Proceedings of the

# 4th International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2011)

Friday, December 9 2011
Berlin, Germany

*Nils T Siebel and Yohannes Kassahun*

http://www.erlars.org/

Proceedings of the 4th International Workshop on Evolutionary and Reinforcement Learning
for Autonomous Robot Systems (ERLARS 2011)
Editors: Nils T Siebel and Yohannes Kassahun

ISSN 2190-5576 (Print)
ISSN 2190-5746 (Internet)

Published and Printed by
Nils T Siebel
Wilmsstr. 5
10961 Berlin
Germany
1st Edition, December 2011

## Table of Contents

# Towards Learning Robots

## A Welcome Message from the Chairs

We would like to welcome you to the 4th International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems, ERLARS 2011. This year's ERLARS workshop takes place at the HTW University of Applied Sciences in Berlin, Germany on December 9 2011.

The ERLARS workshop is concerned with research on efficient algorithms for evolutionary and reinforcement learning methods to make them more suitable for autonomous robot systems. The long term goal is to develop methods that enable robot systems to learn completely, directly and continuously through interaction with the environment. In order to achieve this, methods are examined that can make the search for suitable robot control strategies more feasible for situations in which only few measurements about the environment can be obtained.

The articles contained in these proceedings are steps along this way. We hope that they can serve as a useful set of ideas and methods to achieve the long term research goal.

Six papers have been accepted for the ERLARS 2011 workshop. Three papers discuss algorithms that have been used on real robots, three other papers are concerned with the developments of efficient algorithms that can be applied to real robots. In order to give researchers a chance to discuss their work at an early stage this proceedings volume also includes short papers / research statements.

We would like to thank the program committee members who provided very good and helpful reviews. We are also especially indebted to the authors of the articles sent to this workshop for providing the material to make us think and discuss.

It has been a great pleasure organising this event and we are happy to be supported by such a strong team of researchers. We sincerely hope that you enjoy the workshop and we look forward, with your help, to continue building a strong community around this event in the future.

Nils T Siebel and Yohannes Kassahun, Chairs, ERLARS 2011 Workshop.

# Organisation of the ERLARS 2011 Workshop

## Workshop Chairs

Nils T Siebel
Building Automation Lab
Department of Engineering I
HTW University of Applied Sciences
Berlin, Germany

Yohannes Kassahun
Research Group Robotics
DFKI Lab Bremen
University of Bremen
Bremen, Germany

## Programme Committee

**Stéphane Doncieux**, Institut des Systèmes Intelligents et de Robotique, Université Pierre et Marie Curie, Paris, France.

**Peter Dürr**, Sony Systems Technologies Laboratories, Tokyo, Japan.

**Lutz Frommberger**, Cognitive Systems Group, University of Bremen, Germany.

**Faustino Gomez**, Swiss AI Lab IDSIA, Lugano, Switzerland.

**Todd Hester**, Department of Computer Science, University of Texas at Austin, USA.

**Christian Igel**, The Image Group, Department of Computer Science, University of Copenhagen, Denmark.

**Jean-Baptiste Mouret**, Institut des Systèmes Intelligents et de Robotique, Université Pierre et Marie Curie, Paris, France.

**Jan Peters**, Intelligent Autonomous Systems Group, Computer Science Department, Technical University Darmstadt, Germany.

**Daniel Polani**, Department of Computer Science, University of Hertfordshire, Hatfield, UK.

**Stefan Schiffer**, Knowledge-based Systems Group, Department of Computer Science, RWTH Aachen University, Germany.

**Richard S Sutton**, Department of Computing Science, University of Alberta, Edmonton, Canada.

**Sergiu-Dan Stan**, Department of Mechanisms, Precision Mechanics and Mechatronics, Technical University of Cluj-Napoca, Romania.

**Matthew E Taylor**, Department of Computer Science, Lafayette College, Easton, USA.

4th International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2011)
Berlin, Germany, December 9 2011

# About the Transformation between Output and Weight Space in Neural Computation

**Jörn Fischer**[1], **Thomas Ihme and Andreas Knecht**

**Abstract.** Time-discrete recurrent neural networks are complex dynamical systems. In the research area of the synthesis and analysis of such networks it would be desirable to have an algorithm which offers the ability to transform a given neural trajectory directly into a set of neural weights. Though non-linear transfer functions seem to aggravate this problem crucially, such an algorithm is found using the inverse of the transfer function and a least squares approach to find the optimal weights. This constructive method is presented in detail and discussed on different examples.

## 1 Introduction

Time-discrete recurrent neural networks offer a wide field of research area covering a large number of different disciplines. Though the subject of these disciplines might differ a lot, most of them have in common that they try to determine coherences between the topology of a network including the strength of neural interconnection and its behavior. Non-linearities of the neural transfer function aggravate the exploration of such coherences and often methods as gradient descent [7, 8], or evolutionary techniques are chosen to find network structures which result in a desirable predetermined mapping between input and output neurons.

On the other hand, fixed network structures of recurrent neural networks are analyzed to understand their behavior. Detailed description is only found for few types of networks, especially for those with symmetric weight matrices [4, 1, 2], for small neural entities [9, 10, 11, 6], or for very large networks called reservoirs [5]. In this context it could be helpful if we could proof that a network with a given attractor exists. It would be desirable if we could analyze the influence of phase shifts between different neural output, if we could change the amplitude of a trajectory or if we could prove or disprove the existence of a network with given trajectory.

The following article describes a method to transform a given n-dimensional trajectory into a weight matrix of n neurons. Finally the method is discussed and demonstrated on different examples.

## 2 Basic definitions

The following equations are given to clarify the mathematical base. The activity of a discrete-time neuron is defined as follows:

$$\varphi_j^{t+1} = \theta_j + \sum_{i=1}^{n} w_{ji} o_i^t(\varphi_j^t) \tag{1}$$

[1] Mannheim University of Applied Sciences, Paul-Wittsack-Str. 10, 68163 Mannheim, Germany, email: j.fischer@hs-mannheim.de

Here $\varphi_j$ denotes the activity of neuron $j$, $w_{ji} \in \mathbb{R}$ is the weight from neuron $i$ to neuron $j$, $\theta_j$ is the bias term and $o_i^t \in \mathbb{R}$ is the output activity of the neuron $i$ at time $t \in \mathbb{N}$. The output as a function of $\varphi_j^t$ is given by the transfer function, which could be the standard sigmoid:

$$o_j^t(\varphi_j^t) = \sigma(\varphi_j^t) = \frac{1}{1 + e^{-\varphi_j^t}} \tag{2}$$

As long as we use transfer functions, which are invertible, it is possible for each time step $t$ to calculate the $\varphi_j^t$ from the corresponding $o_j^t$. If the transfer function is the standard sigmoid (see equation 2) the inverse is calculated as

$$\varphi_j^t(o_j^t) = -ln(\frac{1}{o_j^t} - 1). \tag{3}$$

The first approach to reconstruct a given trajectory is the following: If we know the output $o_j^t$ and with this the activity $\varphi_j^t(o_j^t)$ of all neurons for $t = 1..(n+2)$ successive time steps, with equation 1 for each of the n neurons a system of $n+1$ linear equations results to be solved with $n$ weights and 1 bias as variables:

$$-ln(\frac{1}{o_j^{t+1}} - 1) = \theta_j + \sum_{i=1}^{n} w_{ji} o_i^t \tag{4}$$

If this system of linear equations is solvable and if we start with the network output initialized with $o_i^{t=0}$, then the resulting network will follow the given trajectory $o_i^t$ for $n+1$ time steps. How the neural network behaves after these steps is previously unknown.

## 3 A simple example

As an example the following table shows the output values of two neurons for four time steps.

| $t$ | $o_1$ | $o_2$ |
|---|---|---|
| 1 | 0.2500001 | 0.604037 |
| 2 | 0.2549832 | 0.647125 |
| 3 | 0.2697353 | 0.684348 |
| 4 | 0.2936664 | 0.714222 |

From this table, with the help of equation 3, we can derive the activity of all neurons in each time step:

| $t$ | $\varphi_1$ | $\varphi_2$ |
|---|---|---|
| 1 | -1.098612 | 0.422315 |
| 2 | -1.072210 | 0.606426 |
| 3 | -0.995967 | 0.773826 |
| 4 | -0.877645 | 0.915979 |

Now using equation 1 we get three linear equations for each neuron:

$$-1.072210 = \theta_1 + 0.2500001w_{11} + 0.604037w_{12} \quad (5)$$

$$-0.995967 = \theta_1 + 0.2549832w_{11} + 0.647125w_{12} \quad (6)$$

$$-0.877645 = \theta_1 + 0.2697353w_{11} + 0.684348w_{12} \quad (7)$$

and

$$0.606426 = \theta_2 + 0.2500001w_{21} + 0.604037w_{22} \quad (8)$$

$$0.773826 = \theta_2 + 0.2549832w_{21} + 0.647125w_{22} \quad (9)$$

$$0.915979 = \theta_2 + 0.2697353w_{21} + 0.684348w_{22}. \quad (10)$$

Solving these equations we receive the corresponding weights and biases:

| $weight$ | $value$ |
|---|---|
| $\theta_1$ | -3.045571 |
| $w_{11}$ | 5.021175 |
| $w_{12}$ | 1.188780 |
| $\theta_2$ | -1.697853 |
| $w_{21}$ | -0.235685 |
| $w_{22}$ | 3.912343 |

## 4 The Least Mean Squares approach

Least Mean Squares (*LMS*) is a quite successful method for approximating an overdetermined set of data. The simplest case is the linear regression, where a straight line, a plane or a hyper plane is fit to a set of predetermined data points. To find the best gradient and intercept the *LMS*-algorithm minimizes the sum of square errors for $k$ coordinates, which is the sum of squared distances between the data points and the straight line, plane or hyper plane. This works quite well even for an $n$-dimensional space, assuming that the number of data points $k$ is higher than the number of dimensions $n$.

Coming back to neural computation the following question arises: If we have a desired output $\mathbf{O}_j^t$ for more than $n+2$ time steps, is it possible to calculate a weight matrix which approximates this trajectory using the *LMS*-algorithm? The Answer is: Yes it is! To calculate the weights of a network from these desired output values, we would have to minimize the square errors $\sum_t (o_j^t - \mathbf{O}_j^t)^2$ between the real output of an output neuron $j$ and the target value $\mathbf{O_j^t}$ for all times $t$. But replacing $o_j^t$ by $\sigma(\theta_j + \sum_i \mathbf{O}_i^{t-1} w_{ji})$ would result in a system of nonlinear equations. At this step another view is introduced, which simplifies the problem essentially: Instead of minimizing the Mean Square Error *MSE* of the output of the neurons, we minimize the *MSE* of the neurons activity. The solutions of both methods are identical, if the trajectory to be learned "exists" exactly. We replace $\theta$ by $w_{j0}$ with $i \in [0..n]$ and define $\mathbf{O}_0^t = 1$ for all times $t$ to simplify the following equations. The square error of the activity $\varphi_j^{t+1}$ of neuron $j$ is

$$(\varphi_j^{t+1} - \varphi_{target}^{t+1})^2 \quad (11)$$

$$((\sum_i o_i^t w_{ji}) + ln(\frac{1}{\mathbf{O}_j^{t+1}} - 1))^2. \quad (12)$$

We replace $o_i^t$ by $\mathbf{O}_i^t$, pretending that the output and the target values are nearly identical. To find a minimum of this square error of the activity of neuron $j$ the derivative of the sum over all time steps of the square error functions is calculated and set equal to zero as follows:

$$\frac{\partial \sum_t (\sum_i \mathbf{O}_i^t w_{ji} + ln(\frac{1}{\mathbf{O}_j^{t+1}} - 1))^2}{\partial w_{jk}} = 0 \quad (13)$$

which results in

$$\sum_t 2\mathbf{O}_k^t (\sum_i \mathbf{O}_i^t w_{ji} + ln(\frac{1}{\mathbf{O}_j^{t+1}} - 1)) = 0 \quad (14)$$

Where $j$ is the neuron of which the incoming weights are to be optimized and $k = [0..n]$ and $i = [0..n]$ are indices running over all biases and neurons in the network. For each neuron $j$ with $k = [0..n]$ a system of $n+1$ linear equations results which may be solved e.g. with the Gauss elimination algorithm and an expense of $O(n^3)$. It is not necessary to calculate the second derivative to decide whether the weights define a sattle point a minimum or a maximum of the error function. The sum of the error function has no sattle point, because it is of parabolic shape. It has no maxima, because it has a positive sign. So the result must be an absolute minimum.

## 5 A quasi periodic attractor

The first experiment is given by calculating a two neurons weight-matrix for an output given by the functions

$$f_1(t) = 0.5sin(t/10.0)^2 + 0.25 \quad (15)$$

and

$$f_1(t) = 0.5sin(t/10.0 + 1.0)^2 + 0.25. \quad (16)$$

So one could ask: "How could we know that such functions are reproducable by the network?" The answer is simple: "We do not know if the trajectory is reproducable until we try it out." So we calculate a weight matrix using the functions $f_1$ and $f_2$ as target (see fig. 1), minimizing the *MSE* within 100 time steps.

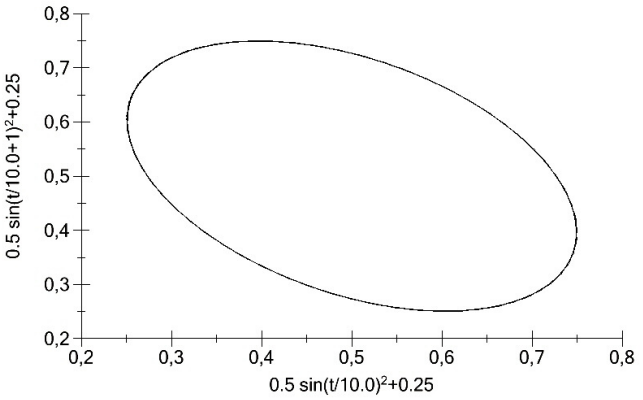



**Figure 1.** This is the target attractor.

The mean square error of these trained trajectories for the output is $MSE = 0.003548$. The trajectory is stable (see Fig. 2) and looks quite similar to the original shown in Fig. 1.

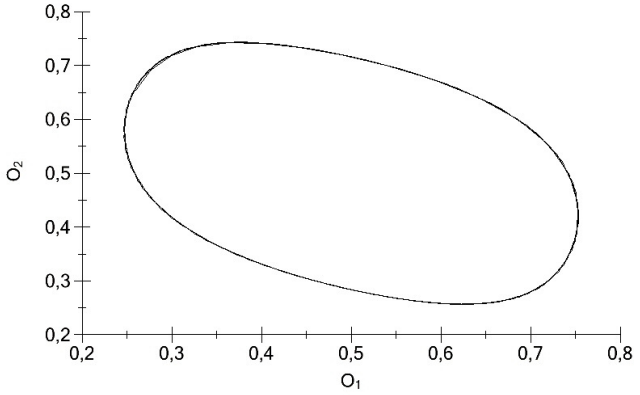



**Figure 2.** This is the output for the network calculated from the target trajectories $f_1(t)$ and $f_2(t)$ (equation 15 and 16). The following weights result: $w_{11} = 4.592138$, $w_{21} = -0.934564$, $w_{12} = 0.938969$, $w_{22} = 3.813414$, $\theta 1 = 2.765417$, $\theta 2 = 1.439541$.

## 6 A chaotic neural attractor

To be able to reconstruct a neural network with chaotic attractor often results in several problems. Using gradient descent leads to oscillations of the weight values and to slow convergence. Such difficulties result from bifurcations in the activation dynamics of the neural output.

With the described *LMS*-method these problems do not exist. The following example shows, that it is possible to reconstruct a network which produces a chaotic attractor. Fig. 3 shows the logistic map and the corresponding Lyapunov exponent $\lambda$ of a two neuron network with five fixed and one variable weight. The Lyapunov exponent $\lambda$ is a measurement for the dynamical behavior of a system. $\lambda > 0$ means that the attractor is chaotic. We choose one parameter set with $\lambda > 0$ e.g. $\theta_1 = -4$, $\theta_2 = 4$, $w_{12} = 10$, $w_{21} = -10$, $w_{22} = 0$ and $w_{11} = 6$ ($\lambda \approx 0.0177$).
We generate some output data as follows:

| $t$ | $o_1$ | $o_2$ |
|---|---|---|
| 1 | 0.770719 | 0.00978897 |
| 2 | 0.673092 | 0.0239584 |
| 3 | 0.569089 | 0.0611735 |
| 4 | 0.506566 | 0.155659 |
| 5 | 0.644739 | 0.256229 |
| 6 | 0.91914 | 0.0796299 |
| 7 | 0.909795 | 0.00553343 |

With equation 14 a system of 6 linear equations results, where the 6 variables correspond to the self connections of both neurons, the interconnections between the neurons and the threshold of each neuron. To reconstruct the weights and thresholds we just have to solve the system of linear equations. A short C-program with low precision and bad rounding leads to the following values:

| $weight$ | $value$ |
|---|---|
| $\theta_1$ | 3.9999 |
| $w_{11}$ | 5.9999 |
| $w_{12}$ | 9.9999 |
| $\theta_2$ | -3.9999 |
| $w_{21}$ | -9.9999 |
| $w_{22}$ | 0.0001 |

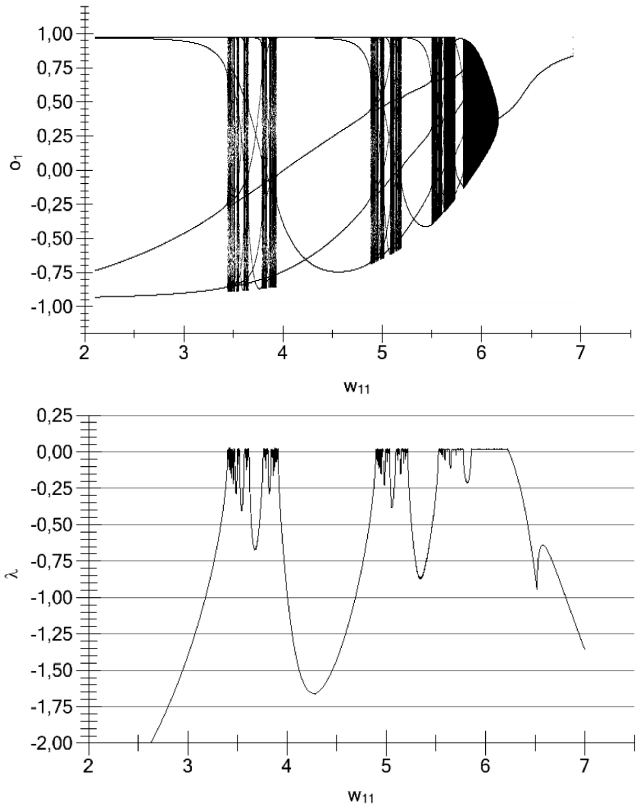



**Figure 3.** Top: the logistic map of a two neuron ensemble with the weights $\theta_0 = -4$, $\theta_1 = 4$, $w_{12} = 10$, $w_{21} = -10$, $w_{22} = 0$, while $w_{11}$ is tuned from 0 to 7. Bottom: the Lyapunov exponent shows where the network produces a chaotic behavior ($\lambda > 0$) and where we could pretend a fixed point attractor ($\lambda < 0$).

## 7 A real world application: Network Shrinking

To know the size of a network architecture from what it should be able to learn is not a trivial estimate. Starting with a large network with many neurons could lead to convincing results [3], but is it possible to reduce the number of neurons without destroying the dynamical properties of the rest of the network? What if a quasi periodic attractor becomes a fixed point attractor? What if a periodic attractor turns into a chaotic one? The question is: How could we cut out one neuron with a minimal change of the trajectories of the remaining neurons? The usual strategy is to cut out those neurons, which have the smallest connection weights and therefore have the smallest influence on the rest of the network. In general this is done without adapting the weights of the remaining neurons.

This strategy is not really satisfying because in most cases nearly all neurons of a trained network are massively connected to others. There are usually no ”light weighted” candidates which could be neglected.
To make things more clear we present a toy example as follows: Given a network with two output and one hidden neuron as shown in fig. 4: is it possible to cut out the hidden neuron without destroying the quasi periodic attractor? As we can see in fig. 5 it is. For this toy example one might find a solution without the Least Mean Squares method: Having a closer look at the figure one will find that

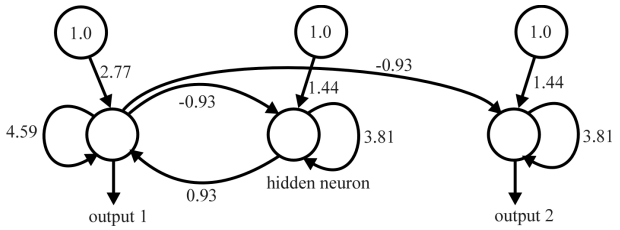



**Figure 4.** A toy example: two output neurons (left and right) and one hidden neuron (center) generating a quasi periodic attractor. How can we cut out the hidden neuron without changing the attractor?

the output 2 neuron has the same inputs as the hidden neuron. This means that it also will generate the same output as the hidden neuron. Cutting the hidden neuron out without adapting the weights is a bad idea. The attractor would get stuck in a fixed point. Instead we have to transfer the connection from the hidden neuron to the output 1 neuron as a connection from the output 2 neuron to the output 1 neuron as shown in fig. 5. Then we can delete the hidden neuron. The *LMS*-Method directly leads to the desired network, even if the network is much more complex. The algorithm is proven to work with thousands of neurons and connections.

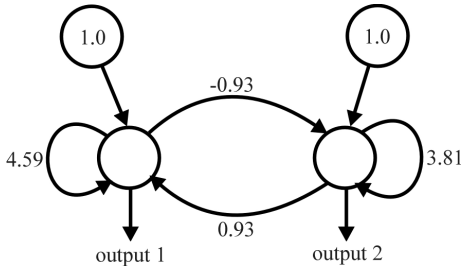



**Figure 5.** Toy example: The reduced network producing the same attractor. The output connection from the hidden neuron is transferred to the output 2 neuron.

## 8 Discussion

The procedure presented is efficient for all kinds of trajectories. It is possible to reproduce fixed point, periodic, quasi periodic and chaotic behavior. The only condition for successful reproduction is that the given attractor fits into the network. To get an idea about how much data fits into an $n$-neuron network we may think of a simple linear regression task, where a straight line should be fit to a set of data points. In this case we have two variables: the gradient and the intercept. Using two data points it can be guaranteed, that the line will exactly contain both points. If we have three data points in three dimensions we may guarantee that the approximated plane contains these three data points. This means that if we have an $n$-dimensional neural network, we could approximate $n$ data points exactly. All further data points lead to a deterioration of the quality of the reproduced trajectory. On the other hand attractors of periods $p < n + 1$ or fixed point attractors may lead to a system of equations which is over determined. There are more variables (weights) than equations. In this case we might predefine $(n + 1) - p$ weights e.g. with $w_{ij} = 0$ and compute the rest of the weight matrix in order to get a network which follows the given attractor.

## 9 Conclusion

The findings presented in this paper show up the duality of the weight- and the output space of a neural network. Once the output trajectory is known for a minimum of $n + 2$ time steps for a network with $n$ neurons, the weights of the network may be reconstructed directly in one step. The described *LMS*-method is efficient and easy to apply to any given period of output trajectories as there are fixed point attractors, quasi periodic, periodic or chaotic neural network trajectories.

In this work the Method is approved to networks with attractors of different dynamical properties. To show that the finding is not only a useless theoretical construct we demonstrate an efficient shrinking method for recurrent neural networks.

We might use the *LMS*-Method for network recovery as an analyzing tool. We could induce amplitude changes or phase shifts between different outputs, we even could try to reconstruct small networks with slowly changing weights at different time intervals (assuming that the weights are nearly constant during that interval) and to see how the weights are changing from one point in time to another. May be even the learning rule, assuming that it is an unsupervised one, might be extracted.

If the method could be adapted to simple models of biological neurons is one of the multitude of open questions. Our hope is to find answers to some of them.

## REFERENCES

[1] T. J. Sejnowski D. H. Ackley, G. E. Hinton, 'A learning algorithm for bolzmann machines', *Cognitive Science*, **9**, 147–169, (1985).
[2] R. Salakhutdinov G.E. Hinton, 'Reducing the dimensionality of data with neural networks', *Science*, **313(5786)**, 504–507, (2006).
[3] H. Haas H. Jäger, 'Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication', *Science*, **304 no. 5667**, 78–80, (2004). recurrent neural networks.
[4] J. Hopfield, 'Neural networks and physical systems with emergent collective computational abilities', in *PNAS USA, 79*, pp. 2554–2558, (1982).
[5] H. Jäger, 'The echo state approach to analysing and training recurrent neural networks', Technical report, German National Research Institute for Computer Science, (2001).
[6] F. Pasemann, 'Structure and dynamics of recurrent neuromodules', *Theory in Biosciences*, **117**, 1–17, (1998).
[7] F.J. Pineda, 'Generelization of back-propagation to recurrent neural networks', *Physical Review Letters*, **59**, 2229–2232, (1987).
[8] D. Zisper R.J. Williams, 'A learning algorithm for continually running fully recurrent neural networks', *Neural Computation*, **1**, 270–280, (1989).
[9] X. Wang, 'Period-doublings to chaos in a simple network: An analytical proof', *Complex Systems*, **5**, 425–441, (1991).
[10] X. Wang, 'Dynamics and bifurcation of neural networks', in *Handbook of Neural Networks*, (1995).
[11] T.G. Kincaid Y. Fang, 'Stability analysis of dynamical neural networks', in *IEEE Transactions of Neural Networks, Volume 7*, pp. 996–106, (1996).

4th International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2011)
Berlin, Germany, December 9 2011

# Hierarchical Exhaustive Construction of Autonomously Learning Networks

**Goren Gordon** [1]

**Abstract.** Autonomous learning is the ability to learn without external teachers. What *can* an agent learn autonomously? To answer this question we propose a hierarchical exhaustive combinatorial constructive algorithm. It generates subnetworks that attempt to learn all possible correlations between subsets of available raw data from the agent's sensors and motors. Using the concept of pruning, subnetworks that are presented with uncorrelated data sets are removed, resulting in a small pool of *viable* subnetworks. These augment the raw information dataset in higher levels, in which the exhaustive construction and pruning are repeated. The end result of the hierarchical process is a pool of viable and reliable subnetworks that represent *all* the correlations the agent can autonomously learn. One can then construct full networks by wiring learned subnetworks in order to perform specific tasks. The algorithm is implemented on a robot with a moving camera and an arm, highlighting novel concepts regarding active sensing and autonomous learning. We show that the robot's autonomously learned viable and reliable subnetworks are its sensory-motor internal models, motion detection, visual self-recognition and camera to arm coordinate transformation. The robot's only non-trivial closed-loop execution network is shown to perform reaching movements towards a moving object and is robust to noise and changes in the robot's sensors and motors due to its concurrent execution and re-learning capabilities.

## 1 Introduction

One of the brain's greatest virtues is its ability to learn. However, one can distinguish between two learning categories, namely, external- or teacher-mediated learning and autonomous learning, i.e. learning from internally accessible information. While naming of objects and colors is externally taught, e.g. one must be told that the word "yellow" is associated to a specific color perception, controlling your own body movements is autonomously learned [20, 5]. However, learning reaching movements are not so easy to classify [30, 2].

In this contribution we address the question: what *can* be autonomously learned, without external teachers? We wish to model our view of the brain's solution to this question. For this reason we construct a hierarchical neural network that attempts to autonomously learn *all* correlations between available data, given an agent's sensors, motors and performed actions. By *all*, we mean an exhaustive combinatorial construction of subnetworks, representing all possible subsets of available data, wherein each subnetwork *attempts* to learn a specific data subset's correlation. Many such subsets hold no correlations and are thus unlearnable; by employing the concept of pruning [26, 8], the associated subnetworks become *non-viable*, i.e. networks with no contributing neurons. The phenomenon in which an over-sized neuronal network is first initialized, followed by elimination of non-active elements is prevalent in the brain and is called exuberance [13].

Hierarchy is achieved by augmenting a higher level's available data by lower levels' viable networks. Thus, a new level exhaustively constructs subnetworks that attempt to learn correlations between outputs of lower levels' networks and raw sensory-motor data. Exuberance and pruning follows in order to distill the viable and reliable subnetworks of this level in the hierarchy. The process continues for higher hierarchical levels.

The end result of the hierarchical construction is a pool of subnetworks that represents *all* the correlations the agent can autonomously learn. This pool can then be wired in such a way so as to perform specific tasks. Since the agent cannot learn any other correlation, the combination of all possible wiring of the subnetwork pool represents the entire repertoire of tasks the agent can perform. Furthermore, since all the elements are autonomously learned, concurrent execution and learning can be performed, overcoming calibration and deterioration errors on-line.

We demonstrate the process on a real robot, with a 1 degree-of-freedom arm and a camera mounted on a single motor, representing the eye. We show that the viable subnetworks of the first level of the hierarchy represent only the internal models (IM) [14, 21, 29] of the sensory-motor coupling, among which visual motion-detection is a notable example. The second level uses the first level's subnetworks to learn more complex correlations, such as visual self-recognition [20, 5, 17]. The third and final level encompasses the entire visual field and autonomously learns visual-arm coordinate transformation [22]. The viable and reliable subnetworks are then wired to achieve the *only* functional closed-loop circuit, given the learning schedule, i.e. the only circuit that performs non-trivial action. The circuit performs a reaching task, with concurrent autonomous learning of the composing elements.

The novel features of this paper are: (i) a comprehensive brain-inspired framework of hierarchical autonomous learning of sensory-motor correlations; (ii) connection between autonomous and active sensing paradigms; (iii) a single learning algorithm that generates motion detection, self-recognition and hand-eye coordinate transformation; (iii) demonstration of a fully autonomous learning reaching robot.

The paper is organized as follows. We begin with a brief description of the model architecture and framework in Sec. 2. We then present in Sec. 3 the mathematical notations of the agent, data sets and subnetworks, followed by a description of the learning and pruning processes. Section 4 details the hierarchical exhaustive combina-

[1] Department of Neurobiology, Weizmann Institute of Science, Israel, email: goren@gorengordon.com

torial construction of all the subnetworks, followed by an analysis of the growing complexity of the network and possible execution tasks. Both sections are accompanied by a running example (Fig. 4), whose details are given in Sec. 5. Related works are decsribed in Sec.6 and the discussion in Sec. 7 concludes the paper.

## 2 Model Architecture and Framework

The main concept behind the proposed architecture, Fig. 1, is autonomous learning of sensory-motor correlations. The implemented algorithm is *internally supervised* learning, i.e. supervised learning where a "labeled" training set is provided by the agent itself. This is not a form of unsupervised learning [34, 25], but rather learning to predict correlations between subsets of available information. This information is the time-series of raw data from the sensors and motors of the agent, Fig. 1(a).

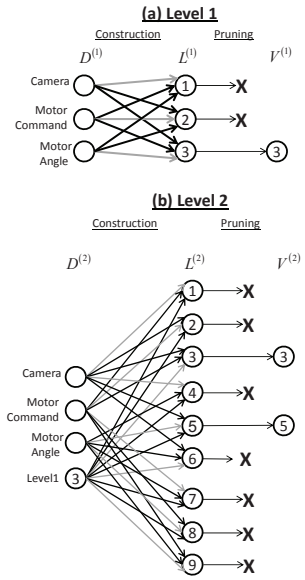



**Figure 1.** Model architecture, where each subnetwork (numbered circle) autonomously learns the correlation between two inputs (black arrows) and one output (gray arrows). (a) Level 1 construction of *all* subnetworks $L^{(1)}$, given camera sensor and a motor information $D^{(1)}$; followed by pruning that leaves only one subnetwork viable, $V^{(1)}$. (b) Level 2 construction of *all* subnetworks $L^{(2)}$ that include level 1 viable network; pruning leaves only subnetworks 3 and 5 viable, $V^{(2)}$.

The architecture construction is exhaustive in the sense that correlations between all subsets of available information are learned. We are interested in a brain-like architecture and thus employ a prevalent phenomenon in the brain called exuberance [13], which describes a rapid growth of connectivity between many neurons on many levels, followed by deterioration of unused or non-correlated connections. In our implementation, each subnetwork is an artificial neural network, initialized with many hidden neurons, followed by pruning [26, 8] of neurons whose decaying weights are smaller than a given threshold. If no correlations are present, the pruning process will result in a *non-viable* subnetwork, i.e. a network with no contributing neurons, thus exemplifying the network's inability to learn the subset's correlation. In some situations, usually for large-input subnetworks, pruning still results in a viable subnetwork, but it is *unreliable*, i.e. its prediction error even on the training set is high, thus exhibiting another form of inability to learn.

Following the brain's hierarchical structure, our proposed architecture is hierarchical in the sense that higher level subnetworks learn correlations between lower levels' subnetwork outputs and the raw sensory-motor information. This is reminiscent of cascade correlation networks [7, 16, 31] in which each new hidden layer neuron is connected to the input layer and lower-level hidden neurons. However, in our construction, each correlation learned is a whole (learned and viable) subnetwork that augments the input-space and allows learning of new correlations.

More specifically, in the algorithm's first level of the hierarchy, only *raw unprocessed* data from the sensors and motors are used in the aforementioned process, which ends with a small number of viable subnetworks, Fig. 1(a). In the next level of the hierarchy, the previous level's learned and viable subnetworks are combined with the sensory-motor data, Fig. 1(b). Another exhaustive combinatorial construction of new subnetworks is performed, where now each subset must include *at least* one learned subnetwork from the previous hierarchical level. Exuberance and pruning follows in order to distill the viable and reliable subnetworks. The process continues for higher hierarchical levels.

While hierarchical construction of unsupervised learning networks have been used on pure sensory data, e.g. images [12, 25], our construction focuses on internally supervised learning of correlations between sensory flow and motor actions. Hence, *active sensing* [4, 28], in which sensors are moved and controlled by the agent, is paramount to the understanding of the learned correlations. These represent what the agent can learn and predict about its own body and how it senses the environment in an active fashion. Thus, the agent can learn to predict an actuator's influence on its mounted sensor's information flow, as well as learn to determine the appropriate motor command that will generate a specific sensory input. These are the active sensing counterparts of the forward and inverse models, respectively [14, 21, 29].

## 3 Agent and Subnetwork Notations

This section introduces the basic elements of the proposed model, namely, the agent, its sensory-motor data and the subnetworks. The subnetworks' learning and pruning processes are then described. The section is accompanied by a running example of the implementation of the process on a real robot (Fig. 2), to better elucidate the finer details of the model.

### 3.1 Agent and Sensory-Motor Data

The agent is composed of $N_M$ motors and $N_S$ sensors. The former executes motor commands $m_t^i$, $i = 1, \ldots, N_M$ and the latter receives sensory input $s_t^j$, $j = 1, \ldots, N_S$. The dataset time-series is composed of copies of the motor commands, known as efference copies [18, 3], and sensory input, with possible delays: $D_t^{(1)} = \{m^i_{\tau^i_m}, s^j_{\tau^j_s} | \tau^i_m = t, t-1, \ldots, t-d^i_m; \tau^j_s = t, t-1, \ldots, t-d^j_s\}$, where $d^i_m$ is the maximal delay of the $i$th motor command and $d^j_s$ is the maximal delay of the $j$th sensory input. The $(1)$ superscript denotes the first level of the hierarchy, which has $||D_t^{(1)}||$ data elements.

*Example*. The example agent is a LEGO Mindstorm robot (Fig. 2(a)) with $N_M = 2$ motors; the first controls an arm and the second the camera. It has $N_S = 3$ sensors, where the first two are

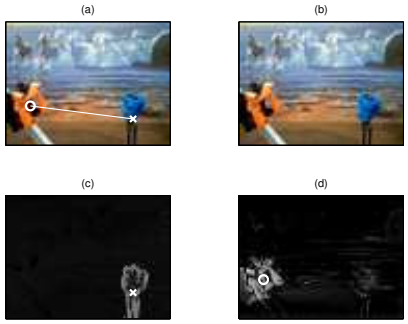



**Figure 2.** (a,b) Two consecutive images as captured by camera, $c_t, c_{t-1}$. O marks the center of the arm, X marks the center of the moving object. The line shows the distance between the two. (c) Output of motion detection subnetwork, $V_7^{(1)}$. (d) Output of visual self-recognition subnetwork, $V_1^{(2)}$ (see text for details).

proprioception sensors of the motors, $p_t^{1,2} := s_t^{1,2}$, i.e. report the motor angle, and the third is the camera, $c_t := s_t^3$. The camera is composed of $80 \times 60$ pixels, and we shall consider an increasing receptive field with increasing hierarchical levels, ranging from a single pixel in the first level, through patches of 7x7 pixels in the second level, up to the whole image in the third level of the hierarchy (see below). We consider no motor delays, $d_m^{1,2} = 0$ and a single time delay for the sensors, $d_s^{1,2,3} = 1$. Thus the data set at each time step is $D_t^{(1)} = \{m_t^1, m_t^2, p_t^1, p_{t-1}^1, p_t^2, p_{t-1}^2, c_t, c_{t-1}\}$, with $||D_t^{(1)}|| = 8$.

## 3.2 Subnetworks, Learning and Pruning

**Subnetworks**. A subnetwork is a function approximator that attempts to learn the correlation between elements of the data set. It is denoted by $L_{ijk} = p(D_i|D_j, D_k), \forall i, j, k \in D, i \neq j \neq k \neq i$, where the subscript $i, j, k$ are indexes of the data-set elements (different from the subscript $t$, which indicates the current time step of the *whole* data set). We have restricted the subnetworks to a $2 \mapsto 1$ networks, i.e. only mapping of two dataset elements to another, where extensions to more elaborate mappings is straightforward. Hence, there are $||L|| = ||D|| \times (||D|| - 1) \times (||D|| - 2)/2$ possible subnetworks. The implementation of the subnetworks is via an artificial neural network (ANN), with input neurons that receive the data elements $D_j, D_k$, several hidden layers and output neurons that encode $D_i$.

Each subnetwork was taken to be multi-layered in order to allow learning of complex sensory-motor correlations. A-priori the correlation is not known, and hence initially a complex network is required for all subnetworks to allow generality. Furthermore, in this implementation there is a unique approximated output for every input element, i.e. the function approximation is deterministic, eliminating the need for probability normalization. It is also imperative, for proper comparison, that the ANN structure and parameters be the same for all subnetworks. Thus, all dataset elements were normalized to be $D \in [-1, 1]$ and saturated-linear transfer functions were used in the output neurons.

**Learning**. Motion of the agent's motors produces the dataset time-series, which is treated as the subnetworks' training set. Autonomous internally supervised on-line learning proceeds with the presentation of this dataset time-series to *all* the subnetworks, in parallel. We have implemented a back-propagation learning algorithm, with learning rate $\beta$ and momentum $\alpha$.

**Pruning**. Concurrent with the learning algorithm, we have implemented a pruning algorithm [26, 8] via weight decay. An additional penalty term was introduced to the ANN weights' update rule in the form of $-\gamma \text{sign}(w_{ij})$, where $0 < \gamma < 1$ is the pruning rate and $w_{ij}$ is the respective weight. Hence, the full update rule is given by $\Delta w_{ij} = \beta \epsilon f'(x) + \alpha w_{ij} - \gamma \text{sign}(w_{ij})$, where $\epsilon$ is the (backpropagated-) error and $f(x)$ is the neuron's transfer function. For each level we have chosen $\gamma$ to be proportional to the overall learning time of all networks, i.e. while learning and pruning were concurrent, as expressed in the update rule, effectively pruning manifested after learning (were possible) was stabilized.

At each time step, the sum of the absolute weights for each neuron in the hidden layers (both input and output weights) was compared to a given threshold, $h_{\text{threshold}}$; if the sum was below the threshold, that neuron was pruned. If the last neuron of a hidden layer was pruned, the subnetwork was deemed non-viable. Furthermore, for subnetworks with only two input neurons (see below), the sums of the absolute weights of the input neurons were calculated. If one sum was more than $i_{\text{mul}}$ times greater than the other, the subnetwork was termed non-viable, since it depends only on a single input, and not both. We denote the viable subnetworks by $V \subseteq L$.

*Example*. Since the robot's raw dataset has $||D_t^{(1)}|| = 8$ elements, there are a total of $||L^{(1)}|| = 168$ possible subnetworks. One example of a viable subnetwork is the first motor's forward model [29] $L = p(p_t^1|p_{t-1}^1, m_t^1)$, i.e. the prediction of the next motor angle, given the previous angle and the given motor command. $L = p(p_t^1|p_{t-1}^2, c_t)$, on the other hand, is non-viable since there is no correlation between the first motor's angle, the previous other motor's angle and the current image pixels.

# 4 Hierarchical Construction

Using the general notations described above, we proceed to the exhaustive combinatorial construction of the subnetworks in an hierarchical fashion. We first describe the augmentation of the dataset by previous level's viable networks and then produce the current level's subnetwork pool. A complexity analysis follows, showing the double-exponential increase in network complexity, had exuberance and pruning were not implemented. The exact increase in complexity cannot be a-priori computed since it depends on the agent and its environment, but a drastic decrease in complexity results if one assumes proportional pruning. The viable and reliable subnetworks allow the wiring of specific circuits that can perform specific tasks. This is discussed at the end of the section.

## 4.1 Dataset Augmentation and Subnetwork Pool

The first hierarchical level dataset is composed strictly of raw sensory-motor data, $D^{(1)}$. Using these time-series as a training set, a pool of subnetworks is composed $L_{ijk}^{(1)} = p(D_i^{(1)}|D_j^{(1)}, D_k^{(1)})$, $\forall i, j, k \in D^{(1)}, i \neq j \neq k \neq i$. However, not all subnetworks are viable and following the process of concurrent learning and pruning, a subset of viable subnetworks is produced $V^{(1)} \subseteq L^{(1)}$.

The second hierarchical level now has access to the first level's viable subnetworks processed information, i.e. presented with the raw sensory-motor dataset, the viable networks produce predictions based on their learned correlations. Hence, the augmented second level dataset is $D^{(2)} = D^{(1)} \cup V^{(1)}$. One can then proceed to exhaustively construct subnetworks that will attempt to learn all possible $2 \mapsto 1$ correlations of the augmented dataset. However, only sub-

networks that include *at least* one of the previous level's viable networks will be constructed. The rest were already learned in the previous level. The second level's subnetwork pool is denoted by $L^{(2)}_{ijk} = p(D^{(2)}_i|D^{(2)}_j, D^{(2)}_k)$, $\forall i,j,k \in D^{(2)}, i \neq j \neq k \neq i$ such that $\exists D^{(2)}_{i,j,k} \in V^{(1)}$. The total number of such subnetworks is then given by $||L^{(2)}|| = ||D^{(2)}|| \times (||D^{(2)}|| - 1) \times (||D^{(2)}|| - 2)/2 - ||L^{(1)}||$. Concurrent learning and pruning then follows to produce $V^{(2)}$ viable networks.

This can be easily generalized to higher levels, as follows:

$$D^{(n)} = D^{(n-1)} \cup V^{(n-1)} \quad (1)$$

$$L^{(n)}_{ijk} = p(D^{(n)}_i|D^{(n)}_j, D^{(n)}_k), \quad \forall i,j,k \in D^{(n)}, i \neq j \neq k \neq i \quad \text{s.t.} \exists D^{(n)}_{i,j,k} \in V^{(n-1)} \quad (2)$$

$$||L^{(n)}|| = ||D^{(n)}|| \times (||D^{(n)}|| - 1) \times (||D^{(n)}|| - 2)/2 - \sum_{m=1}^{n-1} ||L^{(m)}|| \quad (3)$$

*Example*. The robot's first level of the hierarchy produces $||L^{(1)}|| = 168$ subnetworks. However, only internal models (IM) of the sensory-motor dataset have correlations (see below, Fig. 3). These are presented to three subnetworks that relate $p^1_t, p^1_{t-1}, m^1_t$ (IM of the arm); three subnetworks that relate $p^2_t, p^2_{t-1}, m^2_t$ (IM of the eye-motor) and; three subnetworks that relate $c_t, c_{t-1}, m^2_t$ (IM of the eye-motor and camera). Hence, $||V^{(1)}|| = 9$ and $||D^{(2)}|| = 17$, resulting in $||L^{(2)}|| = 1872$.

## 4.2 Complexity Analysis

One can consider the increase in the number of subnetworks as the hierarchical levels grow. Assume that pruning is not implemented; this results in changing Eq. (1) to $D^{(n)} = D^{(n-1)} \cup L^{(n-1)}$. Thus, denoting $x_n := ||D^{(n)}||$ and $y_n := ||L^{(n)}||$, we get the following recursion relations:

$$x_n = x_{n-1} + y_{n-1} \quad (4)$$

$$y_n = x_n(x_n - 1)(x_n - 2)/2 - y_{n-1} \quad y_0 = 0 \quad (5)$$

This results in double exponential dependency on the hierarchy level: $y_n \sim O(x_1^{3^n})$.

However, one can counter this increase by the use of exuberance and pruning. Consider that only a very small subset of the subnetwork pool remains viable, such that $||V^{(n)}|| = \kappa||D^{(n)}||$, $\kappa > 1$. This results in a drastic decrease in subnetwork pool complexity, to a single exponential dependence: $y_n \sim O(\kappa^{3(n-1)})$.

Furthermore, as seen in the example below, there is a possibility that several of the viable networks are *equivalent*, i.e. they convey the same information and are thus redundant. This may result in a linear increase in the number of informative viable networks as hierarchical level increases. This is indeed the case in the robot example analyzed below.

## 4.3 Execution of a Task

Once the hierarchical levels' viable subnetworks were learned, one can construct a full network to perform specific tasks. This can be done by connecting one subnetwork's output to another's input. Notice that during execution the inputs to the subnetworks may differ from those during the learning phase. However, they must accommodate the inputs' dimensionality and content.

In order for a task to be operational, the last subnetwork must have a motor output. This drastically restricts the number of subnetworks that may reside in the end of the task network. However, the number of possible wirings cannot be a-priori computed, since the number and characteristics of the *viable* networks are not known.

*Example*. One of the first level's viable subnetwork is the arm's inverse model, $V^{(1)} = p(m^1_t|p^1_t, p^1_{t-1})$. During autonomous learning, it was presented with the known time-series, where $p^1_{t-1}$ was the delayed proprioception input. However, during execution it can serve as a mechanism to achieve a specific angle position, $p^1_{\text{goal}}$: $V^{(1)} = p(m^1_{t+1}|p^1_{\text{goal}}, p^1_t)$, i.e. it determines the next motor command $m^1_{t+1}$ via the goal position and the current position. Hence, it may reside at the end of a functional network.

# 5 Robot Implementation

We have implemented the proposed model on a LEGO Mindstorm robot (Fig. 1(a), Supp. Movie), with a 1 degree-of-freedom (DOF) arm, $m^1_t$, and a single motor $m^2_t$ that controls the pan of a USB camera, $c_t$. As described above, the raw data set at each time step is $D^{(1)}_t = \{m^1_t, m^2_t, p^1_t, p^1_{t-1}, p^2_t, p^2_{t-1}, c_t, c_{t-1}\}$, where $p^{1,2}_t$ is the proprioception information relating the motor angle. First, the hierarchical construction of the viable subnetwork pool is presented. It is followed by a presentation of a possible wiring of the learned subnetworks to accomplish a reaching task.

## 5.1 Construction of Viable Subnetwork Pools

### 5.1.1 First level

The first level data set, $D^{(1)}$, was used to construct an exhaustive set of all $||L^{(1)}|| = 168$ possible subnetworks. All the subnetworks had two hidden layers with four neurons each, and $\beta = \alpha = 0.1$, $\gamma = 0.0001$. The robot moved its arm motor randomly, and its camera motor in a succession of random motion and then no motion. This was repeated for 100,000 time steps, in which concurrent learning and pruning was performed on all subnetworks in parallel, Fig. 3.

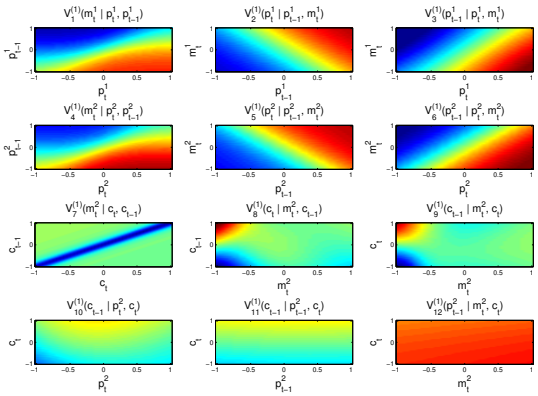


**Figure 3.** Mapping of viable subnetworks of the first level. In order to visualize the image-space, here $c_t$ is taken to be the normazlized gray-scale value of the RGB pixel in $V^{(1)}_{7-12}$.

The first level of the hierarchy represents the lowest level in the visual pathway, e.g. retina. Hence, $c_t$ in this level was set to be a single RGB pixel. During the learning process, a subnetwork that had either

$c_t$ or $c_{t-1}$ was presented with all the image pixels in a randomized permutation order, thus having a training set 4800-times larger than other networks. Furthermore, it means that the input/output layers of such networks had more neurons: networks that had one or two camera inputs had four or six input neurons, respectively; networks that had a camera output had three output neurons (RGB). We thus normalized the learning and pruning rate, $\beta, \gamma$, by scaling them by $1/4800$. We further factorized pruning by scaling the pruning rate, $\gamma$ by one over the number of input neurons.

Moreover we introduced a minor modification to expedite learning: Motor commands were continuous in their regime, $m_t^{1,2} \in [-1, 1]$. However, learning visuo-motor correlation is more related to the existence of motion, rather than its direction. Learning to distinguish between $|m_t^{1,2}|$ using $m_t^{1,2}$ is very difficult and time-consuming for a small ANN, since it is non-monotonic. Hence, for all subnetworks in the first level that contained at least one visual input and at least one motor command, the latter was taken to be its absolute value. For example, $L^{(1)} = p(m_t^2|c_t, c_{t-1}) \Rightarrow p(|m_t^2||c_t, c_{t-1})$.

**Viable subnetworks**. The viable networks in the end of this process were (Fig. 3): $V_1^{(1)} = p(m_t^1|p_t^1, p_{t-1}^1)$, $V_2^{(1)} = p(p_t^1|m_t^1, p_{t-1}^1)$ and $V_3^{(1)} = p(p_{t-1}^1|p_t^1, m_t^1)$ representing the inverse, forward and postdiction internal models of the arm motor, respectively [29]; $V_4^{(1)} = p(m_t^2|p_t^2, p_{t-1}^2)$, $V_5^{(1)} = p(p_t^2|m_t^2, p_{t-1}^2)$ and $V_6^{(1)} = p(p_{t-1}^2|p_t^2, m_t^2)$ representing the inverse, forward and postdiction IM of the camera motor, respectively, and; $V_7^{(1)} = p(m_t^2|c_t, c_{t-1})$, $V_8^{(1)} = p(c_t|m_t^2, c_{t-1})$ and $V_9^{(1)} = p(c_{t-1}|c_t, m_t^2)$ representing the inverse, forward and postdiction IM of the camera motor and camera image, respectively. Three more networks were viable, but their corresponding prediction maps hold no information: $V_{10}^{(1)} = p(c_{t-1}|p_t^2, c_t)$,$V_{11}^{(1)} = p(c_t|p_{t-1}^2, c_{t-1})$ and $V_{12}^{(1)} = p(p_{t-1}^2|m_t^2, c_t)$. We believe that further learning would have resulted in their pruning.

The first six viable networks have been described intensively in the literature [14, 21, 29]. For a 1 DOF constellation they are very simple, whereas for more DOF there are known problems, mainly in the inverse models [14, 21]. However, this is not the main topic of the paper and thus we do not elaborate on it.

Of special interest is $V_7^{(1)} = p(m_t^2|c_t, c_{t-1})$ (Fig. 2(c)); it receives the current image, the previous image and learns whether the camera motor has moved. In a non-homogeneous visual environment, when the camera moves, most of the pixels' colors change; when the camera does not move, most of the pixels' colors do not change. Hence, this subnetwork represents an *autonomously learned* visual change detector.

Figure 3 shows the learned mapping of the viable subnetworks of the first level. As can be seen, the motor's internal models are (almost) linear mapping. $V_7^{(1)}$ is the visual change detection: as described above, the output was rescaled to be 1 for motion and -1 for no motion. This shows that if two pixels are identical, the output is -1, whereas if they are different, the output is 1, constituting a true change detector. Notice that $V_{8,9}^{(1)}$ are similar and represent the visual prediction: if there is no motion (left side), the output pixel is identical to the input pixel; if there is motion (right side), nothing can be said of the output pixel. Finally, the maps of $V_{10-12}^{(1)}$ are almost flat, representing no information or correlation.

**Novel features**. The nine viable networks of the first level hint towards a more general concept of autonomous learning: active sensing circuits [4, 28], whereby motor commands influence sensory information, enables autonomous learning of internal models of the sensory-motor coupling. Thus, while one interpretation of $V_7^{(1)}$ is visual change detection, it is learned similarly to $V_{1,4}^{(1)}$ which are "proper" inverse models [14, 21]. Hence, it can be used as a visual inverse model, or active vision [1, 23, 19]: given the current image and a goal image, what is the proper motor command? Conversely, $V_{1,4}^{(1)}$ can be used as joint-angle change detectors: given the current and previous angles, was the joint moved? The exhaustive construction of all subnetworks brought these novel concepts to light: (i) There is one-to-one mapping between inverse models and change detectors and; (ii) only active sensing coupling enables autonomous learning of internal models and change detectors.

### *5.1.2 Second level*

Eqs. (1-3) result in $||D^{(2)}|| = 17$ and $||L^{(2)}|| = 1872$. This was computationally too expensive for the setup we have considered, so made the following restrictions: (i) only the arm motor was considered, $m_t^1, p_t^1$; (ii) no sensory or motor delays were considered, $d_m^{1,2} = d_s^{1,2,3} = 0$; (iii) only the arm motor internal models and the visual change detection were taken, $V_{1-3,7}^{(1)}$ and; (iv) only subnetworks that had at least one visual component were considered. This resulted in a total of $||L^{(2)}|| = 72$ subnetworks. While this is considerably less than the exhaustive pool, it is still large enough to demonstrate all the proposed concepts. All the subnetworks had two hidden layers with ten neurons each, and $\beta = 5, \alpha = 0.1, \gamma = 0.001$. The robot moved its arm motor randomly for 10,000 time steps, in which concurrent learning and pruning was performed on all subnetworks in parallel.

The second level of the hierarchy represents a higher level in the visual pathway, one in which features are detected [15]. Hence, $c_t$ in this level was set to be a 7x7 RGB pixel array. As with the previous level, each time step all 7x7 arrays composing the image were presented to the subnetworks in a randomized permutation order. Similarly, the number of the input/output layers' neurons were enlarged and the learning and pruning rates were rescaled. Since all the subnetworks had at least one image component, the rescaled learning rate $\beta$ was always much smaller than 1.

**Viable subnetworks**. Only five subnetworks were viable at the end of the process (Fig. 4), namely $V_{1-5}^{(2)} = p(V_7^{(1)}|c_t, K)$, where $K = \{m_t^1, p_t^1, V_{1,2,3}^{(1)}\}$. Three more subnetworks "survived" the learning and pruning process, but held no information: $V_6^{(2)} = p(p_t^1|m_t^1, V_7^{(1)})$, $V_7^{(2)} = p(p_t^1|c_t, V_1^{(1)})$ and $V_7^{(2)} = p(p_t^1|c_t, V_3^{(1)})$. Examining $V_{1-5}^{(2)}$ reveals that in the implemented learning schedule, i.e. only a moving arm, they are all similar; they learn the transformation from image patches, $c_t$, to moving objects in the visual field $V_7^{(1)}$. Since the only moving object in the training set was the arm itself, they all represent *autonomous learning* of visual self-recognition, Fig. 2(d).

Figure 4 shows the execution output of the 8 viable subnetworks of level two. As can be seen, $V_{1-5}^{(2)}$ are practically identical. While $V_6^{(2)}$ seems to hold some information, the distinction between the arm and background is extremely small.

**Novel features**. Other works have shown autonomous learning of visual self-recognition [20, 17, 5], yet none have done so in a hierarchical manner starting from raw sensory data. Rather, all have used intensive pre-programmed image processing prior to learning. Furthermore, while we have not used the fully exhaustive second-level subnetwork pool, due to computational hardware limitations, this mapping *emerged* as the only viable and functional mapping out of 72 others.

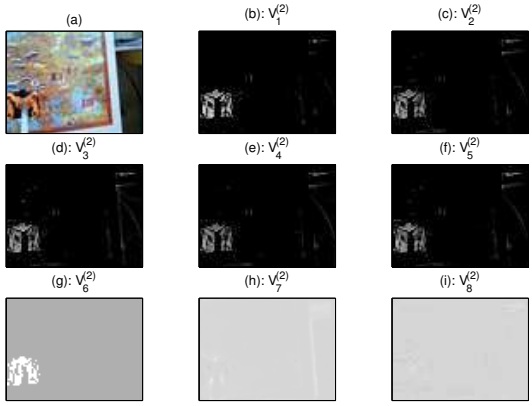



**Figure 4.** Execution map of the viable level 2 subnetworks.

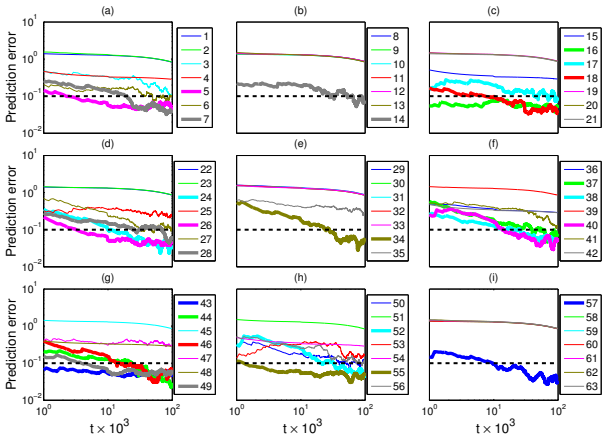



**Figure 5.** Prediction error of all 63 subnetworks of level 3 (see Appendix). Reliable networks are emphasized. Dashed black line delineate reliability threshold.

### *5.1.3 Third level*

Since we have shown the equivalence of the five viable second-level subnetworks, we can augment the data set with only one, e.g. $V_1^{(2)}$. This still results in a large exhaustive pool of subnetworks, so we made the same restrictions as in the second level. Furthermore, we have introduced a modification to the execution of second level viable subnetworks whose output was a single-channel image. This is usually due to *detection* of visual objects and hence its magnitude is not important, only its sign. We therefore changed: $\text{output} \in [-1, 1] \Rightarrow \text{output} \in \{-1, 1\}$, by thresholding at 0. This only expedited the learning of the higher levels, and does not change the results of the paper. For example, the execution output $V_5^{(2)} = p(V_7^{(1)}|c_t, p_t^1)$ was set to be the sign of the output neuron.

These alterations result in $||L^{(3)}|| = 63$ subnetworks, Fig. 5 (see Appendix). Again, this is a rather small pool, but can still demonstrate the basic principles presented here. All the subnetworks had two hidden layers with ten neurons each, with parameters $\beta = 0.01, \alpha = 0.1$. The robot moved its arm motor randomly for 100,000 time steps, in which only learning was performed on all subnetworks in parallel. Since this is the last level of the hierarchy and the subnetworks had numerous input neurons (see below), pruning proved to be an inefficient mechanism for distilling viable networks. Hence we employ a prediction error threshold, $e_{\text{threshold}} = 0.1$; subnetworks whose average prediction error did not decline below it at the end of the learning stage were deemed *unreliable*, Fig. 5.

The third and last level of the hierarchy represents the highest level in the visual pathway, one that considers the entire field of view [6]. Hence, $c_t$ in this level was set to be the entire image and in contrary to the previous two levels, a single presentation was done in each time step. Furthermore, rescaling of $\beta$ was not required.

**Viable subnetworks**. Twenty subnetworks ended up reliable, $V_{1-20}^{(3)}$ (see Appendix). However, they have several things in common: $V_1^{(2)}$ is always an input (and not an output), and the output consists of arm parameters (not image, or change detection). From this and further analysis, one can interpret the learned correlation: it is the transformation between the visual location of the hand, via $V_1^{(2)}$, and the current arm position. Hence, this constitutes *autonomous learning* of visual-to-arm coordinate transformation.

**Novel features**. Previous works have described algorithms for coordinate transformations [22]. However, they employed intense image processing prior to learning. Furthermore, this is the first time, to the best of our knowledge, that the same algorithm produces motion detection, visual self-recognition and visual-to-arm coordinate transformation.

## 5.2 Reaching Network

We next wish to construct a *closed loop* wiring between learned subnetworks that begins with sensory information and ends up in a motor command. By *closed loop* we mean that there is no external goal or task, but that the motion of the motors are determined by the wiring itself. Hierarchical construction followed by pruning resulted in the following viable and reliable subnetworks: nine networks from the first level, five equivalent subnetworks from the second level and twenty equivalent subnetworks from the third and last level.

We focus on the motion of the arm only, hence the subnetwork at the end of the closed loop must have an output of an arm motor-related parameter, Fig. 6. The arm's inverse model, $V_1^{(1)}$ is the natural subnetwork that obeys this requirement. However, some of the third level subnetworks, $V_{13-18}^{(3)}$ (see Appendix), have the inverse model as their output and thus also obey that requirement.

The image-arm coordinate transformation subnetworks, $V_{1-12}^{(3)}$ (see Appendix) are the only non-trivial subnetworks that can serve as an input to the arm's inverse model. This is so because these subnetworks have either $p_t^1$ or $V_2^{(1)}$, which is the forward model of the arm, as their output. This equivalent class transforms visual self-recognition, $V_1^{(2)}$ to the arm's angle, regardless of the other input, e.g. $m_t^1, c_t$. Hence, these are actually a $1 \mapsto 1$ subnetworks. However, since they are third-level subnetworks, they have an input of *the whole image*, resulting in 4800 input neurons.

We focus on the arm inverse model. During execution, the inverse model receives the current position of the arm and should receive a goal position. However, since we are interested in a closed loop, the goal should come from another subnetwork. The viable candidate subnetworks are $V_{2,3}^{(1)}$ and $V_{1-12}^{(3)}$, where the former group closes a trivial loop of forward/inverse models of the same motor. The latter group are equivalent and transform a visual image to an arm coordinate, so for simplicity we choose $V_1^{(3)} = p(p_t^1|V_1^{(2)}, m_t^1)$. The wiring of $V_1^{(3)} \rightarrow V_1^{(1)}$ results in motion towards a visual object. Finally, $V_1^{(3)}$ receives a *single channel* full image as its input. Six viable

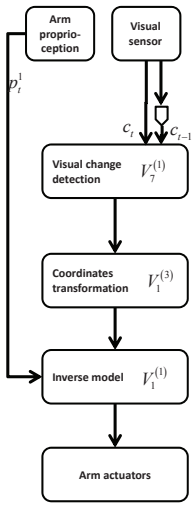



**Figure 6.** Reaching close-loop network, connecting viable subnetworks $V_7^{(1)} \rightarrow V_1^{(3)} \rightarrow V_1^{(1)}$.

subnetworks produce this output, namely, $V_{1-5}^{(2)}$ and $V_7^{(1)}$. The former five are equivalent, representing the arm's visual self-recognition (Fig. 2(d)) and result in trivial motion of the arm towards itself.

The latter's motion detection capabilities (Fig. 2(c)) close the loop: $(c_t, c_{t-1}) \rightarrow V_7^{(1)} \rightarrow V_1^{(3)} \rightarrow V_1^{(1)}$. In words, two consecutive images produce a change map; coordinate transformation transfers the location of detected changes into the arm's position; the arm's inverse model produces the correct arm motor command to reach towards the moving object, Fig. 6.

As mentioned above, $V_{13-18}^{(3)}$ can also be the final subnetworks of the closed loop (see Appendix). Analyzing their input/output relations show that they are equivalent and can serve as a combination of visual-to-arm coordinate transformation *and* inverse model. This means that one can replace the wiring $V_7^{(1)} \rightarrow V_{1-12}^{(3)} \rightarrow V_1^{(1)}$ described above with $V_7^{(1)} \rightarrow V_{13-18}^{(3)}$. We have not done so in the robot implementation in order to show that several subnetworks can be executed and concurrently re-learned.

We wish to emphasize that the presented execution circuit is the *only* functional non-trivial closed loop composed of viable and reliable subnetworks that operate the arm. The other possible wirings that result in motion of the arm perform trivial motions, such as moving towards the current position of the arm, i.e. not moving. While we have introduced learning schedule restrictions in the second and third levels, the process has still demonstrated extreme convergence: from a total of $168+72+63 = 303$ initial subnetworks, exuberance, pruning and reliability reduced the pool to a mere $9+5+20 = 34$ viable and reliable subnetworks, which can produce a single *emergent* functional closed loop, namely, reaching a moving target.

## 5.3 Concurrent Execution and Learning

Since all the components of the reaching network can be autonomously learned, the network possesses a unique characteristic: it can continuously and autonomously re-learn all its elements. We have shown that the inputs to the viable subnetworks are different in the learning and execution phases. Hence, during execution of the reaching motion, re-learning requires another "pass" through the learning network. Thus, for example, during concurrent learning, the coordinate transformation subnetwork $V_1^{(3)}$ should receive its input from the self-recognition subnetwork, $V_1^{(2)}$, and not the motion detection one $V_7^{(1)}$, as in the execution phase. This requires the maintenance and re-learning of subnetworks that do not actually contribute to the reaching task, e.g. $V_1^{(2)}$.

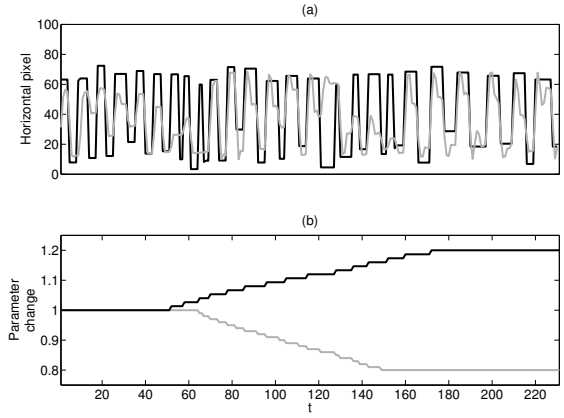



**Figure 7.** Reaching a moving object with concurrent learning (see Supp. Movie). (a) Horizontal position of moving object (black) and arm (gray) as a function of time steps. (b) Percent change of motor (black) and camera (gray) parameters.

Furthermore, we have implemented a specific arm and camera motion schedule to improve learning during the learning phase: motion detection was learned quickly because the camera consecutively moved and then rested; arm visual self-recognition and coordinate transformation were learned when only the arm moved. This means that during the reaching execution stage, some biases could be introduced. For example, if motion detection were learned only when the *arm* moved, it would re-learn that only arm features contribute to motion detection and would not average out all possible pixel combinations to produce a true motion detector. Hence, in the sequence demonstrated in Fig. 7(a), motion detection was not re-learned, due to the camera's immobility. However, the rest of the components, namely, $V_1^{(1)}, V_1^{(2)}$ and $V_1^{(3)}$ were learned concurrently with the reaching execution.

In the testing of the entire reaching network, we have added another LEGO Mindstorm motor that controlled a moving object, Fig. 2(a,blue object;c). It was moved to a random position to the left and then right of the image where it then made a sudden move, Fig. 7(a, black). The reaching network was then executed and monitored via the camera, Fig. 2(a,red hand;d). We have also introduced gradual changes to the robot's sensors and motors to ascertain its concurrent execution and re-learning capabilities.

Figure 7(a) shows the horizontal location of the two detected entities, namely, arm and object and nicely demonstrates how the arm follows the movement of the object. The figure first shows initial calibration errors, corrected after 70 times steps (see Supp. Movie). It is followed by an introduction of a slow change in the motor plant of the arm, whose maximal power increased by $20\%$ and a degradation of the red channel of the camera to $80\%$ of its value, Fig. 7(b). As can be seen in Fig. 7(a), the reaching motion is again accurate after the change due to the network's concurrent re-learning capabilities.

## 6 Related Works

In [5] an information theoretic approach was taken in order to autonomously learn what a robot can control. By using mutual information and moving its own hand, the robot could discern its hands and then its fingers. However, the model included image processing in order to extract the relevant information measures.

Ref. [30] explores several models of learning how to reach, whose main features are the learning of internal models, namely, forward and inverse kinematics and dynamics of the arm. While the suggested model autonomously learns these internal models by moving the arm, it does not address the issue of arm self-recognition, but rather uses intense image processing to acquire the external coordinates of the arm and the objects it interacts with.

Another humanoid robot was used to autonomously learn the coordinates transformation between the head and the arm [22]. This was done by using a fixed gaze by which the head was turned to keep the hand in the center of the image. Then the head and arm proprioception angle information was used to autonomously learn the coordinates transformation.

Work of the same group [20] has also implemented learning of the self via periodic motions of the hand and finding the corresponding image features. Furthermore, building saliency maps of the visual image and interacting with the environment, enabled building object models in the vicinity of the robot.

Building hierarchical learning networks that extract higher order correlations in the data was also performed by several groups [12, 25]. However, they have focused on image processing, while the model presented here focuses more on the interaction between the motorized action and sensors, both visual and proprioception. Ref. [33] learns generalized value functions which indeed relate states and actions, but does not perform an exhaustive search over these possibilities. Another related model is that of Hierarchical Temporal Memory [9], which learns temporal sensory information in different hierarchies, where the time-scales change with hierarchy level. The model presented here focuses more on the sensory-*motor* correlations with the emphasis of creating a functional executable circuit from the learned and viable subnetworks. This novel focus highlights the relevance of active sensing [4, 28] to the autonomous learning paradigm.

In the current implementation, we have used random motion in order to learn the sensory-motor correlations. However, many works have implemented active learning concepts to expedite such learning (see [10] and references therein). More specifically, the concept of intrinsic reward that originates from learning these transformation is a promising avenue of research [24, 32, 11] and will be explored in future work.

## 7 Discussion

A brain-inspired novel algorithm implementing the prevalent concept of exuberance [13] was introduced: it starts by constructing an exhaustive pool of subnetworks and then during learning prunes away those that are presented with uncorrelated data sets. Repeating the procedure in a hierarchical manner results in a pool of *viable* and *reliable* subnetworks that represent *all* the correlations the agent can autonomously learn. These serve as an alphabet to construct executable circuits that perform specific tasks, with concurrent autonomous learning of all composing elements.

The complexity analysis presented above can hint to the underlying cause of exuberance in the brain [13]. Initially, the organism does not know which sensory information correlates with which motor command. Hence, starting with an exhaustive connectivity and then pruning away the non-functional elements in a hierarchical manner results in reduced hierarchical complexity.

We have focused on $2 \mapsto 1$ correlations and not all-to-all correlations, as in self-organizing maps [27], in order to show the emergence of specific functional networks, e.g. motion detection and self-recognition. Had we constructed the first level network to have all the inputs and one output, e.g. the camera motor command $p(m_t^2|D^{(1)})$, it would not have learned motion detection $p(m_t^2|c_t, c_{t-1})$, since the camera motor inverse model $p(m_t^2|p_t^2, p_{t-1}^2)$ would have better predicted the motor command. One may thus conclude that dividing the input information to subsets is beneficial for constructing a truly exhaustive map of learnable transformations.

Furthermore, the architecture presented here used a large initial network and then utilized pruning. One may have opted for starting with a small network and increasing it via, e.g. cascade correlation networks [7, 16, 31]. However, since there is evidence for death of inactive neurons, but less of increase in the number of neurons in the brain, we believe that pruning connections and non-active neurons is more suitable to our brain-inspired framework than adding neurons to a network. A thorough comparison of the performance of the two options is beyond the scope of this paper.

The proposed model and its implementation give rise to several interesting aspects of autonomous learning in general. From a neurobiological perspective, the specific suggested architecture for learning how to reach has a novel prediction that suggests connectivities that are mandatory in order to achieve learning, e.g. an efference copy of the eye muscles must reach the first motion detection station, which is as low as the retina. However, it seems unlikely that basic change detection is a learned quality of the nervous system, since many (if not all) low-level neurons have that characteristic. How can this be resolved? First, the question we ask is fundamental: what can be learned, expanding beyond what indeed is learned in biological systems. Second, evolution may also play a significant role, i.e. it is possible that lower species have a learned motion detection architecture, but higher ones developed a genetically-encoded mechanism to achieve the same goal. This has the added advantage of requiring less time to manifest, meaning an organism with a "hard-wired" motion detector will detect motion earlier in development than one that has to learn it. Furthermore, it enables the learning neuronal system to focus on more complicated sensory-motor correlations, i.e. higher loops. Third, the biological substrates that living organisms are made of have some inherent qualities, and change detection, or other simple transformations, may be one of them, thus eliminating the need to learn it during development.

Another important issue is what determines the overall execution connectivity of the viable subnetworks. A possible biological reasoning is that initially all the subnetworks are connected to each other and only those that serve some purpose and achieve a specific (rewarding) goal survive in development. Hence, one can picture a fully connected network in which all available information serve as both input and output to many correlation-learning neuronal networks. These networks are then only way-stations to other neuronal networks that serve as the second tier in the hierarchy and so on. Selecting which network will be activated and which will control the muscles at any given moment probably involves a rewarding mechanism which is beyond the scope of the present work.

The exhaustive constructive algorithm has also surfaced a novel concept relating active sensing [4, 28] and autonomous learning: in the first level of the hierarchy, *only* active sensing sensory-motor cou-

plings produce viable subnetworks and those represent internal models. Specifically, the inverse model subnetworks can be used in two complementary ways: (i) determining the proper motor command to achieve a specific sensory goal and; (ii) detecting changes in the sensory information.

Finally, the proposed implementation of the exhaustive architecture is just the first step towards a more complex network with many more capabilities. One can think of new subnetworks in all levels of the hierarchy: a network that learns to map a visual receptive field to the specific directional motion of the eye can learn orientation lines and edge detection; using two cameras, one can autonomously learn a stereoscopic coordinates transformation; head, torso and leg movements can also be autonomously learned and add more flavor to a much richer network.

To conclude, we have developed a methodology of an exhaustive hierarchical construction of autonomously learning agents. We have shown that by implementing exuberance and pruning, only viable and reliable networks that actually encode correlations in the sensory-motor information survive. Those augment higher levels' available data to introduce more complex subnetworks. We have implemented it on a robot and showed that three levels of the hierarchy can produce a completely autonomously learned reaching arm, that is robust to noise due to its concurrent execution and re-learning capabilities. We envision that this model can be implemented on virtually any robotic agent and can augment existing pre-programed algorithmic controls.

## Appendix: Level 3 Subnetworks

$$
\begin{array}{ll}
L_1^{(3)} = p(V_1^{(2)}|p_t^1, m_t^1) & L_2^{(3)} = p(V_1^{(2)}|p_t^1, c_t) \\
L_3^{(3)} = p(m_t^1|p_t^1, V_1^{(2)}) & L_4^{(3)} = p(c_t|p_t^1, V_1^{(2)}) \\
L_5^{(3)} = p(V_1^{(1)}|p_t^1, V_1^{(2)}) & L_6^{(3)} = p(V_3^{(1)}|p_t^1, V_1^{(2)}) \\
L_7^{(3)} = p(V_2^{(1)}|p_t^1, V_1^{(2)}) & L_8^{(3)} = p(V_7^{(1)}|p_t^1, V_1^{(2)}) \\
L_9^{(3)} = p(V_1^{(2)}|p_t^1, V_1^{(1)}) & L_{10}^{(3)} = p(V_1^{(2)}|p_t^1, V_3^{(1)}) \\
L_{11}^{(3)} = p(V_1^{(2)}|p_t^1, V_2^{(1)}) & L_{12}^{(3)} = p(V_1^{(2)}|p_t^1, V_7^{(1)}) \\
L_{13}^{(3)} = p(V_1^{(2)}|m_t^1, c_t) & L_{14}^{(3)} = p(p_t^1|m_t^1, V_1^{(2)}) \\
L_{15}^{(3)} = p(c_t|m_t^1, V_1^{(2)}) & L_{16}^{(3)} = p(V_1^{(1)}|m_t^1, V_1^{(2)}) \\
L_{17}^{(3)} = p(V_3^{(1)}|m_t^1, V_1^{(2)}) & L_{18}^{(3)} = p(V_2^{(1)}|m_t^1, V_1^{(2)}) \\
L_{19}^{(3)} = p(V_7^{(1)}|m_t^1, V_1^{(2)}) & L_{20}^{(3)} = p(V_1^{(2)}|m_t^1, V_1^{(1)}) \\
L_{21}^{(3)} = p(V_1^{(2)}|m_t^1, V_3^{(1)}) & L_{22}^{(3)} = p(V_1^{(2)}|m_t^1, V_2^{(1)}) \\
L_{23}^{(3)} = p(V_1^{(2)}|m_t^1, V_7^{(1)}) & L_{24}^{(3)} = p(p_t^1|c_t, V_1^{(2)}) \\
L_{25}^{(3)} = p(m_t^1|c_t, V_1^{(2)}) & L_{26}^{(3)} = p(V_1^{(1)}|c_t, V_1^{(2)}) \\
L_{27}^{(3)} = p(V_3^{(1)}|c_t, V_1^{(2)}) & L_{28}^{(3)} = p(V_2^{(1)}|c_t, V_1^{(2)}) \\
L_{29}^{(3)} = p(V_7^{(1)}|c_t, V_1^{(2)}) & L_{30}^{(3)} = p(V_1^{(2)}|c_t, V_1^{(1)}) \\
L_{31}^{(3)} = p(V_1^{(2)}|c_t, V_3^{(1)}) & L_{32}^{(3)} = p(V_1^{(2)}|c_t, V_2^{(1)}) \\
L_{33}^{(3)} = p(V_1^{(2)}|c_t, V_7^{(1)}) & L_{34}^{(3)} = p(p_t^1|V_1^{(2)}, V_1^{(1)}) \\
L_{35}^{(3)} = p(m_t^1|V_1^{(2)}, V_1^{(1)}) & L_{36}^{(3)} = p(c_t|V_1^{(2)}, V_1^{(1)}) \\
L_{37}^{(3)} = p(V_3^{(1)}|V_1^{(2)}, V_1^{(1)}) & L_{38}^{(3)} = p(V_2^{(1)}|V_1^{(2)}, V_1^{(1)}) \\
L_{39}^{(3)} = p(V_7^{(1)}|V_1^{(2)}, V_1^{(1)}) & L_{40}^{(3)} = p(p_t^1|V_1^{(2)}, V_3^{(1)}) \\
L_{41}^{(3)} = p(m_t^1|V_1^{(2)}, V_3^{(1)}) & L_{42}^{(3)} = p(c_t|V_1^{(2)}, V_3^{(1)}) \\
L_{43}^{(3)} = p(V_1^{(1)}|V_1^{(2)}, V_3^{(1)}) & L_{44}^{(3)} = p(V_2^{(1)}|V_1^{(2)}, V_3^{(1)}) \\
L_{45}^{(3)} = p(V_7^{(1)}|V_1^{(2)}, V_3^{(1)}) & L_{46}^{(3)} = p(p_t^1|V_1^{(2)}, V_2^{(1)}) \\
L_{47}^{(3)} = p(m_t^1|V_1^{(2)}, V_2^{(1)}) & L_{48}^{(3)} = p(c_t|V_1^{(2)}, V_2^{(1)}) \\
L_{49}^{(3)} = p(V_1^{(1)}|V_1^{(2)}, V_2^{(1)}) & L_{50}^{(3)} = p(V_3^{(1)}|V_1^{(2)}, V_2^{(1)}) \\
L_{51}^{(3)} = p(V_7^{(1)}|V_1^{(2)}, V_2^{(1)}) & L_{52}^{(3)} = p(p_t^1|V_1^{(2)}, V_7^{(1)}) \\
L_{53}^{(3)} = p(m_t^1|V_1^{(2)}, V_7^{(1)}) & L_{54}^{(3)} = p(c_t|V_1^{(2)}, V_7^{(1)}) \\
L_{55}^{(3)} = p(V_1^{(1)}|V_1^{(2)}, V_7^{(1)}) & L_{56}^{(3)} = p(V_3^{(1)}|V_1^{(2)}, V_7^{(1)}) \\
L_{57}^{(3)} = p(V_2^{(1)}|V_1^{(2)}, V_7^{(1)}) & L_{58}^{(3)} = p(V_1^{(2)}|V_1^{(1)}, V_3^{(1)}) \\
L_{59}^{(3)} = p(V_1^{(2)}|V_1^{(1)}, V_2^{(1)}) & L_{60}^{(3)} = p(V_1^{(2)}|V_1^{(1)}, V_7^{(1)}) \\
L_{61}^{(3)} = p(V_1^{(2)}|V_3^{(1)}, V_2^{(1)}) & L_{62}^{(3)} = p(V_1^{(2)}|V_3^{(1)}, V_7^{(1)}) \\
L_{63}^{(3)} = p(V_1^{(2)}|V_2^{(1)}, V_7^{(1)}) & .
\end{array}
\tag{6}
$$

The twenty viable subnetworks can be grouped into three equivalent classes:

1. Subnetworks with arm angle-related output: $V_{1-12}^{(3)} = L_{14,24,34,40,46,52,7,18,28,38,44,57}^{(3)}$. Notice that $V_2^{(1)}$ is the forward model of the arm motor and during training is completely equivalent to $p_t^1$.
2. Subnetworks with arm motor-related output: $V_{13-18}^{(3)} = L_{5,16,26,43,49,55}^{(3)}$. Notice that $V_1^{(1)}$ which appears as an output in these subnetworks is the inverse model of the arm. During training it is completely equivalent to $m_t^1$.
3. Subnetworks with the arm's previous angle output: $V_{19-20}^{(3)} = L_{17,37}^{(3)}$. Notice that $V_3^{(1)}$ which appears the output in these subnetworks is the postdiction model of the arm and during training is completely equivalent to $p_{t-1}^1$.

## REFERENCES

[1] D. H. Ballard, 'Animate vision', *Artif. Intell.*, **48**(1), 57–86, (1991).
[2] N. E. Berthier and R. Keen, 'Development of reaching in infancy', *Exp Brain Res*, **169**(4), 507–18, (2006).
[3] T. B. Crapse and M. A. Sommer, 'Corollary discharge across the animal kingdom', *Nat Rev Neurosci*, **9**(8), 587–600, (2008).

[4] K. E. Cullen, 'Sensory signals during active versus passive movement', *Curr Opin Neurobiol*, **14**(6), 698–706, (2004).
[5] A. Edsinger and C. C. Kemp, 'What can i control? a framework for robot self-discovery', in *The sixth international conference on epigenetic robotics (EpiRob)*.
[6] R. Epstein and N. Kanwisher, 'A cortical representation of the local visual environment', *Nature*, **392**(6676), 598–601, (1998).
[7] S.E. Fahlman and C. Lebiere. The cascade-correlation learning architecture, 1990.
[8] N. Fnaiech, F. Fnaiech, and B. Jervis, *Feedforward Neural Networks Pruning Algorithms*, 1–16, Electrical Engineering Handbook, CRC Press, 2011.
[9] D. George and J. Hawkins, 'Towards a mathematical theory of cortical micro-circuits', *PLoS Comput Biol*, **5**(10), e1000532, (2009).
[10] G. Gordon, D. M. Kaplan, B. Lankow, D. Y. Little, J. Sherwin, B. A. Suter, and L. Thaler, 'Toward an integrated approach to perception and action: conference report and future directions', *Front Syst Neurosci*, **5**, 20, (2011).
[11] G. Gordon and E. Ahissar, 'Reinforcement active learning hierarchical loops', in *International Joint Conference on Neural Networks (IJCNN)*.
[12] G. E. Hinton and R. R. Salakhutdinov, 'Reducing the dimensionality of data with neural networks', *Science*, **313**(5786), 504–7, (2006).
[13] G. M. Innocenti and D. J. Price, 'Exuberance in the development of cortical networks', *Nat Rev Neurosci*, **6**(12), 955–965, (2005).
[14] M. I. Jordan, 'Forward models: Supervised learning with a distal teacher', *Cognitive Science*, **16**, 307–354, (1992).
[15] N. Kanwisher, J. McDermott, and M. M. Chun, 'The fusiform face area: a module in human extrastriate cortex specialized for face perception', *J Neurosci*, **17**(11), 4302–11, (1997).
[16] M. Kawato, Y. Maeda, Y. Uno, and R. Suzuki, 'Trajectory formation of arm movement by cascade neural network model based on minimum torque-change criterion', *Biol Cybern*, **62**(4), 275–88, (1990).
[17] C. C. Kemp and A. Edsinger, 'What can i control?: The development of visual categories for a robot's body and the world that it in uences.', in *5th IEEE International Conference on Development and Learning (ICDL5): Special Session on Perceptual Systems and their Development*.
[18] H. Lalazar and E. Vaadia, 'Neural basis of sensorimotor learning: modifying internal models', *Curr Opin Neurobiol*, (2008).
[19] E. P. Merriam and C. L. Colby, 'Active vision in parietal and extrastriate cortex', *The Neuroscientist*, **11**(5), 484–493, (2005).
[20] L. Natale, F. Orbona, G. Metta, and G. Sandini, 'Exploring the world through grasping: a developmental approach', in *IEEE International Symposium on Computational Intelligence in Robots and Automation*.
[21] D. Nguyen-Tuong, J. Peters, M. Seeger, and B. Schlkopf, 'Learning inverse dynamics: A comparison', in *European Symposium on Artificial Neural Networks (ESANN 2008)*, pp. 13–18.
[22] F. Nori, L. Natale, G. Sandini, and G. Metta, 'Autonomous learning of 3d reaching in a humanoid robot', in *IEEE/RSJ International Conference on Intelligent Robots and Systems*.
[23] J. Kevin O'Regan, No, euml, and Alva , 'A sensorimotor account of vision and visual consciousness', *Behavioral and Brain Sciences*, **24**(05), 939–973, (2001).
[24] P. Y. Oudeyer, F. Kaplan, and V. V. Hafner, 'Intrinsic motivation systems for autonomous mental development', *Evolutionary Computation, IEEE Transactions on*, **11**(2), 265–286, (2007).
[25] M. Ranzato, Y. lan Boureau, and Y. Lecun, 'Sparse feature learning for deep belief networks', in *Advances in Neural Information Processing Systems*.
[26] R. Reed, 'Pruning algorithms-a survey', *Neural Networks, IEEE Transactions on*, **4**(5), 740–747, (1993).
[27] H. Ritter, 'Self-organizing maps for robot control', in *Proceedings of the 7th International Conference on Artificial Neural Networks* (1997).
[28] C. E. Schroeder, D. A. Wilson, T. Radman, H. Scharfman, and P. Lakatos, 'Dynamics of active sensing and perceptual selection', *Curr Opin Neurobiol*, **20**(2), 172–6, (2010).
[29] R. Shadmehr and J. W. Krakauer, 'A computational neuroanatomy for motor control', *Exp Brain Res*, **185**(3), 359–81, (2008).
[30] R. Shadmehr, 'Generalization as a behavioral window to the neural mechanisms of learning internal models', *Human Movement Science*, **23**, 543–568, (2004).
[31] S. K. Sharma and P. Chandra, 'Constructive neural networks: A review', *International Journal of Engineering Science and Technology*, **2**(12), 7847–7855, (2010).
[32] S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg, 'Intrinsically motivated reinforcement learning: An evolutionary perspective', *Autonomous Mental Development, IEEE Transactions on*, **2**(2), 70–82, (2010).
[33] Richard S. Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M. Pilarski, Adam White, and Doina Precup, 'Horde: a scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction', in *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2* (2011).
[34] E. Todorov and Z. Ghahramani, 'Unsupervised learning of sensory-motor primitives', in *25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. **2**, 1750-1753, (2003).

4th International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2011)
Berlin, Germany, December 9 2011

# An Application of Genetic Algorithms to Model Leg–Soil Interaction

**Malte Römmermann**[1] and **Mohammed Ahmed**[1] and **Lorenz Quack**[1] and **Yohannes Kassahun**[2]

**Abstract.** This paper introduces an application of a genetic algorithm to generate a model for leg–soil interaction to be used in an interactive real time simulation. In the field of legged robotics, the use of walking and climbing robots becomes very useful for extraterrestrial applications, e.g., collection of samples from lunar crater beds. To efficiently simulate such a space mission, a realistic robot leg–soil interaction model is required. Using artificial neural networks as contact model for an interactive real time simulation is novel and this paper describes how the integration can be done. This paper deals only with the normal force of a foot soil contact. However, an outlook on how the lateral forces can be integrated is given as well.

A genetic algorithm is implemented to evolve artificial neural networks representing the leg–soil interaction. The data to evolve the neural networks with is collected by a series of experiments performed with an industrial robotic arm equipped with a six axes force/torque sensor and a state of the art walking and climbing robot's foot. The genetic algorithm evolves the structure and the parameters of the neural network. The paper describes the neural network, the genetic algorithm, and the indirect graph representation. Moreover, the integration of the model into a fully rigid body legged robot simulator is presented.

## 1 INTRODUCTION

Creating an analytical model for soil-contact mechanics is a well researched field of wheeled robotics [6, 7, 10]. Even if there are well defined physical soil-properties that are used within the analytical models, there are always several parameter that have to be chosen empirically. Thus, a realistic soil-contact model needs always a tuning by comparison with real measured experiment data. Due to the fact, that real data is always necessary, this paper describes an approach that approximates the real data by evolving a neural network. The neural network includes an interpolation of the real measurement data and is able to generalise the whole variance of the soil behaviour, which is measured due to slightly different soil compactions. The focus of this paper is a soil-contact model for a legged robot developed in the SpaceClimber project [13].

### 1.1 Review of Neuroevolution (NE)

The development of a neural network through an evolutionary algorithm is called neuroevolution (NE). In neuroevoution, an artificial neural network (ANN) is used to provide a straightforward mapping between inputs (states perceived by the sensors) and outputs (actions executed by the actuators). ANNs are useful in robotics since they are robust to noise. Their robustness comes from their inherent structure, i.e their units are typically based upon a sum of several weighted signals. Thus, oscillations in the individual values of these signals do not drastically affect the behaviour of the network.

The field of neuroevolution can be broadly divided along two major axes depending on which aspects of the ANN are encoded in the genotype and how the mapping from genotype to phenotype is defined. The first axis divides methods which evolve only the connection weights of the ANN but keep the structure (i. e. network topology) fixed [14, 5] from those methods which evolve structure and weights in parallel. The second axis divides methods where the mapping from genotype to phenotype is kept fixed during evolution from those methods which use an adaptive mapping which itself is subject to evolution. This mapping is also called an *embryogeny* or *development function*. According to Bentley and Kumar [1], three different types of embryogenies have been used in evolutionary systems: external, explicit, and implicit. *External* means that the developmental process (i. e. the embryogeny) itself is not subjected to evolution but is hand-designed and defined globally and externally to the genotypes In *explicit* (evolved) embryogeny the developmental process itself is explicitly specified in the genotypes, and thus it is affected by the evolutionary process [4, 3]. Usually, the embryogeny is represented in the genotype as a tree-like structure following the paradigm of genetic programming. The third kind of embryogeny is *implicit* embryogeny, which comprises neither an external nor an explicit internal specification of the growth process. Instead, the embryogeny "emerges" implicitly from the interaction and activation patterns of the different genes [2]. This kind of embryogeny has the strongest resemblance to the process of natural evolution.

In the past, neuroevolution methods have been applied to evolutionary robotics. In evolutionary robotics, many aspects of a robotic system can be evolved, ranging from the structure of its control architecture over certain modules of such an architecture to the evolution of the robot's morphology [11]. In this paper, we present such an application of neuroevolution to the modelling of leg soil interaction for a legged robot. The neuroevoution used in this paper evolves both the structure and parameters of the neural network but the developmental process (embryogeny) is fixed and not subjected to evolution.

## 2 NEURAL NETWORK

The neural network used for this paper is kept simple by using perceptron based neurones that can have connected inputs and outputs. The perceptron used is shown in figure 1. The inputs are weighted and afterwards combined by a transfer function. The

[1] German Research Center for Artificial Intelligence – DFKI Bremen-Robotics Innovation Center – Robert-Hooke-Str. 5, 28359 Bremen, Germany, email: {malte.roemmerman,mohammed.ahmed,lorenz.quack}@dfki.de
[2] Robotics Group – University of Bremen – 28359 Bremen, Germany, email: kassahun@informatik.uni-bremen.de

possible transfer functions are:
`sum, product, divide, min, max, average, and sine.`

If the absolute value of a neurone is greater than a defined threshold, the output is multiplied with a weight and the result defines the neurone's output. Otherwise, the output of the neurone is set to zero. The structure of the neural network itself is mainly defined by the genetic algorithm and will be discussed in section 5.1.

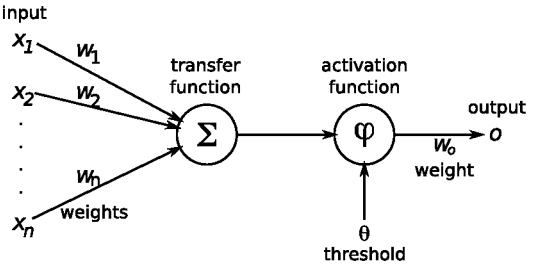



**Figure 1.** The architecture of a simple perceptron with weighted inputs, a transfer function, threshold based activation function, and a weighted output.

The second operation is the crossover operation. The crossover operation combines two parent objects by taking randomly nodes and connections from the first parent and attaches them to selected nodes and connections from the second parent. Two examples of the crossover operation are illustrated in figure 2.

The probability for a operation and/or sub-operation can be set and changed by the application. In this work the probability of the operations that change the structure of the graphs is reduced with the $\sigma$ decreasing. In this way, while the evolution converges the adaptation of the individuals gets focused on the parameters.

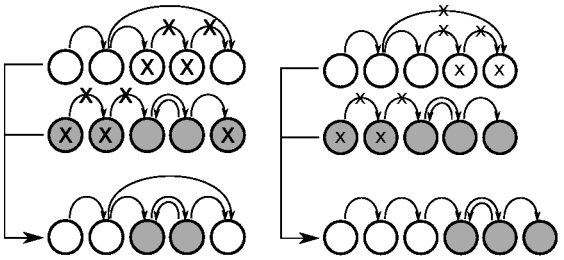


**Figure 2.** Two examples for the crossover operation. In each projection the upper two strings are combined into the lower new ones.

## 3 GENETIC ALGORITHM

Genetic algorithms are inspired by the natural evolution. A generation is a set of individuals that are evaluated in parallel. The best individuals of a generation are selected to represent the parent individuals of a new generation. By performing operations on the parent individuals a new generation is created. This process is repeated until a stopping criteria is fulfilled. A stopping criteria can be for instance, a desired performance of the individuals or the number of generations that are evaluated. In genetic algorithms one individual is called phenotype and is decoded from a string called chromosome or genotype. Thus, the genotype is the internal representation of an individual.

### 3.1 Genotype

The implementation of the genetic algorithm used in this paper is kept as generic as possible. The genotype is defined by a string of nodes including connections that are defined between nodes. Each node and connection can have a predefined number of parameters that are evolved together with the nodes and connections. The number of parameters and the decoding of the genotype into the phenotype is defined by the application.

### 3.2 Operations on Genotype

Two operations are implemented to create a new child genotype from one or two selected parent genotypes. The first operation is the mutation. The mutation operation is split into three sub-operations, whereby only one is used at the same time to create a child individual. All sub-operations take one string as parent object. The first sub-operation adds randomly a node or connection to the structure. The second sub-operation removes randomly a node or a connection and the third sub-operation changes the parameters with a normal distribution defined by a parameter sigma ($\sigma$).

### 3.3 Phenotype

The phenotype is the network decoded from the nodes and connections. Each neurone has three inputs: the penetration depth of a foot, a bias, and a variation parameter. The variation parameter is used to model different soil behaviour measured during the collection of real data. One node of the genotype represents a perceptron with the parameters: *used transfer function, input bias, and weights of the input and output signals*.

A connection of the genotype adds a new output to the first referenced perceptron and connects this output to a new input of the second referenced perceptron. The connection itself includes the references of the two nodes (perceptrons) that are connected and two parameters that are used for the construction of the neural network:

1. the weight of the new output of the first perceptron
2. the weight of the new input of the second perceptron

Additionally, the connections define whether they connect to an existing node or if the second referenced node is duplicated by the decoding of the genotype. Thus, also an indirect encoding is possible as shown in figure 3. After the generation of the neural network, the last

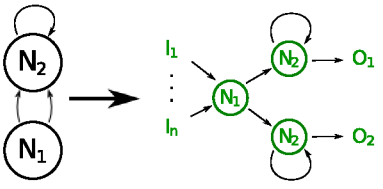


**Figure 3.** The figure depicts a possible indirect decoding from a genotype into a phenotype.

perceptron created by the decoding gets an additional output, which represents the overall network output and is used in the simulation as normal force of the foot-soil contact.

## 4 INTEGRATION

The SpaceClimber simulation is a DFKI internal software already used in the development process of the SpaceClimber [9]. Later on the simulated robot behaviour was evaluated and compared with the real developed robot [8]. The physical simulation is done by the Open Dynamics Engine (ODE) [12], which is a widely used rigid body simulation library.

To integrate the neural network into the simulation, the standard ODE contact points are used. An ODE contact point is created by the collision detection and includes parameters like the contact depth, a friction coefficient and ODE internal parameter like the error reduction parameter ($erp$) and the constraint force mixing parameter ($cfm$).

The $erp$ and $cfm$ parameter can be used to model spring/damping properties of a contact point. The ODE manual provides a formula to compute $erp$ and $cfm$ using a spring constant ($kp$), a damping constant ($kd$), and the simulation time step ($h$). The spring constant is calculated with the normal force output of the neural network ($f$) and the contact depth ($d$) with $kp = \frac{f}{d}$. A damping parameter is chosen manually with the policy to get as little damping as possible without a oscillating contact behaviour. The damping parameter used in the implementation is $kd = 100$.

Using the simple spring/damper implementation results in an incorrect foot–soil behaviour in the dynamic simulation. In the case of a foot contact with a current load $l$ a resulting contact depth is reached with a normal force equal to the load $l$. If the load is decreasing due to force distribution on other legs, the spring properties push the foot to a smaller depth, that results in a normal force corresponding to the new load. This behaviour is not observed in real soil, once the soil is displaced it will not create more force than the current load.

To adapt this behaviour a foot print implementation is done where every new foot soil contact is saved with the maximum immersion (deformation) and a random variance parameter that is used for the neural network. Technically, the ODE depth parameter of the contact point is reduced by the saved immersion before calculating $kp$. Afterwards, the new maximum immersion is saved in the foot print. Each new foot print is stored in a list with the position and the last penetration depth. In this way also the effect of re-entering a foot print is taken into account.

## 5 EXPERIMENTS

To test the functionality of the genetic algorithm a set of test functions are chosen. The complexity of the test functions is similar to the expected complexity of the foot-soil contact behaviour. The second part of this section shortly describes the collection of real experiment data.

### 5.1 Test Functions

The whole implementation is tested on predefined functions $f_n : \mathbb{R} \to \mathbb{R}$. The test functions are:

$$f1(x) = x^3 + 7 \qquad f2(x) = sin(x) \cdot x$$
$$f3(x) = sin(x) + 2x \qquad f4(x) = x^2 + 6 + 4sin(x)$$

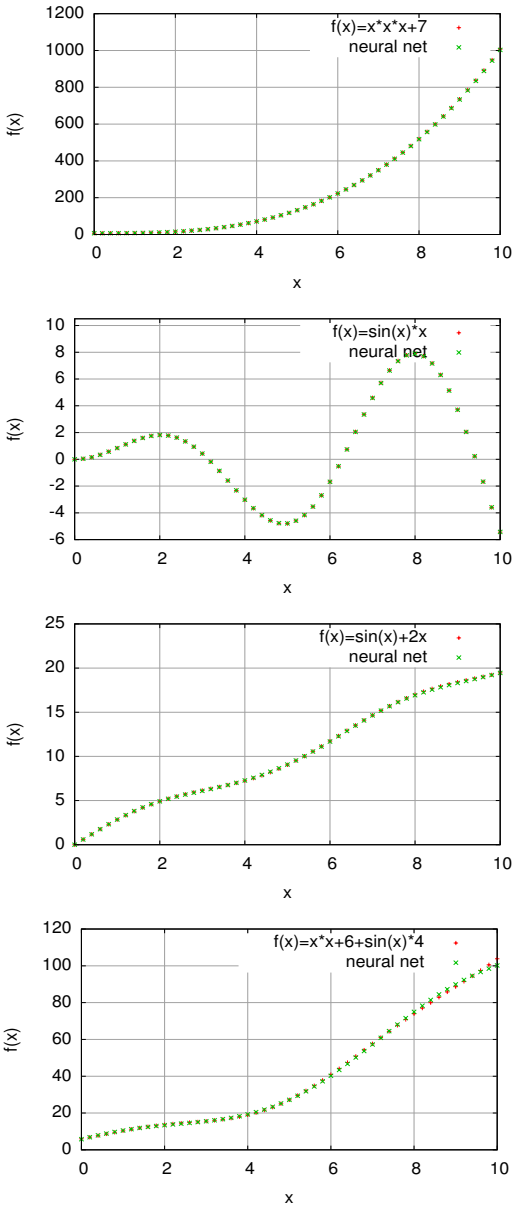



**Figure 4.** Results of the evolved networks plotted together with the test functions $f1$ - $f4$.

where $x$ is the test functions and the neural network input with $0 \leq x \leq 10$. The variance parameter introduced in section 3.3 is not used for the test functions. Altogether, the algorithm is able to find a close approximation of all test functions. The results are shown in figure 4.

### 5.2 Real Experiment Data

In order to obtain reference data for training and verification of the neural network a series of soil experiments is performed. The Schunk LWA-3 robotic arm with a six axes force/torque sensor and a spherical robotic foot attached is used. After positioning the foot above the ground it is pushed perpendicular to the surface into the soil at a constant speed of 1 mm/s. Between experiments the soil is manually loosened and levelled. Still there is noticeable variance in the result-

ing forces corresponding to slightly different preconditioning of the soil. A goal of this paper is to show that this variance which is expected also to occur in real missions can be modelled with the neural network approach. The experimental setup and the vertical component of the measured forces are shown in figure 5 and figure 6.

To obtain a variance parameter for every experiment test run, the forces of the curves at the maximum depth for each test run are linearly projected to a value between one and three, whereby, the minimum force value is projected to one and the maximum value to three.

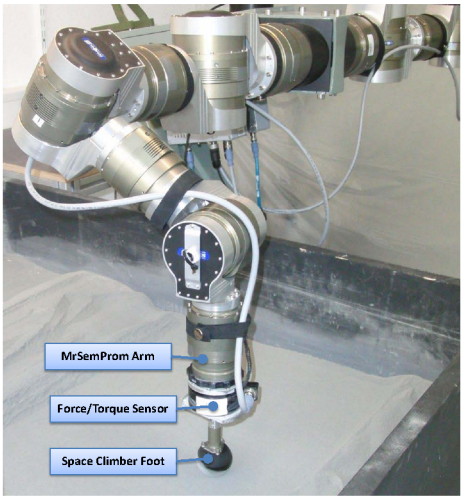



**Figure 5.** The experiment setup for measuring the real foot-soil behaviour.

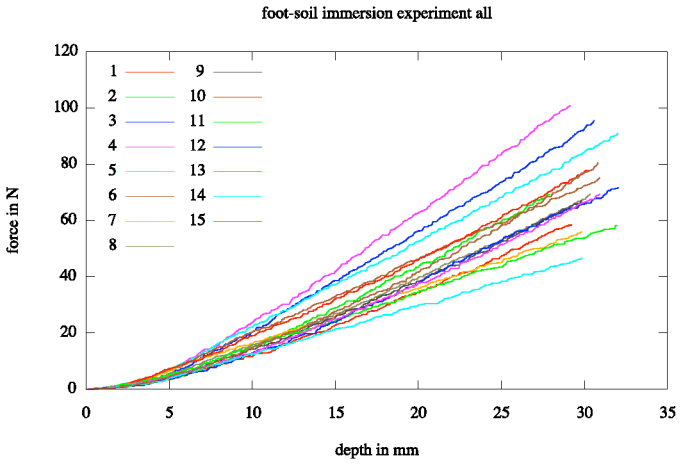



**Figure 6.** The resulting measured forces plotted over immersion depth.

## 5.3 Virtual Experiment Replica

In order to compare the integrated leg–soil interaction with the real experiment, the same manipulation arm setup is constructed in the simulation. Figure 7 shows the virtual setup. The experiment that is performed to compare the simulation with the reality is as follows: first dig the foot one centimetre into the soil, then lift it completely out of the soil, and afterwards dig three centimetre into it again.

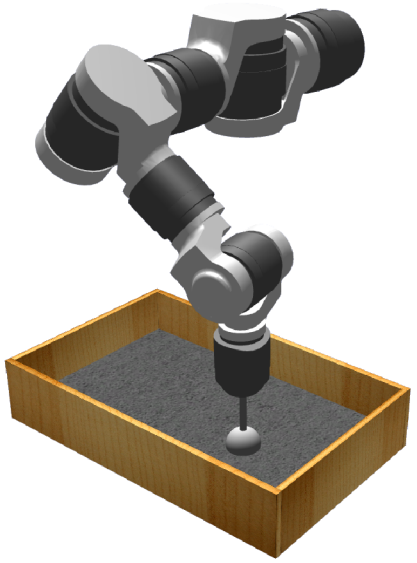


**Figure 7.** Virtual experiment setup to compare the integrated leg–soil interaction with the real experiment measurements.

## 6 RESULTS

The measured data from the real experiments (shown in figure **??**) are used to evolve a neural network. All experiment datasets are used for the network optimisation. The resulting force curve for the specified variance value is shown in figure 11, where every experiment dataset is plotted together with the corresponding network output. The generalisation properties of the network are shown in figure 9, where the whole network output is plotted within the defined variance value range. Figure 10 shows the result of the comparison of the fully integrated leg–soil model with the real experiment setup. The variance value for the experiment is chosen manually to fit the amplitude of the measured normal force. Figure 11 shows the SpaceClimber in the simulation with random variation parameters for the feet.

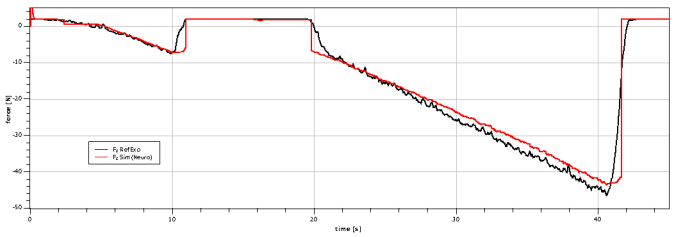


**Figure 10.** Result of the comparison of the fully integrated leg–soil model with the real experiment setup.

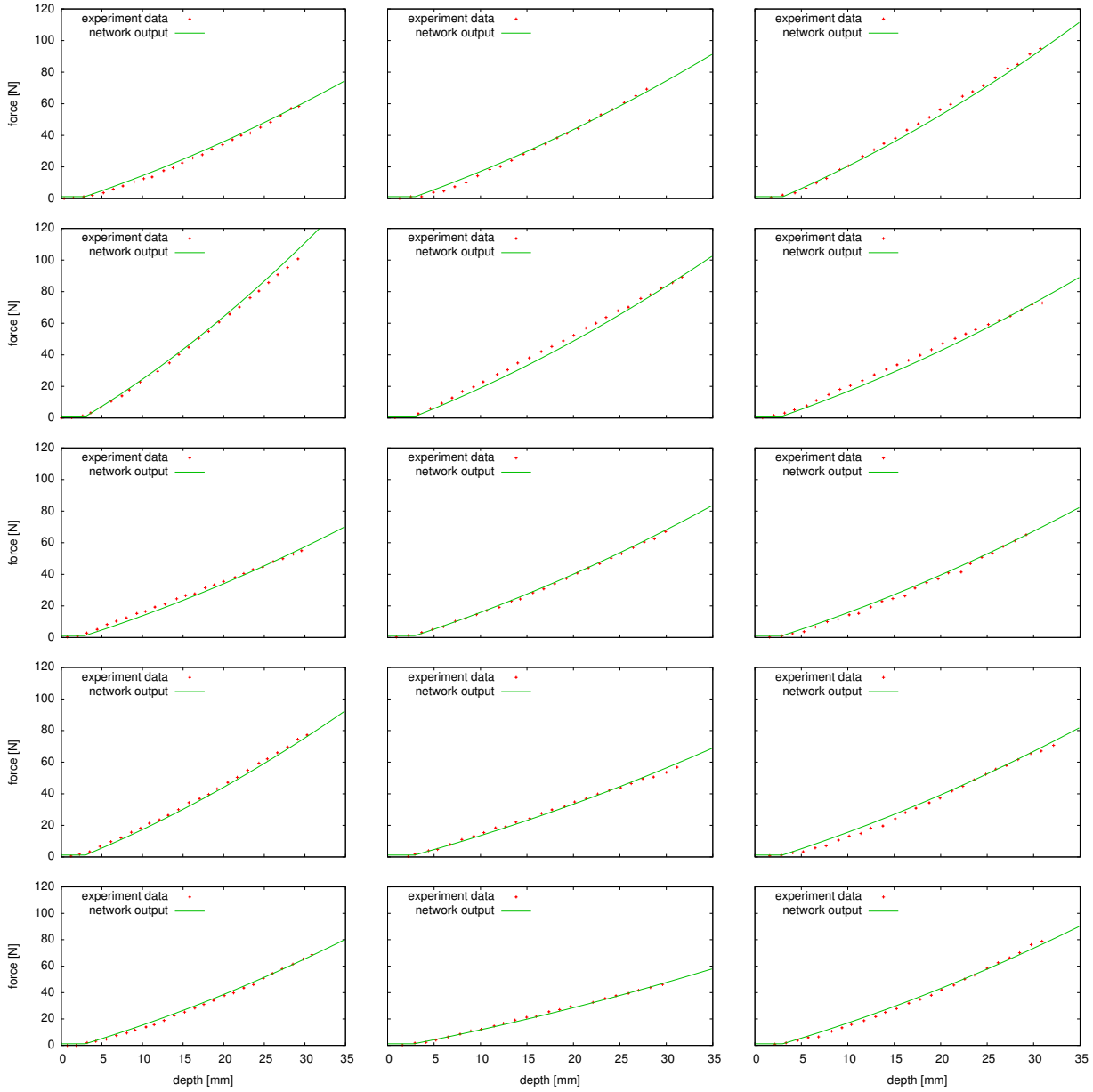



**Figure 8.** Comparison between experiment data and output of the neural network.

## 7 CONCLUSION

Altogether, the neural network approach is able to approximate the measured foot-soil behaviour for the contact normal force. The neural network represents a generalisation of the whole measured variance range and is already combined with a standard rigid body simulation. The calculation time of the network is fast enough to be used in an interactive real–time simulation on a standard desktop PC. Within this paper the whole measured experiment data is used to evolve the neural network. In the future, the generalisation features of the neural network can be verified by further experiments, where one half of the data is used to evolve the network and the other half is used for verification.

Additionally, the approach can be extended by using one neural network for different feet sizes, using the foot size as additional network input. The same might be possible for different soil properties like density and granularity.

The focus of the following work is to measure the shear forces of a foot while moving the foot through the soil. The shear forces can be approximated by a second evolved neural network. The network in combination with the result of this paper should represent the complete foot-soil contact mechanics needed to predict the behaviour of a walking robot on loose soil similar to the soil used for the creation of the networks.

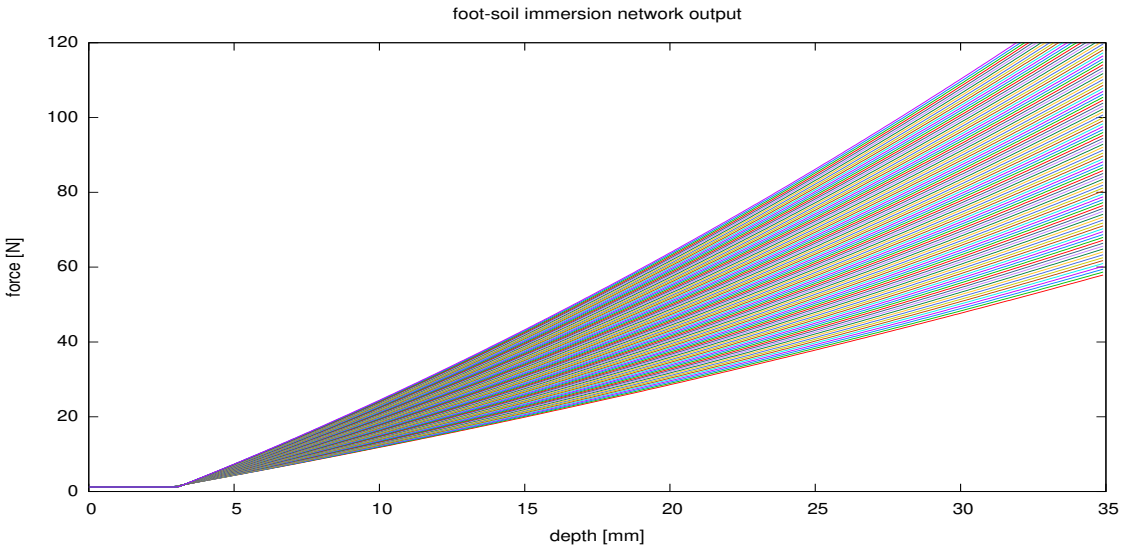



**Figure 9.** Resulting neural network output with the variance parameter in the range of one to tree in 100 steps. Thus, each curve depicts the network output of a corresponding variance value, whereby the lowest curve is plotted with a variance value of 1.0 and the highest curve with a variance value of 3.0.

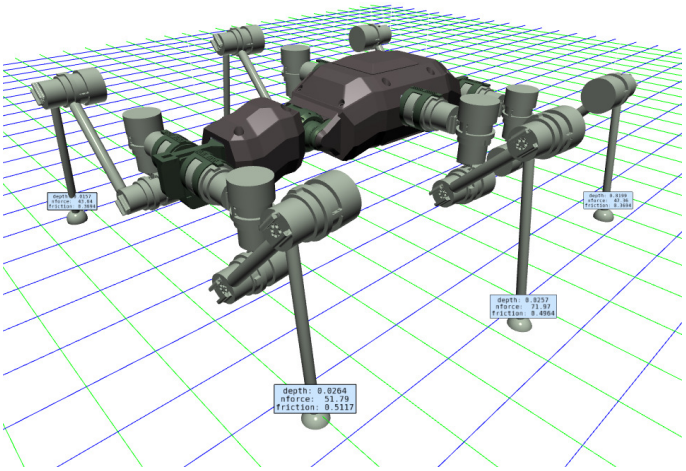


**Figure 11.** The SpaceClimber robot in the simulation with the new developed foot-soil interaction.

## ACKNOWLEDGEMENTS

The presented work is sponsored by the German Aerospace Center through the Virtual Crater [**?**] project (DLR: no. 50RA0903) and the SpaceClimber [13] project (DLR: no. 50RA0705) and the European Space Agency (ESA: contract no. 18116/04/NL/PA).

We thank all team members of the Virtual Crater and SpaceClimber projects at the German Research Center for Artificial Intelligence (DFKI) for their expertise and guidance, which were essential to this study.

## REFERENCES

[1] P. Bentley and S. Kumar, 'Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem', in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, volume 1, pp. 35–43, Orlando, Florida, USA, (July 1999).

[2] J. C. Bongard and R. Pfeifer, 'Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny', in *Proceedings of the Genetic and Evolutionary Computation Conference, (GECCO-2001)*, pp. 829–836, (2001).

[3] D. B. D'Ambrosio and K. O. Stanley, 'A novel generative encoding for exploiting neural network sensor and output geometry', in *Proceedings of the 9th Genetic and Evolutionary Computation Conference (GECCO-2007)*, pp. 974–981, (2007).

[4] F. Gruau, *Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm*, Ph.D. dissertation, Ecole Normale Superieure de Lyon, Laboratoire de l'Informatique du Parallelisme, France, January 1994.

[5] C. Igel, 'Neuroevolution for reinforcement learning using evolution strategies', in *Congress on Evolutionary Computation (CEC2003)*, eds., R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, volume 4, 2588–2595, IEEE Press, (2003).

[6] Hirzinger G Krenn R., 'Simulation of rover locomotion on sandy terrain - modeling, verification and validation', *ASTRA 2008*, (Noordwijk, The Netherlands (November 2008)).

[7] Ellery A. Patel N. and Scott G., 'Application of bekker theory to wheeled, tracked and legged vehicles', *SPACE 2004*, (San Diego, California, USA. (September 2004)).

[8] M. Römemrmann, S. Bartsch, and S. Haase, 'Validation of simulation-based morphology design of a six-legged walking robot', in *13th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines, 2010, 31 August - 03 September, Nagoya, Japan*, (2010).

[9] M. Römmermann, D. Kühn, and F. Kirchner, 'Robot design for space missions using evolutionary computation', in *IEEE Congress on Evolutionary Computation (IEEE CEC 2009)*, (2009).

[10] D. Rubinstein and R. Hitron, 'A detailed multi-body model for dynamic simulation of off-road tracked vehicles', *Journal of Terramechanics*, **41**(2-3), 163–173, (2004). 14th International Conference of the ISTVS.

[11] K. Sims, 'Evolving virtual creatures', *Computer Graphics (SIGGRAPH Proceedings)*, 15–22, (1994).

[12] R. Smith. Open dynamics engine, www.ode.org, 2005.

[13] SpaceClimber Project. http://robotik.dfki-bremen.de/en/research/projects/space-robotics/spaceclimber.html.

[14] A. Wieland, 'Evolving controls for unstable systems', in *Proceedings of the International Joint Conference on Neural Networks*, pp. 667–673, (1991).

4th International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2011)
Berlin, Germany, December 9 2011

# A Model of Head Direction Cells with Changing Preferred Head Direction

**Theocharis Kyriacou** and **John Butcher** and **Charles Day** [1]

**Abstract.**
A biologically inspired model of head direction cells is presented and tested on a small mobile robot. Head direction cells (discovered in the brain of rats in 1984) encode the head orientation of their host irrespective of the host's location in the environment. The head direction system thus acts as a biological compass (though not a magnetic one) for its host. Head direction cells are influenced in different ways by idiothetic (host-centred) and allothetic (not host-centred) cues. The model presented here uses the visual, vestibular and kinesthetic inputs that are simulated by robot sensors. The model is tested under different input conditions whereby the three inputs are in agreement or in disagreement. In particular, the case when the environment is rotated about the robot while the robot is stationary is considered. According to biological observations the preferred head direction of head direction cells changes in proportion to the environment rotation during this condition. A mechanism is proposed to replicate these biological observations.

## 1 Introduction: Biologically Inspired Robot Navigation

Biologically inspired navigation methods can perhaps be put in two broad categories. In the first, methods draw only from observations of animal behaviour without considering the underlying cognitive mechanisms that play a part in navigation. Tolman in 1946 (see [23]) was among the first who conducted experiments with rats that allowed such observations, but more recent work using rodents and insects is presented for example in [2] and [25].

In contrast, "bottom-up" approaches to bio-inspired models of navigation make use of knowledge obtained by observing the brain activity of animals while they perform navigational tasks (see for example [14]). During such experiments, the activity of a few brain cells can be recorded by means of microelectrodes. This gives some clues as to how a navigational mechanism is implemented in the brain. Bio-inspired models in this category make quite a lot of extrapolations that try to fill in the gaps.

Three types of brain cells called *place cells* (see [16]), *head direction cells* (see [21]) and *grid cells* (see [9]) have been discovered (mostly from experiments conducted on rats) and are thought to play a significant role in animal navigation. The work in this paper concentrates on head direction cells (or HD cells) and presents a model of these cells that is inspired by biological observations. A more detailed description of HD cells and an overview of previous work in modelling them is presented below.

[1] All authors are affiliated with the Research Institute for the Environment, Physical Sciences and Applied Mathematics (EPSAM), Keele University, Staffordshire, United Kingdom.

### 1.1 Head Direction Cells

The most recent comprehensive review of neurophysiological observations related to HD cells is found in [26]. Here below, the main characteristics of the HD system are outlined.

Head direction cells were first discovered in 1984 by Ranck Jr. and more detailed findings on them were published in 1990 by Taube and colleagues (see [21]). An HD cell fires maximally when the animal's head points in a particular direction. This is called the *preferred head direction* of the particular cell. The HD system includes a population of HD cells with preferred head directions distributed through $360^\circ$. The activity of an HD cell does not depend on the location of the animal in the environment. The head direction cell system can thus be thought of as being a biological head compass (though not a magnetic one) that is influenced by several senses. When an animal is placed in a new environment the preferred head direction of each cell in the HD system quickly settles to an arbitrary value. The system maintains this alignment for the specific environment even if the animal is removed and re-introduced back to the same environment. This alignment will only be reset (i.e. the environment will be treated as a new, previously unseen one) if several weeks have passed before the animal is re-introduced in the environment ([21]). The visual sense is the major input that helps to align the HD system when the animal is introduced in a previously visited environment. Strong visual cues (for example large and prominent landmarks) can influence the preferred head direction of HD cells (see experiments and observations in [15] and [21]). Another major input to the HD system is the vestibular sense. This allows the animal to maintain a correct head direction for some time after visual input is removed (by switching off the lights for example). Apart from the most important two inputs to the HD system mentioned above, other cues also play a part in influencing the preferred head direction of HD cells. These include olfactory cues (see [7]) and cues that are involved in self-locomotion (motor, kinesthetic and proprioceptive) (see [20] for a review). Cues to the HD system are classified in two categories: they can be *allothetic*, i.e. not self-centred (for example visual and olfactory) or *idiothetic*, i.e. self-centred (for example vestibular and kinesthetic). When two or more inputs to the HD system are in conflict the response of the system depends on several factors such as for example, the relative influence on the HD system, the conflicting perceptual modalities, the extent of the conflict and the magnitude of the inputs to the system (see example in [6]). Conflicts are extensively discussed in [26]. See also [20] for a more concise review.

### 1.2 Models of Head direction Cells

Several attempts have been made to model the HD system. Redish and colleagues in [17] and Goodridge and Touretzky in [8] present

anatomically faithful models but these only use the vestibular sense. In both cases the neural network weights are prescribed (i.e. not obtained by training). The models are however tested using real data obtained from experiments with rats. Models using both visual and vestibular inputs are presented in [18], [19] and [27]. In all three cases the model weights are also prescribed and tests are carried out in simulated environments. Also, only in [27] were input conflict situations simulated and compared with biological observations. Only a few examples exist in the literature of HD system models applied to real robotic agents. Of these, the most notable are presented in [1] and [5]. Both models incorporate visual and vestibular inputs but again, in both cases the models' weights are prescribed.

A model of the head direction system is presented by one of the authors in [10]. The model is original (compared to previous work) in that it uses three inputs (the visual, vestibular and kinesthetic senses) and in that it is trained and tested using (noisy) data obtained from a real robot. The author also considers different input situations and makes qualitative comparisons with biological observations.

In [3] a series of experiments were conducted, using rats in a simplified cylindrical environment, in order to determine the relative influence of the visual and vestibular senses to HD cells. In one of these experiments it was observed that the preferred head direction of HD cells remained mostly fixed when the environment walls were rotated. In other words, the rats maintained their original orientation (in the world reference frame) despite visual cues suggesting otherwise. Human experience suggests that a similar effect must be happening in the human brain. For example, when the environment around us moves while we are aware (from idiothetic cues) that we are not moving, despite the unease that this conflict causes, we are able to tell that it is indeed the environment that is moving and not us.

The model presented in this paper is novel in that it incorporates a mechanism that enables the modelled HD system to respond to the above described condition in the same way as biological experiments suggest. Furthermore, central to the proposed mechanism is a biologically plausible structure (the Continuous Attractor Neural Network or CANN - explained in section 2). To the best knowledge of the authors no previous attempt has been made to incorporate this feature in a model of the HD system.

Details of the model are given in two parts. First the core model is presented and tested in sections 2 to 5. The extended model is presented and tested in sections 6 and 7 respectively. A discussion and concluding remarks are presented in section 8.

## 2 The Core Model

The HD system model presented here is using a continuous attractor neural network (see introduction by Trappenberg in [4]). A continuous attractor neural network (CANN) is a network of interconnected nodes. In a fully connected network each node is connected via weighted connections to every other node including itself. A CANN is thus a form of recurrent network. The operation of the network is such that nodes in close association excite each other (via excitatory connections). The amount of excitation being proportional to the degree of association between the nodes. On the other hand, nodes that are less associated with each other are connected with inhibitory connections. The connection weights from one node to all other nodes are often prescribed by a Gaussian function with its tails below zero (the negative weight values for the inhibitory connections). A CANN gives rise to a self-sustained "hill" of excitation (the attractor) in the network. If the network is perfectly symmetrical about each node

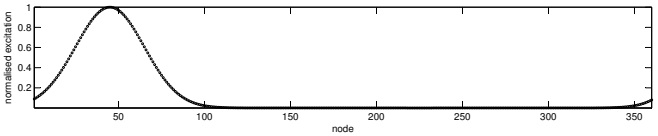



**Figure 1.** The state ($r^{\mathrm{HD}}$) of the HD system when it points at $45^\circ$.

(both in connectivity and weight values to other nodes) the attractor will be stationary when the network has no external influences. External input stimuli that temporarily distort the network's symmetry (by biassing the activation of nodes) can cause the attractor to move.

In the case of the HD system model presented here the CANN network is comprised of 360 nodes (the HD cells) and it is fully connected. Each node is associated with a preferred head direction and it is therefore most active when the subject (the host of the HD system) is facing in that direction. Conceptually the nodes can be considered as being arranged in a circle in order of preferred head direction[2]. Figure 1 shows the state of the HD system ($r^{\mathrm{HD}}$) when it points to $45^\circ$ (from an arbitrary reference direction).

The state (i.e. the excitation of each node) of the network implementation presented here is a function of the previous state of the system and three inputs: visual, vestibular and the kinesthetic. Figure 2 is a partial diagram that shows how these inputs are connected to the HD cells. The reader is referred to the caption of the figure for a detailed explanation.

The activation $h_i^{\mathrm{HD}}$ at time $t$ of a head direction cell $i$ in the model can be determined using the following differential equation:

$$\begin{aligned}\tau\frac{\mathrm{d}h_i^{\mathrm{HD}}(t)}{\mathrm{d}t} &= -h_i^{\mathrm{HD}}(t) \\ &\quad +\theta^{\mathrm{RC}}\sum_j (w_{ij}^{\mathrm{RC}} - w^{\mathrm{INH}})r_j^{\mathrm{HD}}(t) \\ &\quad +\theta^{\mathrm{VIS}} I_i^{\mathrm{VIS}}(t) \\ &\quad +\theta^{\mathrm{VES}}\sum_{jk} w_{ijk}^{\mathrm{VES}} r_j^{\mathrm{HD}}(t) r_k^{\mathrm{VES}}(t) \\ &\quad +\theta^{\mathrm{KIN}}\sum_{jk} w_{ijk}^{\mathrm{KIN}} r_j^{\mathrm{HD}}(t) r_k^{\mathrm{KIN}}(t) \end{aligned} \quad (1)$$

where $r_i^{\mathrm{HD}}(t)$ is the firing rate (excitation) of HD cell $i$ given by the sigmoid function:

$$r_i^{\mathrm{HD}}(t) = \frac{1}{1+e^{-2\beta h_i^{\mathrm{HD}}(t)}} \quad (2)$$

$\tau$ is the time constant of the system, $w_{ij}^{\mathrm{RC}}$ is the recurrent connection weight from HD cell $j$ to HD cell $i$, $I_i^{\mathrm{VIS}}$ is the visual input to HD cell $i$, $r_k^{\mathrm{VES}}$ is the firing rate of vestibular sensor cell $k$, $w_{ijk}^{\mathrm{VES}}$ is the weight value of the connection from HD cell $j$ to vestibular TMHD cell $i$ that is associated with the vestibular sensor cell $k$. Similarly, $r_k^{\mathrm{KIN}}$ is the firing rate of kinesthetic sensor cell $k$ and $w_{ijk}^{\mathrm{KIN}}$ is the weight value of the connection from HD cell $j$ to kinesthetic TMHD cell $i$ that is associated with the kinesthetic sensor cell $k$. The factors $\theta^{\mathrm{RC}}$, $\theta^{\mathrm{VIS}}$, $\theta^{\mathrm{VES}}$ and $\theta^{\mathrm{KIN}}$ control the influence of the

[2] This arrangement however is not necessary. In fact, in the brain, HD cells have not been found to be arranged in any particular order that relates to their preferred head direction.

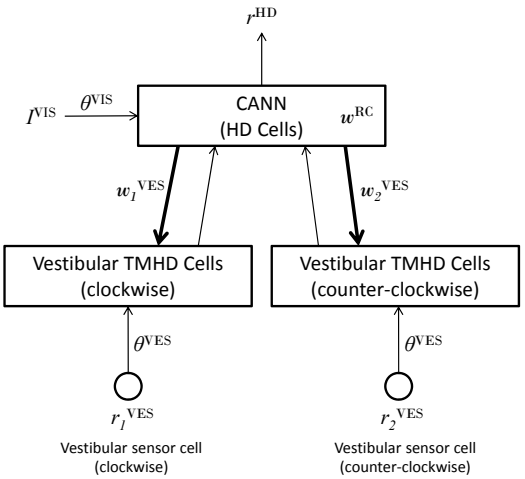



**Figure 2.** A partial diagram of the network of the HD model showing only the visual and vestibular inputs. The CANN (HD cells) is a fully connected network of 360 nodes with connection weights $w^{RC}$. The excitation $r^{HD}$ of the HD layer nodes is the output of the system and it indicates the direction in which the HD system is pointing at a given time. The visual input $I^{VIS}$ is a 360-element vector that is directly (one-to-one) connected to the HD cells. Two groups of cells (for clockwise and anti-clockwise vestibular input) called Turn-Modulated Head Direction cells (TMHD - see [26], chapter 18) are connected to the HD cells in such a way so as to distort the symmetry of the CANN whenever there is vestibular input and thus cause the CANN attractor to move. One TMHD group of cells receives input from a single cell (called the *vestibular sensor cell*) that simulates the clockwise vestibular sense. Similarly the other TMHD group of cells receives input from another vestibular sensor cell that simulates the anti-clockwise vestibular sense. In a similar fashion there exist another two groups of TMHD cells (not shown for clarity) that are associated with the kinesthetic sense. These two groups are driven by two cells respectively that simulate the clockwise and anti-clockwise kinesthetic senses. TMHD cells are implemented here using sigma-pi neurons. These are neurons that compute the sum of products of their inputs (see [12]). Connections shown with bold lines are trained whereas those with thin lines are not.

recurrent, visual, vestibular and kinesthetic inputs respectively to the HD cells. Finally, $w^{INH}$ is a constant negative offset to the recurrent connection weights that serves as a quick way to make the connection weights between distant cells negative (inhibitory). The weights $w^{RC}$, $w^{VES}$ and $w^{KIN}$ can be prescribed using Gaussian-like functions like for example in [18], [19] and [27] but this method is not biologically plausible and conveniently ignores the effects of noise to the weight values. Here, training data is collected using a real robot and applied to the model's weights using Hebbian learning as explained below.

## 3 Experimental Setup

The model described above was trained and tested using a LEGO® robot with an on-board omnidirectional video camera (see figure 3).

The robot is further equipped with a gyroscopic sensor and an acceleration sensor. For locomotion, the robot uses two active wheels and a dummy castor. For the purposes of the work presented here, the visual input to the HD model was provided by processing the image from the omnidirectional video camera on the robot (see figure 4 for an example of a video image). The vestibular input was provided directly by the gyro sensor. One vestibular sensor cell (see figure 2) was driven by the raw gyro signal and the other by the inverted version of the gyro signal. The two kinesthetic inputs were provided by differentiating the signals from the odometric sensors in the motors driving each wheel of the robot. The robot was controlled by a PC via USB connection in order to achieve maximum possible data transfer rates when reading the robot's sensors. The video during each recording session was independently recorded on the video camera and during post-processing it was time-corresponded with the robot's data. This setup allowed a 10Hz sampling rate in all data sources (gyro, motor position, video). Two sets of data were collected using the above setup. The first (training set) was used to train the network weights of the HD model and the second (the test set) was used to test the model.

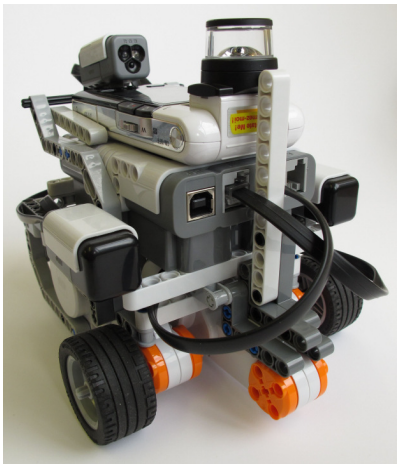


**Figure 3.** The LEGO® MINDSTORMS® NXT robot used for the experiments presented here. The robot is equipped with an on-board omnidirectional video camera (above the NXT brick) a gyroscopic sensor and an acceleration sensor (pictured to the left and right above the wheels). For locomotion, the robot uses two active wheels (seen at the front) and a dummy castor (not visible in the picture).

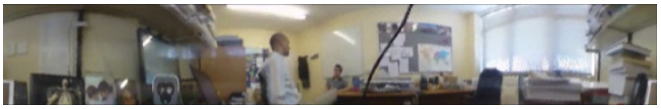


**Figure 4.** Snapshot from the omni directional video camera on-board the robot.

## 4 Training the Model

The duration of the training set was 873 seconds (14.55 minutes). During this time the robot was programmed to continuously rotate in a random direction with a constant rotational speed of approximately 35 degrees/second. In order to obtain the expected output of the system at time $t_n$ (i.e. training pattern $p_n$), initially, the direction of the robot (based on a world reference frame) was extracted from the video data by finding the maximum correlation (along the abscissa of the video image) between the video frame taken at $t_n$ and the first captured video frame taken at $t_0$. As the robot was only rotating on the spot during the experiments described here, this provided a convenient way of establishing the world-based orientation of the robot. After that, $p_n$ was created by translating a Gaussian function (of standard deviation $\sigma$) so that it would be centred at the the robot's orientation at $t_n$.

Training of the recurrent weights $w^{\mathrm{RC}}$ was achieved using the Hebbian learning rule:

$$\delta w_{ij}^{\mathrm{RC}} = k^{\mathrm{RC}} r_i^{\mathrm{HD}} r_j^{\mathrm{HD}} \tag{3}$$

where $k^{\mathrm{RC}}$ is the learning rate. Training of the vestibular weights was achieved using the following rule:

$$\delta w_{ijk}^{\mathrm{VES}} = k^{\mathrm{VES}} r_i^{\mathrm{HD}} \bar{r}_j^{\mathrm{HD}} r_k^{\mathrm{VES}} \tag{4}$$

where $\bar{r}^{\mathrm{HD}}$ is a historical trace value of $r^{\mathrm{HD}}$ and is given by:

$$\bar{r}_j^{\mathrm{HD}}(t+\delta t) = (1-\eta) r^{\mathrm{HD}}(t+\delta t) + \eta \bar{r}_j^{\mathrm{HD}}(t) \tag{5}$$

A similar rule to the one given by equation 4 was also used in order to train the kinesthetic weights $w_{ijk}^{\mathrm{KIN}}$.

Note that equation 4 (through the use of equation 5) considers both current and past values of the state of the HD model. It is this feature that gives rise to idiothetic weight profiles that bias the CANN attractor to move with the right speed and in the right direction according to the idiothetic inputs. In equation 5, $\eta$ is a parameter that dictates the influence of the current and previous state of the network in the trace rule (equation 4) and $\delta t$ is the time delay between the two states considered.

Table 1 lists the parameters of the HD model presented here. Recall that the $\theta$ parameters control the influence of the inputs to the HD model. These parameters are not orthogonal to each other. In most previous work, these parameters (or parameters with similar role) have been obtained by trial-and-error. Here, $\theta^{\mathrm{RC}}$, $\theta^{\mathrm{VIS}}$, $\theta^{\mathrm{VES}}$ and $\theta^{\mathrm{KIN}}$ were obtained using an evolutionary strategy that drew from biological observations (see [11]). The parameters $\theta^{\mathrm{RC}\varphi}$ and $\theta^{\mathrm{TM}\varphi}$ are explained in section 6.

**Table 1.** The parameter values used in the HD model presented.

| | | | |
|---|---|---|---|
| $\tau$ | 0.1s | | |
| $\delta t$ | 0.1s | $\theta^{\mathrm{RC}}$ | 3.7 |
| $\beta$ | 0.1 | $\theta^{\mathrm{VIS}}$ | 19.6 |
| $k^{\mathrm{RC}}$ | 0.01 | $\theta^{\mathrm{VES}}$ | 0.0054 |
| $k^{\mathrm{VES}}$ | 0.01 | $\theta^{\mathrm{KIN}}$ | 0.0015 |
| $k^{\mathrm{KIN}}$ | 0.01 | | |
| $\eta$ | 0.9 | $\theta^{\mathrm{RC}\varphi}$ | 5.5 |
| $\sigma$ | 20° | $\theta^{\mathrm{TM}\varphi}$ | 0.060 |
| $w^{\mathrm{INH}}$ | 0.5 | | |

## 5 Testing the Core Model

Test data was collected for 102.9 seconds (1.715 minutes). During this session the robot was programmed to continuously rotate on the spot and in the same direction with a constant rotational speed of approximately 35 degrees/second. The test data is shown in figure 5.

The same test data set was used for all test cases described in this paper. However, without loss of integrity, the test data was manipulated for each test case in order to present the model with different input conditions. One or more of the following four manipulations was applied for a specified duration during a test:

1. Visual input intensity was set to 0 in order to simulate darkness.
2. The visual cue was "frozen" to a particular vector (taken from a particular instant in the actual data) in order to simulate a fixed visual input.

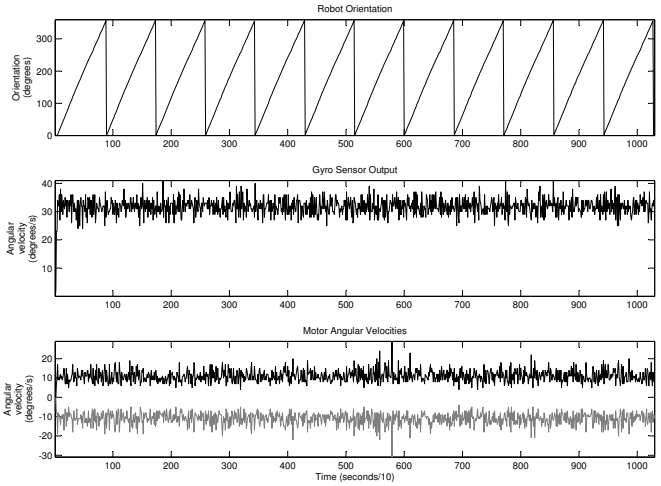



**Figure 5.** The test data sampled at 10Hz. **Plot 1:** The true orientation of the robot (extracted from the video sequence). **Plot 2:** The output of the gyro sensor on the robot used to simulate the vestibular sense. One vestibular sensor cell is fed with this signal and the other is fed with the inverted version of this signal. **Plot 3:** The left and right wheel velocities of the robot (grey and black lines respectively) that are used to simulate the kinesthetic sense. The two signals provide the inputs to each of the kinesthetic sensor cells respectively.

3. The gyro signal (vestibular input) was set to 0 to simulate no vestibular input.
4. The wheel velocities (kinesthetic input) were set to 0 to simulate no kinesthetic input.

The HD model was tested under a number of different input conditions that were created by applying the above manipulations. The test run presented in figure 6 summarises these input conditions.

For this test run the visual input was turned off for $45s < t < 90s$ (see plot 3 of figure 6). Also, for $0s < t < 15s$ and $75s < t < 90s$ the visual cue to the model was kept fixed. This can be seen in plot 2 of figure 6. The vestibular input was set to 0 for $0s < t < 15s$ and $t > 75s$ (plot 4 of figure 6) and the kinesthetic input was set to 0 for $0s < t < 15s$, $30s < t < 45s$ and $t > 60s$ (plot 5 of figure 6). Plot 2 in figure 6 compares the actual head direction (obtained from video as described earlier) and the model-predicted head direction. In effect, this test run provides the system with seven conditions that last 15 seconds each (except the last one which lasts for 13 seconds):

1. $0s < t < 15s$: The lights are on and the robot is neither moving under its own volition nor it is being moved by any external force.
2. $15s < t < 30s$: The robot moves under its own volition (vestibular and kinesthetic input) while the lights are on.
3. $30s < t < 45s$: The robot is being moved by an external force (only vestibular input) while the lights are on.
4. $45s < t < 60s$: The robot is rotating under its own volition (vestibular and kinesthetic input) while the lights are off.
5. $60s < t < 75s$: The robot is being moved by an external force (only vestibular input) while the lights are off.
6. $75s < t < 90s$: The robot is stationary while the lights are off.
7. $t > 90s$: The lights are on and the robot is stationary but the environment is being moved about the robot.

While the lights are on for $0s < t < 45s$ the output of the HD model follows the visual cue regardless of the state of the idiothetic

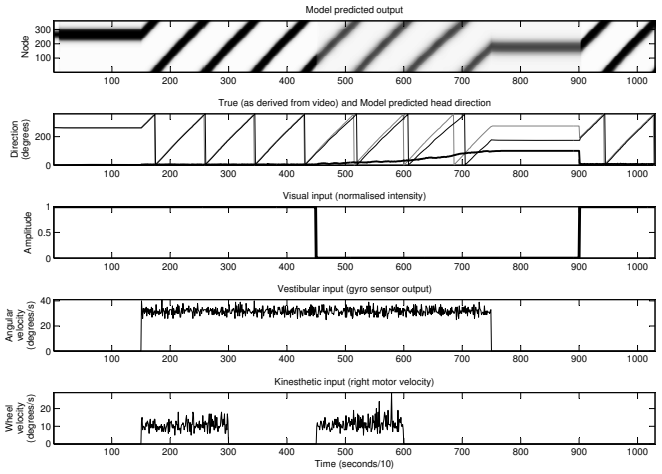



**Figure 6.** Testing the core model. **Plot 1:** The model output. This is a surface plot viewed from the top. The black colour value is proportional to the excitation of each HD cell in the CANN. **Plot 2:** The robot's true head direction (grey line) and the model-predicted head direction (black line) with the absolute difference between them (wide black line). Note that the model-predicted head direction is the crest of the surface in plot 1. **Plot 3:** The normalised intensity of the visual input. When this is 0 the robot is in the dark (no visual input). **Plot 4:** The vestibular input (gyro sensor signal). **Plot 5:** The kinesthetic input (right motor velocity). Note that only the right motor velocity is plotted for clarity.

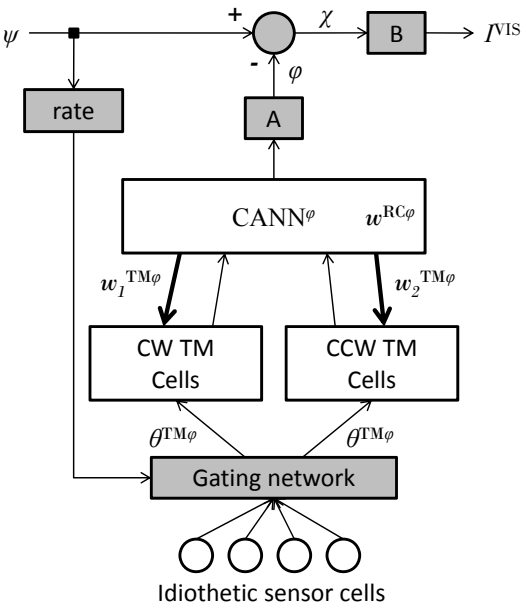



**Figure 7.** The extension to the core model (shown in 2) of the HD system. $\mathrm{CANN}^{\varphi}$ operates as an integrator of $\varphi$. The attractor (the position of which represents angle $\varphi$) is shifted using two groups of turn-modulated cells, one for clockwise shifts and another for counter-clockwise shifts (CW TM and CCW TM respectively). The two groups of TM cells are driven respectively by two signals (weighted by $\theta^{\mathrm{TM}\varphi}$) that carry the rate of change of $\psi$ in the respective direction. The signals are produced by a gating network that is only active (i.e. gate open) when the vestibular and kinesthetic sensor cells are not firing. Block A takes the 360-component output of the attractor and computes angle $\varphi$ (signified by the node with the highest firing rate) so that it can be subtracted from $\psi$. $\chi$ is then converted by block B into the visual input $I^{\mathrm{VIS}}$ (a 360-component vector) for the CANN containing the HD cells as explained in section 4. The shaded blocks can all be realised by neural networks.

inputs. When there is no visual input (lights off) for $45s < t < 90s$ the system integrates the vestibular and kinesthetic inputs to keep track of its direction. However, because these inputs are idiothetic and there is no world-based reference to use in order to relate to the true orientation of the robot the output of the system drifts away from the true orientation of the robot. The drift is greater when only the vestibular input is present ($60s < t < 75s$) than the case when both vestibular and kinesthetic inputs corroborate ($45s < t < 60s$). When no input is present ($75s < t < 90s$) the system maintains its orientation (the CANN attractor is self sustained and stationary). Since the above conditions succeed each other it can be seen that during the last mentioned condition (i.e. for $75s < t < 90s$) the robot is constantly in error since the lights remained off from the previous condition and it has therefore not been allowed to "realise" that it has drifted from the true orientation. This error is instantaneously corrected when the lights come on at $t = 90s$.

For $t > 90s$, the core implementation of the HD system presented thus far, causes the robot to "believe" that it is rotating when in fact it is the environment that is being rotated around it. Experiments with rats (see [3]) however show that under such a condition rats maintain their orientation, in effect "realising" that the environment is being moved around them. In order to address this problem, the model described this far is extended as described in the sections that follow.

## 6 The Extended Model

Let angle $\chi$ be the orientation of the robot in a world reference frame and angle $\psi$ be the orientation of the robot with respect to its environment (i.e. the angle that the robot calculates from its visual input signal). In the test run above $\chi$ and $\psi$ were the same except for $t > 90s$ when an offset between the two angles was introduced due to the environment having been rotated. According to biological observation (see [3]), for $t > 90s$ in the example above, HD cells should maintain their excitation relative to $\chi$ when what actually happens above is that the HD model maintains angle $\psi$ as its reference.

In order to correct for this shift in the reference orientation the core HD model needed to be extended so that, while there is no idiothetic input and the visual cue is moving, the difference between $\chi$ and $\psi$ is integrated into an angle that shall be called $\varphi$ here. Of course the robot has no access to $\chi$ directly since its allothetic cues (visual input) relate to $\psi$ but since the robot is not moving (no idiothetic inputs) during this input condition there is no change in $\chi$ and therefore the difference between $\chi$ and $\psi$ is in fact the change in $\psi$. In this model $\varphi$ is always subtracted from $\psi$ (the angle extracted from the visual input) thus effectively always maintaining an orientation with reference to $\chi$ (the world-based reference frame).

The mechanism that is used here to achieve this can be thought of as an integrator of angle $\varphi$ that operates only when the condition *no idiothetic input and changing visual cue* is met. This is achieved here by another CANN (the integrator) the output of which is a representation of the angle $\varphi$. The position of the attractor in this CANN (that we shall call $\mathrm{CANN}^{\varphi}$) is changed by the *rate of change* of angle $\psi$ when the vestibular and kinesthetic sensor cells are not active. Figure 7 illustrates this mechanism. The reader is referred to the caption of the figure for a detailed explanation.

Apart from the CANN which is the main feature of the diagram in figure 7 the operations of the other network blocks (shown shaded

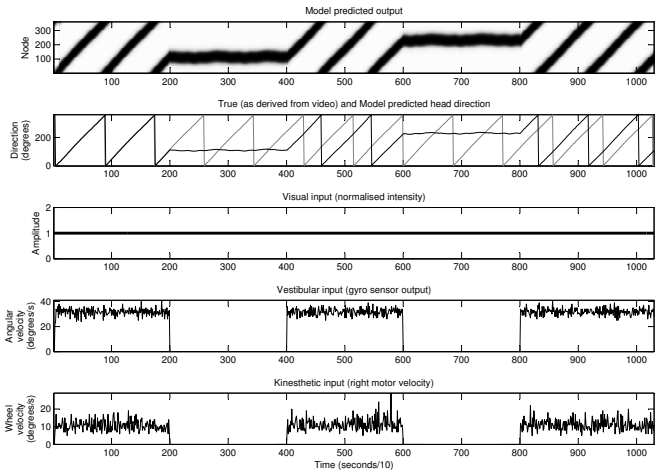



**Figure 8.** Testing the extended model. **Plot 1:** The model output (excitation of HD cells). **Plot 2:** The robot's head direction (grey line) and the model-predicted head direction (black line). **Plot 3:** The normalised intensity of the visual input. **Plot 4:** The vestibular input (gyro sensor signal). **Plot 5:** The kinesthetic input (right motor velocity only).

in the figure) can also be easily realised with neural networks (for example multi-layer perceptron networks and recurrent networks). Detailed expansion of these blocks however is omitted here as it falls out of the main scope of this paper.

This extension to the model introduces the two new parameters $\theta^{\mathrm{RC}\varphi}$ and $\theta^{\mathrm{TM}\varphi}$ that describe the contribution of the relevant inputs to the $\mathrm{CANN}^{\varphi}$. The values of these parameters for the extended implementation described here are given in table 1.

## 7 Testing the Extended Model

In order to test the extended model a new test run was created from the test data set. During this test the visual cue was always on (lights on) and rotating but both idiothetic inputs were made to alternate every 20 seconds between 0 and their original value. In effect, during this test run, conditions alternate between: (a) the robot rotating while the environment is fixed and (b) the robot being stationary while the environment is rotated. Figure 8 shows the response of the system during this test run.

Note how the system now correctly reflects the biological observations made in [3] when the environment is rotated about the robot. In other words, during this time the robot "believes" its orientation is not changing with respect to the world reference frame, even though the robot does not have direct access to this frame of reference.

## 8 Discussion and Conclusion

A biologically inspired model of the head direction system was presented and implemented on a small mobile robot. The model takes three inputs (visual, vestibular and kinesthetic) that are among those that most influence the biological HD system [20]. The model was trained and tested using real data obtained from the robot.

The main contribution of the work presented here is a biologically plausible mechanism that simulates the change in the preferred head direction of HD cells when visual cues suggest rotation of the agent when idiothetic cues do not. No other example has been found in the literature that models this feature of the HD system.

At the present moment it is impossible to observe the real head direction system of an animal in its entirety and therefore we can only speculate about how the whole system works. Like the one presented here, all HD system models so far draw on behavioural observations of animals and in more recent years from microelectrode signal recordings of a small number of HD cells. We fully realise that the entire biological system is much more complex. However, the model proposed here qualitatively replicates some of the biological observations mentioned in the literature. It is by no means claimed here that the presented model is anatomically accurate. This is not the primary intention of the authors.

The work presented here contributes towards a model of biologically inspired robot navigation. Probabilistic methods for robot navigation, that currently prevail in the literature (see [22] for a review), leave quite a few fundamental problems unsolved. These include their dependency on accurate maps, their inability to deal with change in the environment and their dependency on accurate sensors (see [13]). Humans and animals on the other hand are able to deal with these problems seamlessly. The long-term aim of this work is to understand and then simulate this navigational ability of biological organisms in artificial agents such as autonomous robots.

According to Trullier and colleagues (in [24]) biological navigation methods may not always produce the best, most mathematically optimal solution to a navigation problem but they are fast, flexible and adaptive. What is best and most mathematically optimal however depends on the employer of a particular navigational skill. We know little about even the simplest of organisms in nature and it could therefore really be that for a particular organism, their navigation strategy is the best in all respects. Modelling biological mechanisms of navigation helps us understand better the remarkably complex systems in nature. Besides the information value however, the understanding of how these mechanisms evolved, rather than just what they do and how they do it, may lead us to more generalised principles of designing artificial navigation systems that might be the best and most optimal for their intended application.

## REFERENCES

[1] A. Arleo and W. Gerstner, 'Spatial orientation in navigating agents: Modeling head-direction cells', *Neurocomputing*, **38-40**(1-4), 1059–1065, (2001).
[2] R. Biegler and R.G.M. Morris, 'Landmark stability is a prerequisite for spatial but not discrimination learning', *Nature*, **361**, 631–633, (February 1993).
[3] Hugh T. Blair and Patricia E. Sharp, 'Visual and vestibular influences on head-direction cells in the anterior thalamus of the rat', *Behavioral Neuroscience*, **110**(4), 643–660, (1996).
[4] *Recent Developments in Biologically Inspired Computing*, eds., Leandro N. de Castro and Fernando J. Von Zuben, Idea Group Publishing, March 2005.
[5] Thomas Degris, Loïc Lachèze, Christian Boucheny, and Angelo Arleo, 'A spiking neuron model of head-direction cells for robot orientation', in *In Proceedings of the Eighth International Conference on the Simulation of Adaptive Behavior, from Animals to Animats*, pp. 255–263. MIT Press, (2004).
[6] Ariane S. Etienne, Roland Maurer, and Valérie Séguinot, 'Path integration in mammals and its interaction with visual landmarks', *Journal of Experimental Biology*, **199**, 201–209, (1996).
[7] Jeremy P. Goodridge, Paul A. Dudchenko, Kimberly A. Worboys, Edward J. Golob, and Jeffrey S. Taube, 'Cue control and head direction cells', *Behavioral Neuroscience*, **112**(4), 749–761, (1998).
[8] Jeremy P. Goodridge and David S. Touretzky, 'Modeling attractor deformation in the rodent head-direction system', *Journal of Neurophysiology*, **83**, 3402–3410, (2000).
[9] Torkel Hafting, Marianne Fyhn, Sturla Molden, May-Britt B. Moser,

and Edvard I. Moser, 'Microstructure of a spatial map in the entorhinal cortex', *Nature*, **436**(7052), 801–806, (August 2005).

[10] Theocharis Kyriacou, 'An implementation of a biologically inspired model of head direction cells on a robot', in *Towards Autonomous RObotic Systems (TAROS) 2011*, (2011).

[11] Theocharis Kyriacou, 'Using an evolutionary algorithm to determine the parameters of a biologically inspired model of head direction cells', *Journal of Computational Neuroscience*, 1–15, (2011).

[12] Bartlett W. Mel and Christof Koch, 'Sigma-pi learning: on radial basis functions and cortical associative learning', in *Advances in neural information processing systems 2*, ed., David S. Touretzky, 474–481, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, (1990).

[13] Michael Milford, *Robot Navigation from Nature - Simultaneous Localisation, Mapping, and Path Planning based on Hippocampal Models*, volume 41 of *Springer Tracts in Advanced Robotics*, Springer, 2008.

[14] R. G. Morris, P. Garrud, J. N. Rawlins, and J. O'Keefe, 'Place navigation impaired in rats with hippocampal lesions', *Nature*, **297**(5868), 681–683, (June 1982).

[15] R. U. Muller, J. L. Kubie, and J. B. Ranck, 'Spatial firing patterns of hippocampal complex-spike cells in a fixed environment', *Neuroscience*, **7**(7), 1935–1950, (1987).

[16] J. O'Keefe and J. Dostrovsky, 'The hippocampus as a spatial map. preliminary evidence from unit activity in the freely-moving rat', *Brain Research*, **34**(1), 171–175, (November 1971).

[17] A.D. Redish, A.N. Elga, and D.S. Touretzky, 'A coupled attractor model of the rodent head direction system', *Network: Computation in Neural Systems*, **7**(4), 671–685, (1996).

[18] W.E. Skaggs, J.J. Knierim, H.S. Kudrimoti, and B.L. McNaughton, 'A model of the neural basis of the rat's sense of direction', *Advances in Neural Information Processing Systems*, **7**, 173–80, (1995).

[19] S.M. Stringer, T.P. Trappenberg, E.T. Rolls, and I.E. de Araujo, 'Self-organizing continuous attractor networks and path integration: one-dimensional models of head direction cells', *Network: Computation in Neural Systems*, **13**(2), 217–242, (May 2002).

[20] J.S. Taube, 'Head direction cells and the neurophysiological basis for a sense of direction', *Progress Neurobiololy*, **55**(3), 225–256, (1998).

[21] J.S. Taube, R.U. Muller, and J.B. Ranck Jr., 'Head-direction cells recorded from the postsubiculum in freely moving rats. i. description and quantitative analysis', *Neuroscience*, **10**(2), 420–435, (1990).

[22] Sebastian Thrun, Wolfram Burgard, and Dieter Fox, *Probabilistic robotics*, Intelligent robotics and autonomous agents, MIT Press, September 2005.

[23] E.C. Tolman, B.F. Ritchie, and D. Kalish, 'Studies in spatial learning. i. orientation and the short-cut', *Journal of Experimental Psychology*, **36**, 13–24, (1946).

[24] O. Trullier, S. Wiener, A. Berthoz, and J. Meyer, 'Biologically-based artificial navigation systems: Review and prospects', *Progress in Neurobiology*, **51**, 483–544, (1997).

[25] R. Wehner and R. Menzel, 'Do insects have cognitive maps?', *Annual Review of Neuroscience*, **13**, 403–414, (March 1990).

[26] *Head direction cells and the neural mechanisms of spatial orientation*, eds., S. I. Wiener and J. S. Taube, MIT Press, 2005.

[27] P. Zeidman and J.A. Bullinaria, 'Neural models of head-direction cells', in *From Associations to Rules: Connectionist Models of Behavior and Cognition*, eds., R.M. French and E. Thomas, pp. 165–177, (2008).

4th International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2011)
Berlin, Germany, December 9 2011

# Using CMA-ES with Local Models for Difficult Optimisation Problems

**Nils T Siebel**[1] and **Sven Grünewald**[2]

**Abstract.** Evolutionary algorithms are a popular tool to find good solutions for complex optimisation problems. Self-tuning algorithms like CMA-ES ("Covariance Matrix Adaptation Evolution Strategy") are particularly useful for problems that are high-dimensional, ill-conditioned and/or non-separable. A problem with these methods, as with evolutionary algorithms in general, is the computational time needed to locate a solution in the search space. With large optimisation problems, the necessity to evaluate the objective function at a very large number of points poses a considerable obstacle. In this article we examine when the use of a local model of the objective function in CMA-ES can speed up the optimisation process.

## 1 INTRODUCTION

Given a function $f : \mathbb{R}^n \to \mathbb{R}$, the typical task for an optimisation algorithm is to find a solution $x_{\min}$ to the problem

$$x_{\min} \in \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} f(x), \tag{1}$$

or at least some $x^\star$ with $f(x^\star) \approx f(x_{\min})$ for an $x_{\min}$ as above. When there is no analytical solution available numerical optimisation algorithms can be used. These work by starting from a given (or random) starting point $x_0$ and iteratively searching for a solution by evaluating candidate solutions $x_1, x_2, \ldots$ sampled from the search space. For the types of problems considered here these evaluations of the function $f$ are computationally expensive which slows down the optimisation process.

In this article we consider a method which uses a local model (approximation) $\hat{f}$ of the objective function $f$ which is simpler and hence, faster to evaluate than $f$ itself. If $\hat{f}$ approximates $f$ sufficiently well then it can be used to identify candidate solutions where the actual function $f$ can then be evaluated. Such an $\hat{f}$ is often called a *meta model* of the function $f$.

The remainder of the article is organised as follows. Section 2 introduces the terminology and describes related work. The local model used in our experiments is described in Section 3 and validated by experiments in Section 4. Section 5 concludes the article.

## 2 PRELIMINARIES AND RELATED WORK

### 2.1 Evolutionary Algorithms and CMA-ES

Evolutionary algorithms are ways of solving optimisation problems that are inspired by Darwinian principles [4]. A set of candidate solutions ("*individuals*") is generated and maintained by the algorithm. Over time (time steps from one generation to the next) this *population* moves through the search space, looking for a good solution to the problem.

In order to generate a new generation from the current one, an evolutionary algorithm generates new individuals from the existing ones. This process involves a random element, usually probability distributions which model the way the population changes and moves through the space. In order to keep the algorithm as general as possible (and not implicitly assume certain characteristics about the problem), *evolution strategies* change the parameters of the probability distributions over time, adapting it to the given problem and structure of the search space.

While standard evolution strategies adapt these *strategy parameters* randomly *(ibid.)*, Hansen and Ostermeier have presented a method which uses derandomised self-adaptation [6]. Their method, *CMA-ES* ("Covariance Matrix Adaptation Evolution Strategy") maintains an estimate of the covariance matrix of the search distribution. This covariance matrix describes the pairwise dependencies between the components of variables in the solution space. New individuals are created by sampling from the normal distribution spanned by the main axes of the covariance matrix. The covariance matrix updated regularly by CMA-ES converges to the inverse Hessian matrix (up to a constant factor) *(ibid.)*. Therefore CMA-ES can use a locally valid 2nd order model of the structure of the search space without requiring the knowledge or even existence of the objective function's derivative. CMA-ES uses an adaptive step size and analyses the evolutionary path to avoid problems like premature convergence and is known for fast convergence to good solutions even with multi-modal and non-separable functions in high-dimensional spaces. It has also been successfully applied by Igel to reinforcement learning of neural network weights [7], which is known to be a difficult problem due to the curse of dimensionality [2] and numerical ill-conditioning [11].

We have used CMA-ES in our neuro-evolutionary method EANT2 ("Evolutionary Acquisition of Neural Topologies Version 2") [13] and are now looking into ways to speed up the optimisation process by using a local meta model of the objective function, which is very slow to evaluate.

### 2.2 Locally Weighted Regression

*Locally weighted regression* is an approach for avoiding function evaluations by using already known function values from the vicinity of the point to be evaluated [3, 1]. The main idea is to use a model $\hat{f}(x_i, \beta)$ for inter-/extrapolating given function values $\{(x_i, y_i)\}_{i=1,\ldots,m}$ so as to estimate the function value at a point $q$. The structure of the model is constant, usually a polynomial of fixed order. Its parameters $\beta$ are fitted to known function values by min-

[1] HTW University of Berlin, Germany, email: siebel@htw-berlin.de
[2] University of Kiel, Germany

imising an error measure $C(q)$.

The known data points can be weighted by applying a kernel function $K : \mathbb{R} \to [0, 1]$ in order to emphasise those points which are close to the query point $q$. An additional smoothing parameter $h$ can be used to model the density of data around the point $q$. This gives the following error measure:

$$C(q) = \sum_{i=1}^{m} \left[ \left( \hat{f}(x_i, \beta) - y_i \right)^2 K \left( \frac{d(q, x_i)}{h} \right) \right] \quad (2)$$

where $d(q, x_i)$ is the distance between $x_i$ and the point $q$.

### 2.3 CMA-ES with Local Meta Models

The idea to use a (meta) model of the objective function in evolutionary optimisation has been presented by Kern *et al.* [8]. In order to achieve a speedy optimisation many evaluations of the objective function are replaced by evaluations of the meta model and inter-/extrapolated by locally weighted regression. The authors report a significant speedup, e.g. a reduction from 5263 (objective) function evaluations, abbreviated *fevals*, to 626 fevals with the widely used non-separable double sum test function by Schwefel [12],

$$f_{\text{Schwefel}}(x) = \sum_{i=1}^{n} \left( \sum_{j=1}^{i} x_j \right)^2 \quad (3)$$

and dimension $n = 16$.

## 3 DETAILS OF THE METHOD

Our analysis of the method presented in [8] extends their experiments by looking at more difficult problems, in order to find out when CMA-ES with a local meta model can be used. We analyse the overall time (wall clock time) spent by the algorithm, in addition to the number of function evaluations ("fevals") and optimisation result (best function value found).

### 3.1 The Model

We use the following quadratic model:

$$\hat{f} = \beta^T \left( x_1^2, \ldots, x_n^2, x_1 x_2, \ldots, x_{n-1} x_n, x_1, \ldots, x_n, 1 \right)^T \quad (4)$$

which yields the model's dimensionality,

$$D_{\hat{f}} = \frac{n\,(n+3)}{2} + 1 \quad (5)$$

where $n$ is the problem dimension.

### 3.2 The Kernel

The kernel used is biquadratic:

$$K(d) = \begin{cases} \left(1 - d^2\right)^2 & \text{if } d < 1 \\ 0 & \text{else} \end{cases} \quad (6)$$

### 3.3 Bandwidth

The bandwidth $h$ is determined dynamically for each query point $q$ to involve the $k$ nearest neighbours of $q$ among the known data points. The optimum value has been empirically determined and set to

$$k = n\,(n+3) + 2. \quad (7)$$

### 3.4 Test Functions

Our first test function is the double sum Schwefel function $f_{\text{Schwefel}}$ from (3), which is continuous, convex and unimodal but creates difficulties for methods which rely on actual or estimated gradient directions due to its inherent rotation.

The second test function is the unimodal, non-separable Rosenbrock valley ("banana") function [10],

$$f_{\text{Rosenbrock}}(x) = 100 \cdot \sum_{i=1}^{n-1} \left( \left( x_i^2 - x_{i+1} \right)^2 + (x_i - 1)^2 \right), \quad (8)$$

which has its minimum inside a deep valley with the shape of a parabola. Due to the non-linearity of the valley many optimisation algorithms converge slowly because they keep changing the search direction.

The third test function is the following heavily multi-modal function by Rastrigin [9]:

$$f_{\text{Rastrigin}}(x) = 10 \cdot \left( n - \sum_{i=1}^{n} \cos\left(2\pi x_i\right) \right) + \sum_{i=1}^{n} x_i^2, \quad (9)$$

which is difficult to minimise due to its many local minima.

All three functions are plotted in two dimensions in Figure 1.

## 4 EXPERIMENTAL RESULTS

In our experiments we optimised each test function at least 5 times for a given problem dimension. The problem dimensions examined were $n = 2, 5, 10, 15, 20, 25$ and, for $f_{\text{Schwefel}}$, also $n = 30$.

### 4.1 Results for $f_{\text{Schwefel}}$

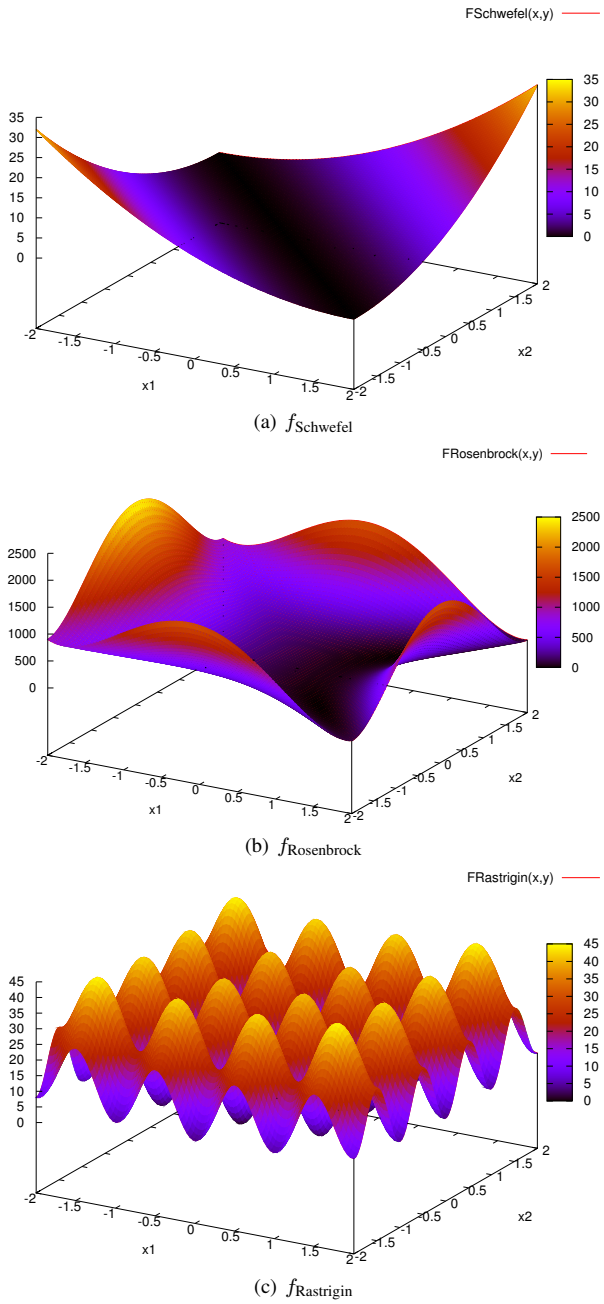
The results for the $f_{\text{Schwefel}}$ double sum function are given in Table 1. As in [8] the advantage of CMA-ES with a local meta model is particularly pronounced for this test function. For the dimension $n = 2$ CMA-ES with a local meta model on average only needs 80 instead of 513 fevals. With $n = 15$ the number can even be reduced from 5526 to 538. Our results support the results by Kern *et al.* and we can see that for higher dimensions (we tested up to $n = 30$) the same improvement can be achieved. With increasing dimensions the fevals increases more slowly with CMA-ES with a local meta model in comparison to the standard CMA-ES.

When considering the function values (see Figure 2, middle graph) one can, however, see that CMA-ES with the local meta model does not achieve the same quality of results. The best function value found differs by several orders of magnitude, e.g. 1.14394e-08 vs. 3.62874e-15 for $n = 20$ (the optimum value is 0). This may be due to a faster convergence, i.e. the stop criterion was true further away from the true optimum.

Even though CMA-ES uses many less function evaluations when using a local meta model the wall clock run time is actually larger than without it. This can be seen again in the Table 1 where run times in seconds are given. For dimensions $n = 2$ and $n = 5$ both methods are approximately the same in speed. However, for dimensions $n > 5$ CMA-ES with the local meta model is significantly slower than CMA-ES. This difference becomes more pronounced with increasing dimension; for $n = 30$ CMA-ES converges in 18 seconds while CMA-ES with local meta model takes more than 3 hours. This trend for large dimensions becomes even more clear in Figure 2, lower graph, where the wall clock time is plotted as a function of the dimensionality.

| | CMA-ES with local meta model | | | | CMA-ES | | |
|---|---|---|---|---|---|---|---|
| $n$ | virt. fevals | fevals | Function value | Run Time | fevals | Function value | Run Time |
| 2 | 389 | 80 | 9.08902e-10 | 1.505 | 513 | 1.29806e-16 | 1.623 |
| 5 | 1117 | 174 | 6.10398e-10 | 2.819 | 1346 | 9.89092e-16 | 2.585 |
| 10 | 2592 | 342 | 2.97969e-09 | 15.471 | 3066 | 1.54028e-15 | 3.664 |
| 15 | 4520 | 538 | 6.83864e-09 | 123.334 | 5526 | 2.33629e-15 | 5.737 |
| 20 | 6787 | 826 | 1.14394e-08 | 812.699 | 8027 | 3.62874e-15 | 8.518 |
| 25 | 9889 | 1138 | 3.841e-08 | 3787.277 | 11535 | 4.35024e-15 | 12.096 |
| 30 | 13010 | 1466 | 3.93902e-08 | 12326.788 | 15245 | 3.73597e-15 | 17.340 |

**Table 1.** Average results for 5 runs each, optimisation of $f_{\text{Schwefel}}$ (run time given in seconds)




(a) $f_{\text{Schwefel}}$

(b) $f_{\text{Rosenbrock}}$

(c) $f_{\text{Rastrigin}}$

**Figure 1.** Test functions plotted in 2 dimensions

The reason for this is the computational effort to build the local meta model (determine its parameters) where a lot of computational time is spent.

## 4.2 Results for $f_{\text{Rosenbrock}}$

With the slightly more difficult $f_{\text{Rosenbrock}}$ function once more CMA-ES uses many fewer fevals when using the local meta model. This can be seen in the Table 2 and the graphs in Figure 3. When considering the function values (middle graph in Figure 3) one can see a considerable variation in the results. This is valid for both approaches. The reason for this are cases in which the optimisers converge to a wrong value of around 3.9. Apart from this both methods show the same quality of results.

With regards to the wall clock run time the effects seen with $f_{\text{Schwefel}}$ are even more pronounced, with CMA-ES being much faster without the local meta model. Unlike with $f_{\text{Schwefel}}$ CMA-ES with the local meta model is already slower at dimension $n = 2$. For $n = 25$ the run time is around 18 hours, but only 22 seconds with the original CMA-ES.

## 4.3 Results for $f_{\text{Rastrigin}}$

When optimising this most difficult function of the three one can see that the CMA-ES with the local meta model never uses less fevals than CMA-ES, see Table 3. This is in agreement with the results reported by Kern *et al.* [8], even if they did not use the standard population sizes but instead larger values, taken from [5]. When looking at the graphs in Figure 4 one can see a particularly wide spread in the results around $n = 15$ when using the local model. In some cases more fevals are needed here than with $n = 25$, which points to significant convergence problems. For larger dimensions both variants of CMA-ES exhibit the same results for fevals.

When looking at the function values both variants show a large variation in their results. At $n \leq 10$ CMA-ES with a local meta model is slightly better than CMA-ES but with larger dimensions this is quite different. However, even here the difference between the two is smaller than the variance of the results so that we cannot easily judge them.

Considering the run time again the original CMA-ES is the clear winner. Due to the convergence problems around dimension $n = 15$ CMA-ES with the local meta model sometimes takes more time than with $n = 20$. In one run it took 73153 virtual fevals (i.e. calls to the evaluation function, which may give the meta model's value or that of the objective function) while the largest value for $n = 20$ was 16899 virtual fevals. Considering the number of virtual fevals, still with the local meta model, on average 33744 virtual fevals at $n = 15$ and 20262 virtual fevals at $n = 25$, but still taking three times as much time, it is very clear that the run time is more dependent on the dimension than on the number of virtual fevals.

| | CMA-ES with local meta model | | | | CMA-ES | | |
|---|---|---|---|---|---|---|---|
| $n$ | virt. fevals | fevals | Function value | Run Time | fevals | Function value | Run Time |
| 2 | 998 | 253 | 2.06051e-10 | 2.849 | 760 | 1.01779e-09 | 0.959 |
| 5 | 3139 | 771 | 1.57234 | 7.339 | 2653 | 1.57672e-09 | 1.721 |
| 10 | 8780 | 2313 | 2.39195 | 136.859 | 7337 | 1.59463 | 3.786 |
| 12 | 14443 | 4076 | 4.20586e-09 | 325.753 | 9949 | 6.02164e-09 | 4.465 |
| 15 | 22494 | 6197 | 1.59465 | 1290.499 | 13642 | 6.89638e-09 | 6.205 |
| 17 | 29160 | 8063 | 3.64668e-08 | 3008.373 | 16383 | 0.797325 | 7.968 |
| 20 | 41751 | 11658 | 0.797325 | 10088.182 | 20932 | 7.54821e-09 | 11.493 |
| 25 | 79277 | 24036 | 1.07025e-07 | 63827.514 | 33333 | 6.12533e-09 | 21.981 |

**Table 2.** Average results for 5 runs each, optimisation of $f_{\text{Rosenbrock}}$ (run time given in seconds)

| | CMA-ES with local meta model | | | | CMA-ES | | |
|---|---|---|---|---|---|---|---|
| $n$ | virt. fevals | fevals | Function value | Run Time | fevals | Function value | Run Time |
| 2 | 684 | 172 | 2.58689 | 2.070 | 724 | 2.18891 | 2.212 |
| 5 | 4619 | 1612 | 9.15361 | 15.015 | 1742 | 8.35764 | 3.133 |
| 10 | 13494 | 6201 | 16.5163 | 270.313 | 3592 | 17.9092 | 4.400 |
| 15 | 33744 | 19691 | 39.7982 | 3996.487 | 5610 | 22.0881 | 5.538 |
| 20 | 15258 | 6024 | 35.4868 | 2945.690 | 6098 | 30.4457 | 6.191 |
| 25 | 20262 | 8562 | 61.4883 | 11916.86 | 7315 | 50.7428 | 7.023 |

**Table 3.** Average results for 5 runs each, optimisation of $f_{\text{Rastrigin}}$ (run time given in seconds)

## 5 CONCLUSIONS

We have considered a variant of CMA-ES which uses a local meta model to reduce the number of function evaluations in an evolutionary algorithm to optimise a function. From our observations and comparisons with the original CMA-ES it became clear that this model can indeed reduce the number of function evaluations. However, the complexity of the model causes a dramatic increase of run time for medium to large dimensions ($n > 5$) of the search space.

With the test function $f_{\text{Schwefel}}$ the variant with the local meta model has shown the best potential for an improvement over the original variant. However, the resulting best function values found by the optimiser were not as good as the ones found with CMA-ES. For the more complex test functions $f_{\text{Rosenbrock}}$ and $f_{\text{Rastrigin}}$ not even a reduction in the number of function evaluations could be achieved.

In conclusion, the use of a local meta model does look like a viable way to avoid expensive function evaluations for some problems. However, until a different meta model—or a significantly faster way to determine its parameters—is found, the original version of CMA-ES outperforms this new variant especially for large dimensions of the problem space.

## ACKNOWLEDGMENTS

The program code used for our experiments is based on the original CMA-ES and LMM-CMA-ES code which has been provided by their authors, Nikolaus Hansen and Stefan Kern. We are grateful for their generosity.

## REFERENCES

[1] Christopher G Atkeson, Andrew W Moore, and Stefan Schaal, 'Locally weighted learning', *Artificial Intelligence Review*, **11**(1), 11–73, (February 1997).
[2] Richard Ernest Bellman, *Adaptive Control Processes*, Princeton University Press, Princeton, USA, 1961.
[3] William S Cleveland, 'Robust locally weighted regression and smoothing scatterplots', *Journal of the American Statistical Association*, **74**(368), 829–836, (1979).
[4] Ágoston E Eiben and James E Smith, *Introduction to Evolutionary Computing*, Springer Verlag, Berlin, Germany, 2003.
[5] Nikolaus Hansen and Stefan Kern, 'Evaluating the CMA evolution strategy on multimodal test functions', in *Proceedings of the Eighth International Conference on Parallel Problem Solving from Nature (PPSN VIII)*, pp. 282–291, Birmingham, UK, (September 2004).
[6] Nikolaus Hansen and Andreas Ostermeier, 'Completely derandomized self-adaptation in evolution strategies', *Evolutionary Computation*, **9**(2), 159–195, (2001).
[7] Christian Igel, 'Neuroevolution for reinforcement learning using evolution strategies', in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2003)*, 2588–2595, IEEE Press, (2003).
[8] Stefan Kern, Nikolaus Hansen, and Petros Koumoutsakos, 'Local metamodels for optimization using evolution strategies', in *Proceedings of the Ninth International Conference on Parallel Problem Solving from Nature (PPSN IX)*, pp. 282–291, Birmingham, UK, (September 2006).
[9] Leonard Andreevich Rastrigin, *Extremal Control Systems*, volume 3 of *Theoretical Foundations of Engineering Cybernetics Series*, Nauka, Moscow, Russia, 1974. (in Russian).
[10] Howard H Rosenbrock, 'An automatic method for finding the greatest or least value of a function', *Computer Journal*, **3**(3), 175–184, (March 1960).
[11] Warren S Sarle. Ill-conditioning in neural networks. Website, September 1999. `ftp://ftp.sas.com/pub/neural/illcond/illcond.html`.
[12] Hans-Paul Schwefel, *Evolution and Optimum Seeking*, John Wiley & Sons, New York, USA, 1995.
[13] Nils T Siebel and Gerald Sommer, 'Evolutionary reinforcement learning of artificial neural networks', *International Journal of Hybrid Intelligent Systems*, **4**(3), 171–183, (October 2007). ISSN 1448-5869.

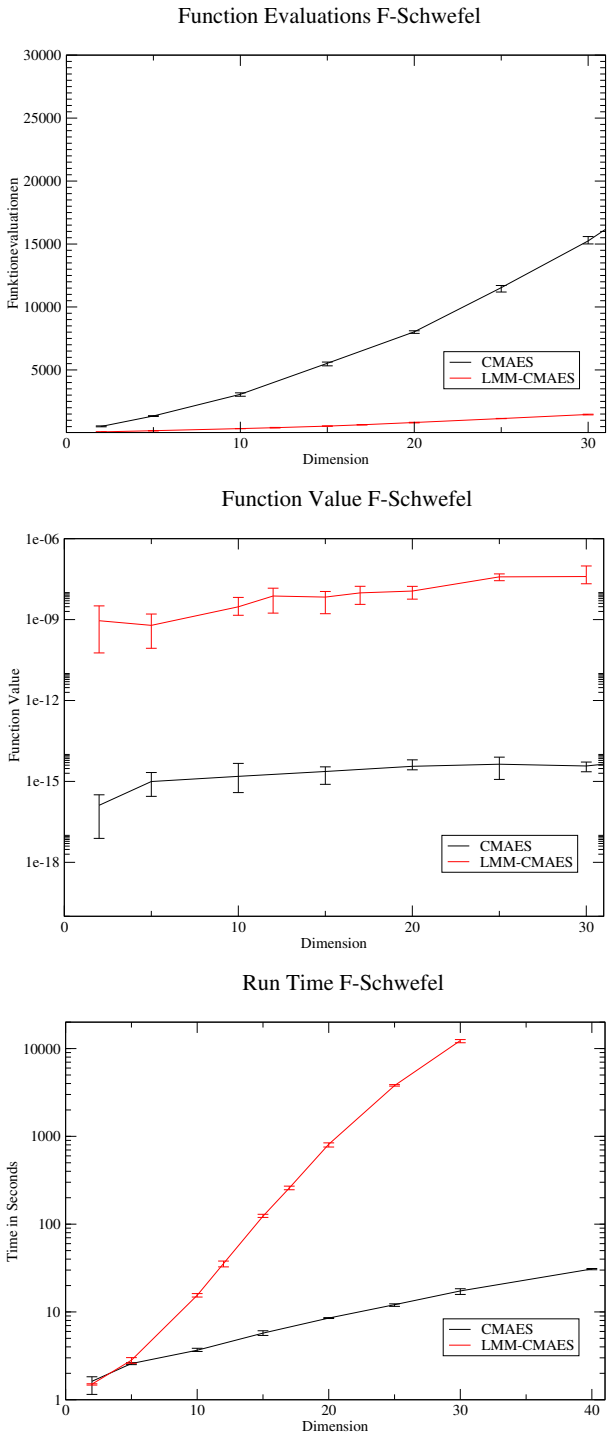



**Figure 2.** fevals, resulting function value and run time (logarithmic scale) for $f_{\text{Schwefel}}$, CMA-ES and CMA-ES with local meta model ("LMM-CMA-ES")

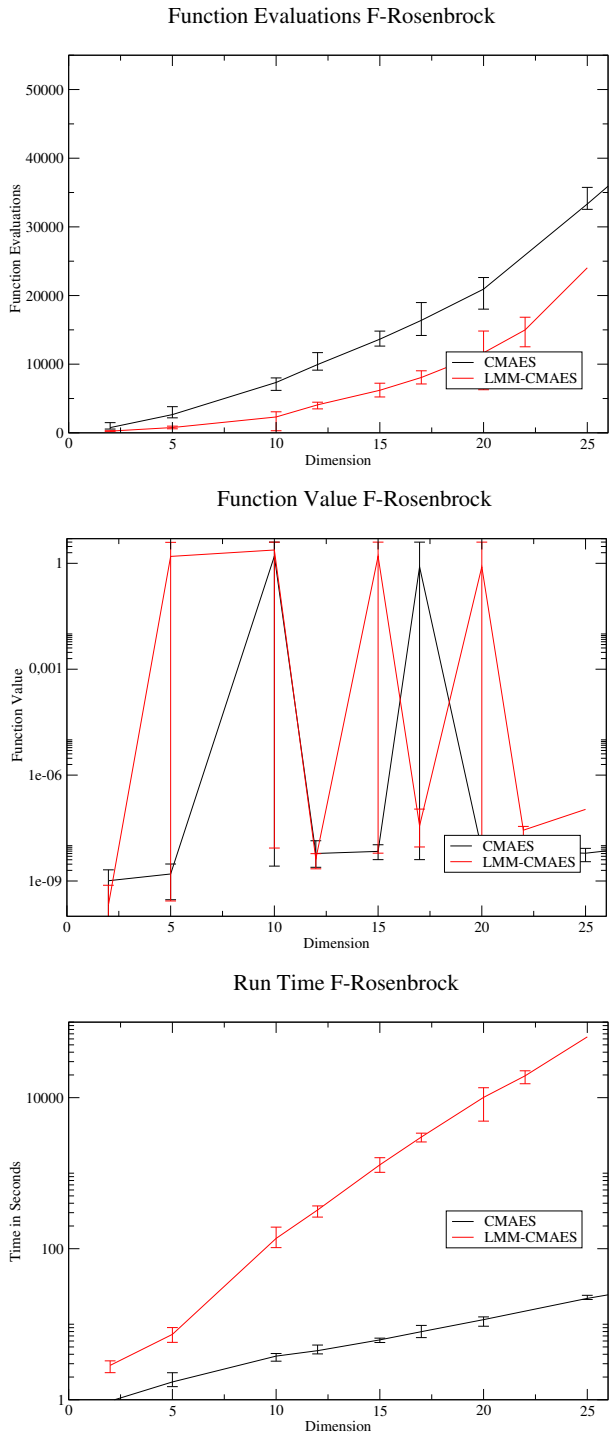



**Figure 3.** fevals, resulting function value and run time (logarithmic scale) for the $f_{\text{Rosenbrock}}$ test function, CMA-ES and CMA-ES with local meta model ("LMM-CMA-ES")

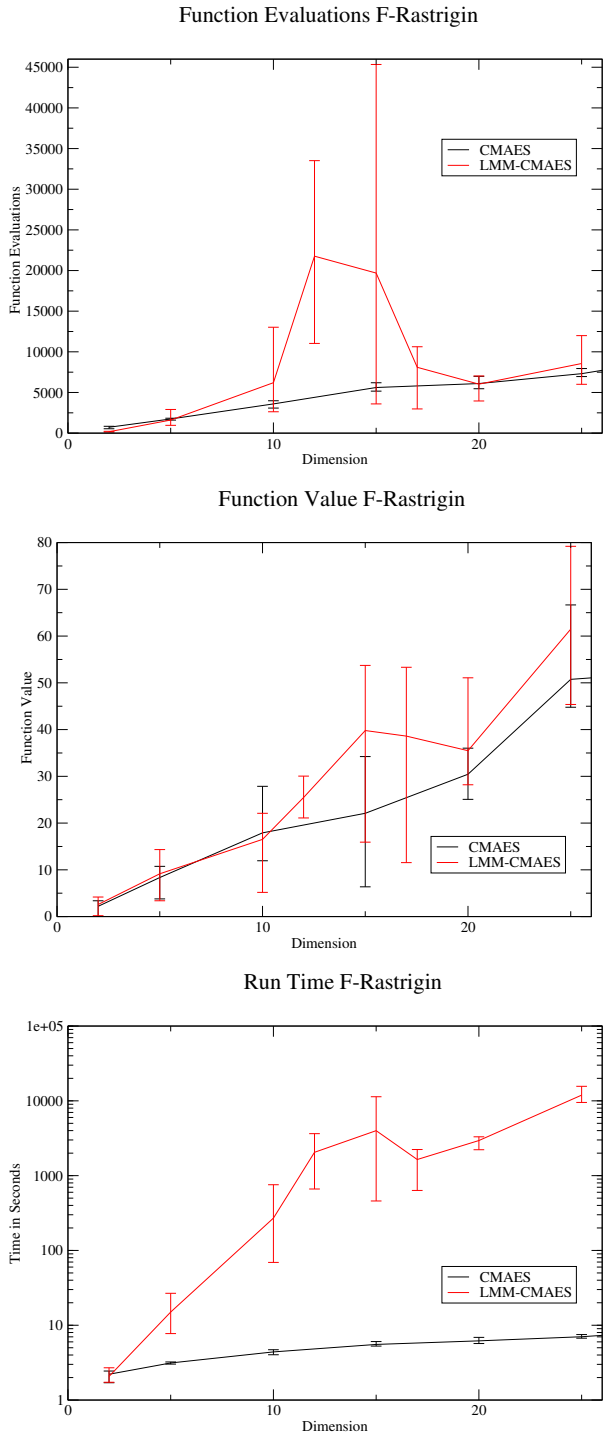



**Figure 4.** fevals, resulting function value and run time (logarithmic scale) for the $f_{\text{Rastrigin}}$ test function, CMA-ES and CMA-ES with local meta model ("LMM-CMA-ES")

4th International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2011)
Berlin, Germany, December 9 2011

# Evolving Augmented Neural Networks in Compressed Parameter Space

**Yohannes Kassahun** [1]

**Abstract.** In this paper we present *preliminary* results on evolving augmented neural networks in compressed parameter space. An augmented neural network is a cascade of $\alpha\beta$ filters and a feed-forward multilayer perceptron. The parameters of the augmented neural network are represented using a weighted sum of orthogonal functions. We show that we need to optimize fewer parameters (the weights of the orthogonal functions) than the number of parameters in the augmented neural networks, and thereby accelerating the evolutionary process. We tested the concept on single and double pole balancing experiments, and found solutions faster than the reported speed in the literature at which solutions are found.

## 1 Introduction

In robotics and many other research fields, the learning task we want to solve is most of the time noisy (noisy sensors and actuators) and partially-observable in addition to being complex (high dimensional state and action spaces). Partially-observable problems have been a challenging domain for machine-learning algorithms. The fundamental reason for this is that such problems place limits on an agent's ability to fully perceive the states of the environment, and in doing so, limit the information upon which an agent can base its decisions. Neuroevolutionary methods have delivered promising results in recent years as methods of solving such learning tasks. In the past, we proposed a solution, where we tried to simplify the topology of the evolved neural network, and as a consequence, reduce the time required to find a solution [11]. The approach exploits the use of a Kalman filter as an input layer for the neural network to be evolved. The Kalman filter inherently provides the system with memory (as recurrent connections would) to estimate and thus recover the unobserved missing state variables. From the viewpoint of the neural network to be evolved (whose inputs are the outputs of the Kalman filter layer), the state information has been augmented and is noise-free. Clearly, this additional information provided to the system permits a simpler neural network solution, thus significantly reducing the time required to find a solution.

Complex tasks require correspondingly complex solutions with many parameters to evolve. This again requires long training time. In this paper, we combine the approach proposed by Koutntik et al. [9] with the augmented neural network to reduce the training time. The paper is organized as follows. First we present the augmented neural network and then we discuss the experiments and results obtained.

[1] Robotics Group, University of Bremen, Robert-Hooke-Str. 5, D-28359, Bremen, Germany, email: kassahun@informatik.uni-bremen.d

## 2 Augmented Neural Network with Kalman Filter (ANKF)

The augmented neural network with Kalman Filter (ANKF) to be evolved is made up of a neural network and a predictor that can estimate the next state based on the *current partially-observable state* (which is possibly corrupted by noise). The predictor we use is composed of $n$ Kalman filters ($\alpha\beta$ filters, see Section 2.1) $\{KF_1, KF_2, \ldots, KF_n\}$ one for each of the $n$ sensory readings, as shown in Figure 1. The outputs of these Kalman filters are connected to a feed-forward neural network $NN$, whose outputs control the plant. A Kalman filter $KF_i$ is used to estimate the sensor value $\hat{x}_i$ and the missing value $\hat{\dot{x}}_i$ from the measured (observed) value $z_i$, where $i \in [1, n]$ and $n$ is the number of observable state variables. The quantity $u_j$, where $j \in [1, m]$, represents a control signal that is

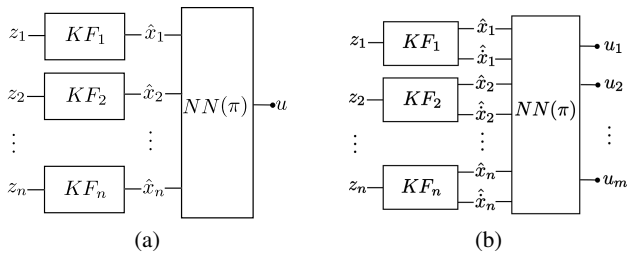



**Figure 1.** Two ways of using the augmented neural network: (a) The Kalman filter $KF_i$ is used to estimate the sensor value $\hat{x}_i$ from the measured (observed) value $z_i$ in the case of complete state variables. (b) The Kalman filter is used to estimate the sensor value $\hat{x}_i$ and the missing value $\hat{\dot{x}}_i$ from the measured (observed) value $z_i$ in the case of incomplete state variables. The quantity $u_j$ represents the control signal that is sent to the plant to be controlled. $NN$ is a feed-forward neural network representing a policy $\pi$.

used to control a plant. The use of Kalman filters provides memory to the system and as a result enables the system to recover missing variables. Because of this, it is not necessary for the neural network to have a recurrent connection, and the use of a feed-forward neural network for the policy $\pi$ to be learned is sufficient. The whole controller is a non-linear function given by

$$u_j = f(w_1, \ldots, w_k, \gamma_1, \ldots, \gamma_n; z_1, \ldots, z_n), \qquad (1)$$

where $u_j$ ($j \in [1, m]$) is one of the outputs of the feed-forward neural network, $w_1, \ldots, w_k$ are the weights of the neural network, $\gamma_1, \ldots, \gamma_n$ are the tracking indices (see Section 2.1) of the $\alpha\beta$ filters realizing $KF_1, \ldots KF_n$, and $z_1, \ldots, z_n$ are the measured values of

input sensors. The index $k$ is greater than or equal to $n$ ($k \geq n$) in the case of complete state variables, and it is greater than or equal to $2n$ ($k \geq 2n$) in the case of incomplete state variables[2]. Table 1 summarizes the usage of the augmented neural network for different task environments.

## 2.1 The $\alpha\beta$ Filter

A Kalman filter $KF_i$ in Figure 1 is realized using an $\alpha\beta$ filter, which is a particular case of the general Kalman filter where the velocity is assumed to be constant. The filter is usually used in tracking applications. The neural network equivalent of the filter is shown in Figure

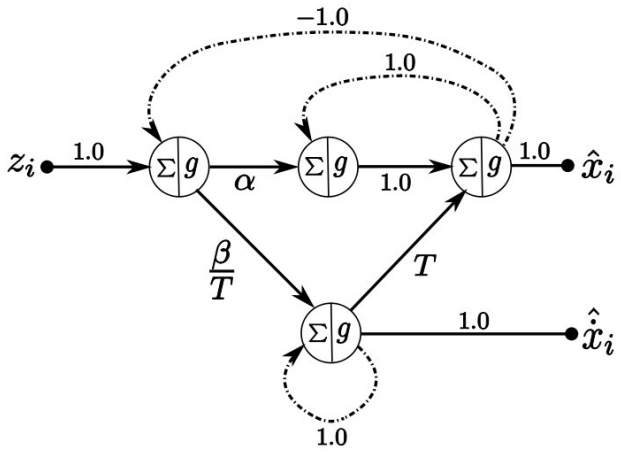



**Figure 2.** An $\alpha\beta$ filter for a single input $z_i$. The activation function $g$ is the identity function, and the recurrent connections have a unit delay.

2. As can be seen in the figure, all the weights of the filter have a magnitude of 1 except for $\alpha$, $\beta$ and $T$, where $T$ is th sampling period. The optimal values for $\alpha$ and $\beta$ are derived by Kalata [10] for assumed variance of both measurement and process noises ($\sigma_v$ and $\sigma_w$) and are given by $\alpha = 1 - r^2$ and $\beta = 2(1 - r)^2$ respectively, where $r = \frac{4+\gamma-\sqrt{8\gamma+\gamma^2}}{4}$ and $\gamma = \frac{T^2\sigma_w}{\sigma_v}$. The term $\gamma$ is referred to as a *tracking index*. Since the parameters $\alpha$ and $\beta$ depend only on $\gamma$, an optimization algorithm *needs only to find a single parameter* $\gamma$ per filter that results in the desired filter performance. An extension of the $\alpha\beta$ filter is the $\alpha\beta\gamma$ filter [4], which is based on a constant acceleration model and is better suited for the tracking of complex signals. Like the $\alpha\beta$ filter, an optimization algorithm needs only to find a single tracking index $\gamma$ per filter to get the desired filter performance. This enables one filter to be easily exchanged with the other.

## 2.2 Evolving Parameters of Augmented Neural Network in Compressed Parameter Space

In this section we assume that the feed-forward neural network of the augmented neural network has no hidden layer, and thus we can assume that we have a vector of parameters to optimize. The number of parameters to optimize in uncompressed parameter space is given by $3n$ for incomplete state variables, where $n$ is the number of inputs to the augmented neural network. Note that the parameters of the $\alpha\beta$ filter are optimized simultaneously with the weights of the feed-forward network. The feed-forward network used in this paper has only one output neuron with linear activation function and the output weight is not optimized.

In order to speed-up the neuroevolutionary process, we approximate $w_k$ using

$$w_k = a_0\phi_0(t_k) + a_1\phi_1(t_k) + a_2\phi_2(t_k) + \ldots + a_M\phi_M(t_k), \quad (2)$$

where $\phi_0(t_k), \phi_1(t_k), \phi_2(t_k), \ldots, \phi_M(t_k)$ form an orthogonal set of basis functions, $t_k \in [0, 1]$ is a parameterization variable and $M < 3n$. The parametrization parameter $t_k = 0$ corresponds to the first parameter in the uncompressed parameter space, and $t_k = 1$ corresponds to the last parameter in uncompressed parameter space. For example, the set $\{1, \cos(\pi t); \cos(2\pi t), \ldots, \cos(M\pi t)\}$ forms an orthogonal set of basis functions over the interval $[-1, 1]$. In the compressed space, we evolve the parameters $\{a_0, a_1, \ldots, a_M\}$.

## 2.3 CMA-ES

CMA-ES [7] is used to evolve the parameters of the augmented neural network in both uncompressed and compressed parameter space. CMA-ES is an advanced form of evolution strategy which can perform efficient optimization even for small population sizes. Each individual is represented by an $n-$dimensional real valued solution vector. The solutions are altered by recombination and mutation. Mutation is realized by adding a normally distributed random vector with zero mean, where the covariance matrix of this distribution itself is adapted during evolution to improve the search strategy. CMA-ES uses important concepts like *derandomization* and *cumulation*. Derandomization is a deterministic way of altering the mutation distribution such that the probability of reproducing steps in the search space that lead to better individuals is increased. A sigma value represents the standard deviation of the mutation distribution. The extent to which an evolution has converged is indicated by this sigma value (smaller values indicate greater convergence). Moreover, the algorithm detects correlations between object variables (i. e. variables in the vector to be optimized), and is invariant under orthogonal transformations of the search space. Correlations between object variables are detected by analyzing the search path of a population over several past generations. These correlations are stored in the covariance matrix and guide the future search path in a promising direction. This principle is known as cumulation.

# 3 Description of Experiments and Results

In this section, we describe the experimental setup, summary of results obtained using $\alpha\beta$ filter in uncompressed parameter space, and results obtained using $\alpha\beta$ filter in compressed parameter space

## 3.1 The Double Pole Balancing Problem Without Velocities Benchmark

The pole balancing system has one or more poles hinged to a wheeled cart on a finite length track. The movement of the cart and the poles are constrained within a 2-dimensional plane. The objective is to balance the poles indefinitely by applying a force to the cart at regular time intervals, such that the cart stays within the track boundaries. An attempt to balance the poles fails if either (1) the angle from vertical of any pole exceeds a certain threshold, or (2) the cart leaves the track boundaries.

[2] By incomplete state variables we mean a set of state variables whose first order derivative with respect to time are missing.

| | **Noise-free environment** | **Noisy environment** |
|---|---|---|
| **Complete state variables** | *Completely observable domain*. Use the setup shown in Figure 1 (a). | *Partially observable domain due to noise*. Use the setup shown in Figure 1 (a). |
| **Incomplete state variables** | *Partially observable domain due to missing state variables*. Use the setup shown in Figure 1 (b). | *Partially observable domain due to noise and missing state variables*. Use the setup shown in Figure 1 (b). |

**Table 1.** Usage of the augmented neural network (ANKF) for different task environments.

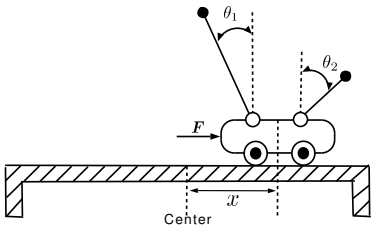



**Figure 3.** The double pole balancing problem. The poles must be balanced simultaneously by applying a continuous force $F$ to the cart. The parameters $x$, $\theta_1$ and $\theta_2$ are the offset of the cart from the center of the track, and the angles from the vertical of the long and short pole, respectively.

In the double pole balancing without velocities benchmark, the controller observes only $x$, $\theta_1$, and $\theta_2$, but *not* $\dot{x}$, $\dot{\theta_1}$, and $\dot{\theta_2}$. A fitness function introduced by Gruau et al. is used in connection with this benchmark [6]. The fitness function is the weighted sum of two separate fitness measurements $f = 0.1f_1 + 0.9f_2$ taken over 1000 timesteps.

$$f_1 = t/1000$$

$$f_2 = \begin{cases} 0 & \text{if} \quad t < 100 \\ \frac{0.75}{\sum_{i=t-100}^{t}\left(|x_i|+|\dot{x}_i|+|\dot{\theta}_{1,i}|+|\dot{\theta}_{2,i}|\right)} & \text{otherwise,} \end{cases} \tag{3}$$

where $t$ is the number of time steps the pole is balanced starting from a fixed initial position. In the initial position, all states are set to zero except $\theta_1 = 4.5°$. The angle of the poles from the vertical must be in the range $[-36°, 36°]$. The defined fitness function favors controllers that can keep the poles near the equilibrium point and minimize the amount of oscillation. The first fitness measure $f_1$ rewards successful balancing, while the second measure $f_2$ penalizes oscillations. The evolution of the neural controllers is stopped when a champion of a generation passes two tests. First, it has to balance the poles for $10^5$ timesteps starting from the $4.5°$ initialization. Second, it has to balance the poles for 1000 steps starting from at least 200 out of 625 different initial starting states. Each start state is chosen by giving each state variable $(x, \dot{x}, \theta_1, \dot{\theta_1}, \theta_2, \dot{\theta_2})$ one of the values 0.05, 0.25, 0.5, 0.75, 0.95,0, 0, scaled to the range of each input variable. The ranges of the input variables are $\pm 2.16$ m for $x$, $\pm 1.35$ m/s for $\dot{x}$, $\pm 3.6°$ for $\theta_1$, and $\pm 8.6°$ for $\dot{\theta_1}$. The number of successful balances is a measure of the generalization performance of the best solution.

The fitness function $f$ is not directly proportional to the performance of an individual in the two tests. Therefore, at the end of a given generation, the controller with the highest fitness is *not necessarily* the controller which performs better on the two tests. It is possible that there could be another controller which was assigned a lower fitness, but has better performance on the two tests. This forces neuroevolutionary methods to be more explorative, and therefore results in a larger number of evaluations needed to solve the benchmark.

## 3.2 Results Obtained in Uncompressed Parameter Space.

The augmented neural network has been tested on the *double pole balancing without velocities* benchmark, and has achieved significantly better results on this benchmark than the published results of other algorithms to date[3]. Table 2 shows the best result obtained by the augmented neural network along with the best results from the literature.

**Table 2.** Results for the double pole-balancing benchmark in uncompressed parameter space using Gruau's fitness measure. Average over 50 independent evolutions

| **Without velocities** | | |
|---|---|---|
| **Method** | **Evaluations** | **Generalization** |
| CE[5] | 840,000 | 300 |
| SANE [12] | 451,612 | - |
| CNE [14] | 87,623 | - |
| ESP [3] | 26,342 | - |
| AGE [1] | 25,065 | 317 |
| EANT [11] | 15,762 | 262 |
| NEAT [13] | 6,929 | - |
| CoSyNE [2] | 3,416 | - |
| CMA-NeuroES [8] | 1,141 | - |
| Augmented neural network [11] | 482 | 455 |

## 3.3 Results Obtained in Compressed Parameter Space

Table 3 shows the performance of learning in compressed parameter space for both single and double pole balancing experiments without velocity information. For this experiment, evolution of augmented neural network in compressed parameter space outperforms significantly the evolution of recurrent neural network in compressed parameter space. The increase in performance is due to the simplification of neural networks through $\alpha\beta$ filters.

[3] All results for the augmented neural network in this paper can be reproduced using the software that can be downloaded at `http://sourceforge.net/projects/eant-project/`.

**Table 3.** Results for the single and double pole-balancing benchmark in compressed parameter space using Gruau's fitness measure. Average over 50 independent evolutions. DOF stands for discrete orthogonal functions.

| Task | Method | Parameters | Evaluations |
|---|---|---|---|
| 1 pole non-markov | CoSyNE + DCT [9] | 4 | 151 |
| 1 pole non-markov | ANKF + DOF | 3 | 12 |
| 2 poles non-markov | CoSyNE + DCT [9] | 5 | 3421 |
| 2 poles non-markov | ANKF + DOF | 5 | 480 |

## 4 Conclusion

We have shown that by evolving the parameters of the augmented neural network in a compressed space, it is possible to accelerate neuroevolution for partially observable domains. The results presented in this paper are *preliminary*, and the method has to be tested on more complex problems to assess the feasibility of evolving augmented neural network for complex problems in compressed parameter space.

## REFERENCES

[1] P. Dürr, C. Mattiussi, and D. Floreano. Neuroevolution with analog genetic encoding. In *Proceedings of the 9th Conference on Parallel Problem Solving from Nature (PPSN IX)*, pages 671–680, 2006.

[2] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 9:937–965, 2008.

[3] F. J. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342, 1997.

[4] J. E. Gray and W. Murray. A derivation of an analytic expression for the tracking index for alpha-beta-gamma filter. *IEEE Transactions on Aerospace and Electronic Systems*, 29(3):1064–1065, 1993.

[5] F. Gruau. *Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm*. PhD thesis, Ecole Normale Superieure de Lyon, Laboratoire de l'Informatique du Parallelisme, France, January 1994.

[6] F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming: Proceedings of the First Annual Conference*, pages 81–89, Standford University, CA, USA, 1996. MIT Press.

[7] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[8] V. Heidrich-Meisner and C. Igel. Neuroevolution strategies for episodic reinforcement learning. *Journal of Algorithms*, 2009. doi:10.1016/j.jalgor.2009.04.002.

[9] J. Koutntik, F. Gomez, and J. Schmidhuber. Evolving neural networks in compressed weight space. In *Proceedings of Genetic and Evolutionary Computation Conference GECCO*, pages 619–626, 2010.

[10] Paul R. Kalata. Alpha-beta target tracking systems: A survey. In *American Control Conference*, pages 832–836, 1992.

[11] Y. Kassahun, J. de Gea, M. Edgington, J. H. Metzen, and F. Kirchner. Accelerating neuroevolutionary methods using a Kalman filter. In *Proceedings of the 10th Genetic and Evolutionary Computation Conference (GECCO-2008)*, pages 1397–1404, 2008.

[12] D. Moriarty and R. Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–33, 1996.

[13] K. O. Stanley. *Efficient Evolution of Neural Networks through Complexification*. PhD thesis, Artificial Intelligence Laboratory. The University of Texas at Austin., Austin, USA, August 2004.

[14] A. Wieland. Evolving controls for unstable systems. In *Proceedings of the International Joint Conference on Neural Networks*, pages 667–673, 1991.