulation for the evaluated controller. Positions are calculated as the midpoint between the tracking markers on the head of the robot. The second component is an accumulation of penalties applied based on the evaluated commands. A penalty is applied per command if any of the boundary limits specified in Table III are violated or the simulated prediction of the command is classified as uncertain. A command is classified as uncertain if for any of its predicted behavioural components, the predicted standard deviation (based on the dropout uncertainty) is greater than a specific uncertainty threshold. The uncertainty threshold for each behavioural component is calculated as the standard deviation of the test data set errors. The fitness value is a ratio of the Euclidean distance travelled and accumulated penalties. The goal of the ER process is to minimise penalties and increase the Euclidean distance travelled.

Algorithm 1 Snake controller fitness evaluation

Require: *positions* ← Simulated path list and *commands* ← List of controller commands
  distance ← Distance between start and end of *positions*
  penalties ← 1
  c ← A large constant
  v ← A large constant
  for each command in *commands* **do**
    if command has reached a boundary limit **then**
      $penalties \leftarrow penalties + c$
    end if
    if command is of high uncertainty **then**
      $penalties \leftarrow penalties + v$
    end if
  end for
  $fitness \leftarrow distance/penalties$
  return *fitness*

Controller are evaluated five times in simulation with noise and the average fitness becomes the controller's fitness. A Gaussian distribution with a mean of zero is used for generating noise. The

https://doi.org/10.1017/S0263574719001140 Published online by Cambridge University Press

Table II. Experimental configuration.

Approach	Uncertainty	Fitness deviation	Adaptive noise	Boundary limit
Baseline (B)	No	No	No	Default
Approach 1 (A1)	Yes	No	No	Default
Approach 2 (A2)	No	Yes	No	Default
Approach 3 (A3)	No	No	Yes	Default
Approach 4 (A4)	No	No	No	Reduced
Approach 5 (A5)	Yes	No	No	Reduced
Approach 6 (A6)	Yes (strict)	No	No	Default
Approach 7 (A7)	Yes (strict)	No	No	Reduced

standard deviation is based on the difference between the simulator predictions and the unseen test data set. The standard deviation of the distributions used to generate noise are 3.03 cm for Δx, 2.00 cm for Δy, 7.17° for Δa and 15.49° for Δo.

6.2. Methodology

The size of the controller population used during the ER process is 400. Before the controller evolution process, the initial population of controllers is generated randomly from a uniform distribution. Tournament selection is used as the crossover selection operator and the tournament size used 50% of the controller population. A two parent crossover method is employed where each parent contains the same number of commands. The mutation rate is the probability of changing any given change in the joint angle by a random amount and is set to a probability of 10%. The size of any given joint perturbation due to a mutation operation came from a uniform distribution and could be between −15° and 15°. The ER process is conducted for precisely 200 generations upon which the fittest controller is selected from the final generation population.

6.3. Robot experiments

The performance metric used to evaluate how well controllers performed in reality are calculated as the projection of the real-world final position vector onto the simulated final position vector. The optimal parameters for the ER process were experimentally determined based on prior work.[10]

Eight controller evolution approaches were tested independently. Each approach is presented as a row in Table II. For each approach, 30 independent ER runs were conducted.

A1, **A5**, **A6** and **A7** use a technique to estimate the level of real-world accuracy of any SNN prediction. During the ER process, SNNs make predictions while the dropout layers are enabled. Thirty independent predictions are obtained for any given SNN command, producing a distribution of predictions. The mean and standard deviation of these predictions is calculated. The mean becomes the actual prediction while the standard deviation estimates the level of uncertainty and penalises a controller's fitness per command if the standard deviation is too high. For **A1** and **A5**, if a prediction's standard deviation is more than the standard deviation of the component's noise distribution, the controller's fitness is penalised. Similarly for **A6** and **A7**, penalisations occur when the standard deviation of a predictions are more than half the standard deviation of the components noise distribution. These penalisations are additionally explained in Algorithm 1.

Noise is injected into each SNN prediction. Therefore, multiple controller evaluations in simulation will yield different paths due to noise. Controller fitness is typically taken as the mean fitness achieved through multiple controller evaluations in simulation. However, fitness estimates for controllers in **A2** are calculated as the average controller fitness in simulation plus the standard deviation. The purpose of integrating the fitness deviation into the ER process is to determine the effects of punishing controllers that vary greatly due to noise.

To add standard simulator noise, every predicted output on an SNN model is modified by a random amount generated from a Gaussian distribution with a mean of zero and fixed standard deviation. Adaptive noise modifies this standard approach by having a variable standard deviation instead of a fixed one. **A3** made use of an adaptive noise technique. The noise generation uses a Gaussian distribution $N(0, x_p * \sigma)$, where x_p is the predicted displacement and σ is the adaptive standard deviation

https://doi.org/10.1017/S0263574719001140 Published online by Cambridge University Press

Table III. Controller boundary limits.

	Orient. limits	**Δ joint limits**	**Δ*x* limits**	**Δ*y* limits**	**Δ*a* limits**	**Δ*o* limits**
Default	[−90°, 90°]	[−30°, 30°]	None	None	None	None
Reduced	[−45°, 45°]	[−20°, 20°]	[−20, 20]	[−10, 10]	[−25°, 25°]	[−25°, 25°]

Table IV. Statistics for the real-world experiments.

Approach	**Failure rate**	**Performance mean (cm)**	**Performance median (cm)**	**Standard deviation (cm)**
Baseline	37%	19.3	13.6	23.8
Approach 1	30%	22.4	6.0	34.3
Approach 2	30%	22.3	16.1	28.3
Approach 3	67%	33.2	31.5	29.7
Approach 4	10%	37.4	39.1	23.4
Approach 5	14%	29.5	27.9	21.1
Approach 6	17%	30.4	21.2	33.6
Approach 7	10%	30.0	31.3	18.6

constant for the particular component being simulated. Each predicted component had a different standard deviation constant (σ) which was calculated by determining the standard deviation of the following equation applied to each test pattern:

$$(x_p - x_a)/x_p$$

where x_p is the predicted displacement and x_a is the actual displacement. Test patterns are not used for the above calculation when the predicted displacement is less than one standard deviation of the test error or the predicted absolute displacement is less than the actual absolute displacement. The standard deviation constants used for noise generation for the components Δx, Δy, Δa and Δo are 0.42, 0.44, 0.37 and 0.37, respectively.

The boundary limits of controllers are reduced for **A4**, **A5** and **A7**. The default and reduced boundary limits are given in Table III. By default, a controller's fitness is only penalised if the simulated orientation is outside the range of −90° to 90° and −45° to 45° when reduced (relative to the upright position). The default change in joint angles for any given command are between −30° and 30° and between −20° and 20° when reduced. By default, predicted outputs of SNNs are not penalised by the fitness function. When boundary limits are reduced, SNN predictions (Δx, Δy, Δa, Δo) are penalised according to the limits specified in Table III.

6.4. Robot experiment results

The results of controller evolution experiments are summarised in Table IV and illustrated in a standard box-and-whisker plot in Fig. 7. Results for each approach are constructed from 30 independent ER runs. Each independent ER run is performed on a mid-range computer with a graphics card. A single ER run can take approximately 15 min for approaches without uncertainty estimation or 32 min for approaches with uncertainty estimation. The failure rates given in Table IV are the percentage of solution controllers that turned upside down at some point during the real-world evaluation. The distance for all failed controllers is calculated using the last captured position just before the failure occurred.

The performance difference between approaches is statistically confirmed using the Friedman statistical test. The p value of the Friedman test is zero when rounded to the fifth decimal place and therefore indicates a highly significant difference between the performances. A Nemenyi *post hoc* test is performed in order to identify which configurations are significantly different in terms of performance from each other. The calculated p values for the *post hoc* analysis are given in Table V. Rows and columns represent the different approaches (Table II). The significance level chosen for the test is 5%.

https://doi.org/10.1017/S0263574719001140 Published online by Cambridge University Press

Table V. The *p* values for comparisons between approaches.

	B	**A1**	**A2**	**A3**	**A4**	**A5**	**A6**
A1	0.99844	-	-	-	-	-	-
A2	0.99998	0.99998	-	-	-	-	-
A3	8.7e-07	2.6e-05	5.0e-06	-	-	-	-
A4	6.7e-12	6.1e-10	6.7e-11	0.62561	-	-	-
A5	0.00369	0.03364	0.01184	0.66133	0.00983	-	-
A6	0.00301	0.02851	0.00983	0.69614	0.01184	1.00000	-
A7	0.00052	0.00671	0.00198	0.91175	0.04631	0.99976	0.99989

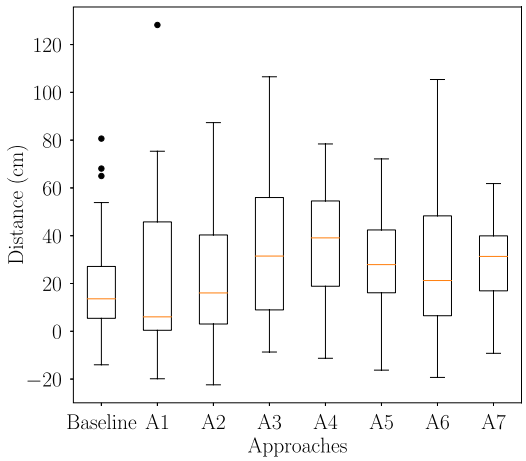


Fig. 7. Performance distribution box plots.

A1, **A2** and the Baseline have no significant differences in terms of overall performance. There is a significant performance difference between **A4** and all other approaches except **A3**. **A3**–**A7** perform significantly better when compared to **A1**, **A2** and the Baseline. The performances observed between **A5**, **A6** and **A7** were found to not significantly differ when compared to each other. If only a strict uncertainty estimation is used (**A6**), the average performance of controllers is not improved by additionally restricting the boundary limits.

Out of all the approaches, the Baseline has the second highest failure rate at 37%. The only approach that resulted in a worse failure rate compared to the Baseline is **A3**, which utilised adaptive noise. The Baseline obtained the lowest average performance and the second lowest mean performance. In Fig. 7, it can be seen that the Baseline only discovered three controllers with a performance metric greater than 60 cm.

A1 achieved a slightly lower failure rate (30%) compared to the Baseline. The average performance is improved compared to the Baseline but the median performance is less than half that of all other approaches. Five controllers were found with a distance metric greater than 60 cm.

A2 integrates the standard deviation of the controller fitness estimates into the final calculated controller fitness. The failure rate and mean performance for **A2** are quite similar to **A1**. Four controllers had a distance metric greater than 60 cm.

A3 has the highest failure rate. Adaptive noise appeared to cause more extreme movements which often cause failures. There are five controllers with a distance metric greater than 60 cm.

For **A4**, the reliability of the developed controllers significantly improves. This approach achieved one of the lowest failure rates at 10%. **A4** also achieved the highest average and median performance. The performance of the best solutions are somewhat more bounded to a lower upper level of performance compared to other approaches.

The failure rate for **A5** was amongst the lowest of all the approaches at 14%. The overall performance is worse than **A4**. This may be due to the combination of reduced boundary limits and uncertainty estimation reducing the search space.

https://doi.org/10.1017/S0263574719001140 Published online by Cambridge University Press

Table VI. Performance metrics of notable controllers.

Solution	Projected distance (cm)	Real-world distance (cm)	Simulated distance (cm)	Heading error (degrees)
(a)	81.3	81.4	54.7	37
(b)	67.8	69.2	66.3	32
(c)	78.4	79.2	112.5	30
(d)	72.1	72.5	131.6	2
(e)	68.1	68.3	67.0	48
(f)	47.0	52.2	44.1	7

For **A6**, the overall performance of controllers is significantly higher compared to **A1** which uses a less strict uncertainty punishment. The failure rate is also lower, at 17%. **A6** has the highest number of controllers (6 controllers) with a distance metric above 60 cm.

A7 has one of the lowest failure rates at 10% and also has the lowest standard deviation in the performances. Only one controller has a distance metric above 60 cm.

6.5. Notable controllers

Notable controllers demonstrating interesting behaviours and good transferral from simulation to reality are illustrated in Fig. 8. At the start of a controller evaluation, the head of the robot is positioned at the origin with the rest of the body along the positive x-axis. The arrows pointing at the origins represent the initial heading of the robot at the start of a controller evaluation. The other arrows represent either the simulated or real-world headings of the robot in the final position of the controller evaluation.

Controllers illustrated in Fig. 8(a) and (b) appear to use similar behavioural strategies. The direction and curvature are similar and are displaying a one step back, two steps forwards movement cycle. The loops generated by the **A1** solution are relatively more open compared to the **A2** solution. The mismatch between the final headings of the solutions are quite similar and likely due to the accumulation of prediction errors over time. The **A2** solution (Fig. 8(a)) integrates the standard deviations of the simulator fitnesses into the final controller fitness while the **A1** (Fig. 8(b)) uses uncertainty penalties.

Figure 8(c) and (d) demonstrated similar behaviours. The direction and curvature of both solutions are an almost mirror image along the horizontal axis. The simulator overestimates the distance travelled by the **A5** solution by almost a factor of two. For the **A4** solution, the simulated and real-world distances travelled are a closer match but the simulator significantly underestimates the change in heading. The similarity could be due to both approaches using search space reduction which tend to generate similar stable behaviours.

In Fig. 8(e), the controller makes use of a set of commands that generate wide opposing movements, forming an interesting zigzag motion. The overall movement is towards the bottom right corner. The simulated and real-world behaviours match closely with the simulator slightly underestimating the size of the loops. The mismatch between the final simulated and real-world headings is large. Inaccuracies are likely due to the accumulation of prediction errors and real-world slippage.

Lastly, Fig. 8(f) is interesting due to similar behaviours observed in simulation and reality. The simulator slightly misjudges the direction of the resulting movement but is still able to accurately predict the distance travelled and the general behaviour. The simulated and real-world loops in the paths generated appear to be quite different but the overall behaviour does not differ significantly. This is one of the few controllers to be observed to have a direct downward trajectory and not also the movement towards the right.

Table VI presents performance metrics of the controller solutions illustrated in Fig. 8. The projected distance column is the projected distance of the real-world final position vector onto the simulated final position vector's direction. Real-world and simulated distances are the Euclidean distances between the start and final positions of the head of the robot in reality and simulation, respectively. Heading error is the difference between the robot's heading in simulation and reality when in the final position.

https://doi.org/10.1017/S0263574719001140 Published online by Cambridge University Press

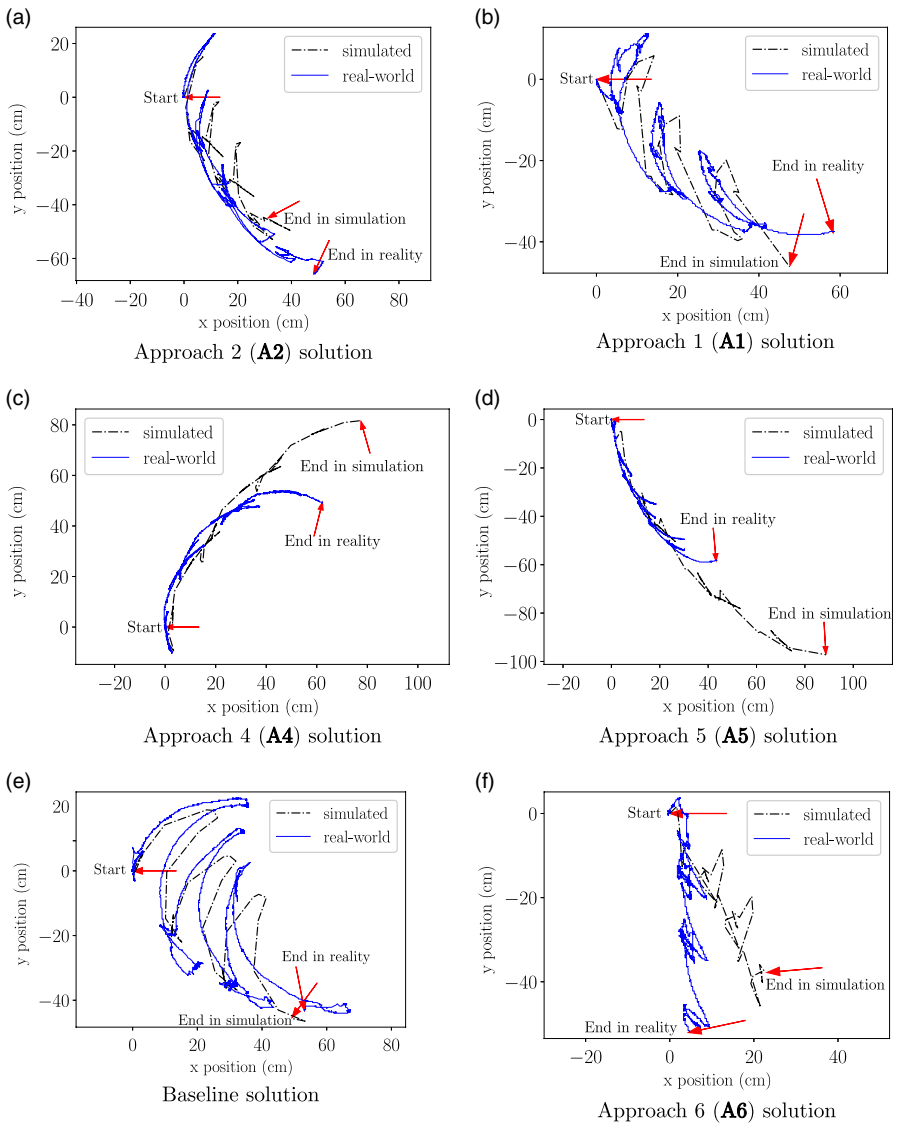


Fig. 8. Examples of discovered controller solutions.

Figure 8(a) has a simulated distance significantly less than the projected or real-world performance. Typically the simulated distance tends to be overestimated. The simulated distances for controllers shown in Fig. 8(c) and (d) are significantly overestimate while Fig. 8(b), (e) and (f) has relatively accurate simulated distances. Most of the controller solutions underestimate in simulation the amount of change in the robot's heading with the exception of controllers shown in Fig. 8(d) and 8(f).

Figure 9 shows a series of frames over time for the controller evaluated in Fig. 8(f). This controller only achieves a performance metric of approximately 50 cm, yet demonstrates a set of complex behaviours[1]. The time gap between frames is 60 s intervals. The robot is completely straight and upright at the beginning of the evaluation. In the first 60 s, the tail end on the right moves downward and the middle of the body is then dragged downwards. At the 120 s mark, the head of the robot begins moving downward too. The robot straightens slightly by the 240 s mark. This pattern of movement repeats for another two cycles and gradually moves the body downwards. The movements are relatively slow to reduce slipping. The robot still experiences slippage and maintains a curved shape that appears to prevent the robot from tipping onto its side during movements.

[1]Video demonstration: https://youtu.be/TdbqiFsNHNc

https://doi.org/10.1017/S0263574719001140 Published online by Cambridge University Press

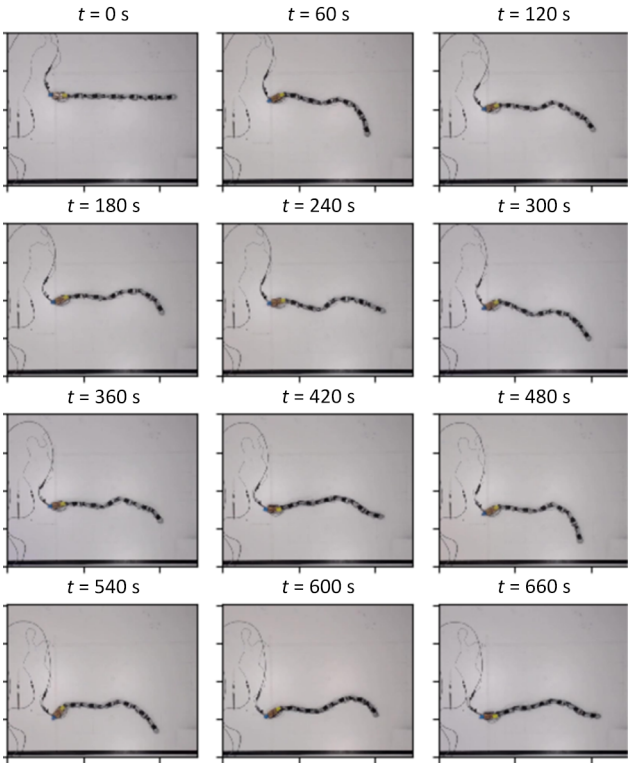


Fig. 9. Time lapse of a solution in Approach 6.

6.6. Discussion

If a controller needs to be found quickly and performance does not need to be optimal, applying boundary limits is a good strategy. Alternatively, if there is a need for high-performance solutions and time is not too limited, then a better strategy would be to use strict uncertainty punishments. Adaptive noise should be avoided due to the high failure rate.

The performance difference between normal noise and adaptive noise indicates that the ER process is sensitive to the type of noise applied. For adaptive noise, a higher standard deviation of applied noise for larger movements appears to reward riskier behaviours that are more likely to result in failures.

7. Conclusions

Prior work demonstrated that using SNNs for simulating complex snake-like robot behaviours is a viable approach when the simulator and controller designs have been simplified. The primary contribution of this paper is the scaling up of the prior work for use on a complex, generalised controller design. This research demonstrates that for the first time, snake-like robot behaviours generated from arbitrary changes in joint angles can be successfully simulated using SNNs and these behaviours transfer well into reality.

The techniques investigated in this research identifies factors influential to the likely performance of developed controllers. Noise generation, uncertainty predictions and limiting the controller search space are influential factors affecting the performance distributions of evolved controllers. Results found that estimating the accuracy of simulator predictions during the ER process and punishing uncertain robot behaviours in simulation significantly improves the likelihood of discovering high-performance controllers. Using dropout uncertainty estimation in ER is a novel technique proposed and demonstrated in this research. Limiting the controller search space improves the consistency and reliability of controllers but negatively affects the likelihood of discovering high-performance solutions.

There are limitations in using SNNs over physics-based approaches. No physical insights into the robotic system are readily available to the extent a physics-based approach would provide. Future work could investigate methods for extracting physical insights from SNNs. Current SNN designs are

https://doi.org/10.1017/S0263574719001140 Published online by Cambridge University Press

unable to handle interactions between multiple robots. However, limitations could be addressed by developing new SNN approaches or hybrid designs integrating SNNs and physics-based approaches could be investigated. Future research should investigate the use of SNNs in multi-agent systems where robots could cooperate in some fashion. SNN training, data gathering and controller evolution could be distributed amongst a decentralised multi-agent system. The work presented in this paper is the early foundational core for more advanced work.

Acknowledgements

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

References

1. J. Bongard, "Evolutionary robotics," *Commun. ACM* **56**(8), 74–83 (2013).
2. S. Nolfi and D. Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines* (MIT Press, Cambridge, MA, 2000).
3. G. Woodford, M. du Plessis and C. Pretorius, "Evolving Snake Robot Controllers Using Artificial Neural Networks as an Alternative to a Physics-Based Simulator," *2015 IEEE Symposium Series on Computational Intelligence*, Cape Town, South Africa (2015) pp. 267–274.
4. C. Pretorius, M. du Plessis and C. Cilliers, "Simulating robots without conventional physics: A neural network approach," *J. Intell. Robotic Syst.*, Springer **71**(3–4), 319–348 (2013).
5. C. Pretorius, M. du Plessis and J. Gonslaves, "A Comparison of Neural Networks and Physics Models as Motion Simulators for Simple Robotic Evolution," *2014 IEEE Congress on Evolutionary Computation (CEC)*, Beijing, China (2014) pp. 2793–2800.
6. S. Prabhu, R. Seals, P. Kyberd, and J. Wetherall, "A survey on evolutionary-aided design in robotics," *Robotica* **36**(12), 1804–1821 (2018).
7. D. Floreano and F. Mondada, "Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot," **In:** D. Cliff, P. Husbands, J.-A. Meyer, S. Wilson (eds.), *From Animals to Animats 3*, (MIT Press, Cambridge, MA, 1994) pp. 421–430.
8. O. Miglino, H. Lund and S. Nolfi, "Evolving mobile robots in simulated and real environments," *Artificial Life* **2**(4), 417–434 (1995).
9. J. Mouret and K. Chatzilygeroudis "20 Years of Reality Gap: A Few Thoughts About Simulators in Evolutionary Robotics," *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, Berlin, Germany (2017) pp. 1121–1124.
10. G. Woodford, M. du Plessis and C. Pretorius, "Concurrent controller and Simulator Neural Network development for a snake-like robot in Evolutionary Robotics," *Robot. Auton. Syst.*, **88**, 37–50 (2017).
11. J. Bongard, V. Zykov and H. Lipson, "Resilient machines through continuous self-modeling," *Science*, Elsevier **314**(5802), 1118–1121 (2006).
12. J. Grefenstette and C. Ramsey, "An Approach to Anytime Learning," *Mach. Learn. Proc.* 189–195 (1992).
13. J. Zagal and J. Ruiz-Del-Solar, "Combining simulation and reality in evolutionary robotics," *J. Intell. Robot. Syst.* **50**(1), 19–39 (2007).
14. A. Cully, J. Clune, D. Tarapore and J. Mouret, "Robots that can adapt like animals," *Nature* **521**(7553), 503–507 (2015).
15. S. Koos, J. Mouret and S. Doncieux, "The transferability approach: Crossing the reality gap in evolutionary robotics," *Evol. Computat.* **17**(1), 122–145 (2013).
16. C. Pretorius, M. du Plessis and J. Gonslaves, "Neuroevolution of Inverted Pendulum Control: A Comparative Study of Simulation Techniques," *J. Intell. Robotic Syst.* **86**(3–4), 419–445 (2017).
17. R. Nardi and O. Holland, "Coevolutionary modelling of a miniature rotorcraft," *Intell. Auton. Syst.* **10**, 364 (2008).
18. G. Woodford, C. Pretorius and M. du Plessis, "Concurrent controller and Simulator Neural Network development for a differentially-steered robot in Evolutionary Robotics," *Robot. Auton. Syst.* **76**, 80–92 (2016).
19. J. Koopaee, B. Van Huijgevoort, C. Pretty and X. Chen, "Parameters tuning of snake robots sidewinding gait using Bayesian optimization," *2018 4th International Conference on Control, Automation and Robotics (ICCAR)*, Auckland, New Zealand (2018) pp. 43–49.
20. K. Melo, M. Hernandez and D. Gonzalez, "Parameterized Space Conditions for the Definition of Locomotion Modes in Modular Snake Robots," *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Guangzhou, China (2012) pp. 2032–2038.
21. K. Dowling, "Limbless locomotion: Learning to crawl," *Proceedings of the IEEE International Conference on Robotics and Automation, 1999*, Detroit, Michigan, USA, vol. 4 (1999) pp. 3001–3006.
22. 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," *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2002*, Washington, DC, USA, vol. 2 (2002) pp. 1507–1512.

https://doi.org/10.1017/S0263574719001140 Published online by Cambridge University Press

23. C. Pretorius, M. du Plessis and J. Gonsalves, "Evolutionary robotics applied to hexapod locomotion: A comparative study of simulation techniques," *J. Intell. Robotic Syst.* 1–23 (2019).
24. Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Insights and applications," *Deep Learning Workshop, ICML* (2015).
25. K. Melo, L. Paez and C. Parra, "Indoor and Outdoor Parametrized Gait Execution with Modular Snake Robots," *IEEE International Conference on Robotics and Automation*, St. Paul, MN, USA (2012).
26. S. Murata and H. Kurokawa, "Self-reconfigurable robots," *Robot. Autom. Mag.* **14**(1), 71–78 (2007).
27. F. Chollet, "Keras," *Github* (2015). https://github.com/keras-team/keras
28. A. Martín and A. Ashish, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," (2015). https://www.tensorflow.org
29. D. Kingma and J. Ba, "Adam: A method for stochastic optimization," (2014). *arXiv preprint arXiv: 1412.6980*.

https://doi.org/10.1017/S0263574719001140 Published online by Cambridge University Press