Evolvable Mathematical Models: A New Artificial Intelligence Paradigm

by

Paul Grouchy

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Aerospace Engineering
University of Toronto

Abstract

Evolvable Mathematical Models: A New Artificial Intelligence Paradigm

Paul Grouchy
Doctor of Philosophy
Graduate Department of Aerospace Engineering
University of Toronto
2014

We develop a novel Artificial Intelligence paradigm to generate autonomously artificial agents as mathematical models of behaviour. Agent/environment inputs are mapped to agent outputs via equation trees which are evolved in a manner similar to Symbolic Regression in Genetic Programming. Equations are comprised of only the four basic mathematical operators, addition, subtraction, multiplication and division, as well as input and output variables and constants. From these operations, equations can be constructed that approximate any analytic function. These Evolvable Mathematical Models (EMMs) are tested and compared to their Artificial Neural Network (ANN) counterparts on two benchmarking tasks: the double-pole balancing without velocity information benchmark and the challenging discrete Double-T Maze experiments with homing. The results from these experiments show that EMMs are capable of solving tasks typically solved by ANNs, and that they have the ability to produce agents that demonstrate learning behaviours. To further explore the capabilities of EMMs, as well as to investigate the evolutionary origins of communication, we develop *NoiseWorld*, an Artificial Life simulation in which interagent communication emerges and evolves from initially noncommunicating EMM-based agents. Agents develop the capability to transmit their x and y position information over a one-dimensional channel via a complex, dialogue-based communication scheme. These evolved communication schemes are analyzed and their evolutionary trajectories examined, yielding significant insight into the emergence and subsequent evolution of cooperative communication. Evolved agents from *NoiseWorld* are successfully transferred onto physical robots, demonstrating the transferability of EMM-based AIs from simulation into physical reality.

Dedication

This thesis is dedicated to my parents.

It is thanks to your unwavering love and support that I am here today.

Acknowledgements

I am deeply grateful to my supervisor, Professor Gabriele M.T. D'Eleuterio, for his invaluable advice, guidance, and ideas. I could not have asked for a better mentor. I would also like to thank Jekanthan Thangavelautham, David Beach, Alexander Ho, Alexander Smith, Jonathan Gammell, Peter Szabo, Adam Trischler, Vidya Menon, Adam Sniderman, and all the other members of the UTIAS Space Robotics Group that I have had the pleasure of working alongside. Throughout my graduate studies, I have been surrounded by wonderful, intelligent people.

Thanks as well to Dr. Craig Boutilier and the members of my Doctoral Examination Committee, Dr. Hugh Liu, Dr. Chris Damaren, and Dr. Richard Zemel, for their helpful questions, suggestions, and support. To Dr. Marco Mirolli: Thank you for your detailed reading, editing, and critiques.

I would like to acknowledgement the generous support of the Natural Sciences and Engineering Research Council of Canada.

A very special thank you to Dr. Jaakko Hollmén for welcoming me to Finland (with the help of IAESTE) and for introducing me to Machine Learning. Thanks to Dr. Taisuke Sato and Dr. Yoshitaka Kameya, as well as the Japan Society for the Promotion of Science, for generously hosting me in Tokyo. Finally, I am forever indebted to Dr. Hod Lipson and the members of the Creative Machines Lab at Cornell for welcoming me to Ithaca, New York. My time there was one of immense personal and professional growth, and an experience that I will not soon forget.

To Graham Holker, thank you for introducing me to Genetic Programming. To Dr. Jeff Clune, your work and passion for Evolutionary Robotics is inspirational.

To McLean, Tony, Tien, Mel, Lindsay, Jenne, Mark, Maggie, Andrea, Alexis, Mike, Laura, Chris, Frank, Maji, Janek, Katherine, Shayan, Shaina, Bradley, Joe, Kwan, Asako, Anita, Jenny, Girish, Varun, Aamod, Nikhil, and all my other friends in Toronto and around the world, thank you. I am so lucky to have such fantastic people in my life.

To the radio.beats.to crew (booya aka greenleader, ddub, Hubert K, precision, magoo, McHat, nyquil), I don't know if I would've made it without you. Is it Wednesday yet?

To Sarah, thank you so much for your love, support, and patience. I am blessed to have such a beautiful person by my side.

And finally, to Mom and Dad, who have worked tirelessly so that I might live this charmed life. I owe you everything, but for now please accept my undying love and gratitude.

Paul Grouchy

Toronto, Ontario, Canada

July 6th, 2014

Foreword

Ever since I can remember, I have been fascinated by the natural world. Such enormous complexity and diversity! Even a relatively simple organism such as an Earthworm has a fascinating range of behaviours. Not only are they able to dig through the ground, find food, survive through harsh winters and escape predators (sometimes...), they are also able to use materials around them to produce offspring, bringing new life into our Universe. This seems nothing short of miraculous, and yet it is only the tip of the iceberg when it comes to "Life."

As I grew older and moved through the education system, my path took me away from Nature and into the world of Human accomplishments: Mathematics, Digital Computation, Engineering, Robotics. I learned that while we have achieved much as a species, we still seem quite far from creating our own Earthworm. Obviously technological progress takes time, but perhaps we are approaching such challenges from too human-centric a point of view (as is often the case). Humanity seems to view itself as outside of Nature, almost as Gods. We can level mountains, kill any beast, reach any depth and height and even bend DNA to our will, and so we think that we too can create our own version of "Life." And so we set to designing robots and artificial intelligences, taking what we know about the Universe and applying it in increasingly complex ways with great success. However, this "intelligent design" approach is most certainly not how the Earthworm came to be.

After completing my undergraduate studies at Queen's University in Kingston, Ontario, Professor Gabriele M. T. D'Eleuterio introduced me to the concept of Evolutionary Robotics. This class of design and optimization methods looks to create Artificial Intelligence and Artificial Life by applying what we know about Natural Evolution, the process that produced the Earthworm, and all other life that we know of. Here then was my opportunity to steer my education path back towards Nature: my graduate studies were to be about Mathematics, Digital Computation, Engineering, Robotics, and Evolution.

Contents

Table of Notation

1D	One-dimensional
2D	Two-dimensional
3D	Three-dimensional
AGE	Analog Genetic Encoding
AGI	Artificial General Intelligence
AI	Artificial Intelligence
ANN	Artificial Neural Network
C	Constant
C++	Programming language
C#	Programming language
cm	Centimeter
CPPN	Compositional Pattern Producing Network
CPU	Central Processing Unit
CTRNN	Continuous-Time Recurrent Neural Network
EC	Evolutionary Computation
EMM	Evolvable Mathematical Model
ER	Evolutionary Robotics
F	Force
f	Mathematical function
$\mathbf{f}$	Vector of mathematical functions
GA	Genetic Algorithm
GHz	Gigahertz
GP	Genetic Programming
GPU	Graphics Processing Unit
H	Shannon's information entropy
HyperNEAT	Hypercube-based NeuroEvolution of Augmenting Topologies
ID	Identification
kg	Kilogram
l	Length
LRNN	Locally Recurrent Neural Network
M	Number of experimenter-defined agent inputs
m	Meter
m^t	Modulatory signal

m	Mass
MATLAB	Matrix laboratory, a numerical computing environment/programming language
mm	Millimeter
N	Newton unit of force
N	Number of experimenter-defined agent outputs
N'	Number of extra equations/state variables of an EMM-based agent
n	Number of nodes/neurons in a network
NEAT	NeuroEvolution of Augmenting Topologies
P	Probability distribution
RNN	Recurrent Neural Network
S	Agent speed
s	Seconds
SR	Symbolic Regression
T	Transpose
t	Timestep variable
TCP/IP	Collection of communication protocols for the Internet and similar networks
u	Input variable
$\mathbf{u}$	Input vector
v	Output variable
$\mathbf{v}$	Output vector
w_{ij}	Connection weight from node i to node j
W	Maximum weight magnitude
β	Scaling multiplier for mutation and crossover probabilities
γ	Number of births over the past 10,000 timesteps on a given island
η	Learning rate
θ	Angle in radians
μ	Mean
ξ	Number of migrants received at the previous island synchronization
σ	Standard deviation
τ	Time constant
ϕ	Activation function
Ω	Set of all possible communication outputs
ω	Real-valued communication signal

Chapter 1

Thesis Introduction

Since the dawn of the digital age, Humanity has looked to reproduce intelligence *in silico*. Indeed, much progress has been made in the field of Artificial Intelligence (AI), with computers now able to beat the best living humans at several games requiring a high level of intelligence, such as chess and Jeopardy! AI has quietly crept into many facets of our lives, giving us personalized book and movie recommendations, keeping spam out of our inboxes, finding meaning within our troves of data, and soon, driving our cars. And yet, Artificial General Intelligence (AGI)—machine intelligence comparable to our own—has remained elusive. So, while we have AIs that can play chess and AIs that can drive cars, we have yet to produce an AI that can play chess *and* drive cars (*and* hold a conversation *and* make music *and* design spacecraft, etc.). In fact, we have yet to produce an AI with capabilities on par with lower lifeforms, such as a dog or even a mouse.

At the crux of the problem is our lack of a formal understanding of "intelligence," let alone how such a thing might arise from electrical and chemical signals such as those in a brain. The scientific and engineering discipline known as Machine Learning [13] looks to side-step this issue, however, by developing computer programs that can in turn design other computer programs to produce a desired behaviour. So, instead of having to understand the inner workings of a human's brain as she attempts to distinguish spam from legitimate emails, one can simply give a Machine Learning algorithm many examples of both (as determined by a human), and this algorithm will in turn produce an "intelligent" spam filter program [6]. Typically, a Machine Learning algorithm works in this fashion; solutions are produced via training on data sets, which may or may not require human-generated labels [69].

An alternative Machine Learning paradigm, Evolutionary Robotics (see Section 3.5), uses artificial evolution to "train" autonomous computer programs. AIs are evaluated on how well they perform on a given task (either in simulation or on hardware), receiving data inputs via sensors and having their outputs interpreted as behaviours, which influence themselves

and their environment. By repeatedly selecting and reproducing (with variation) AIs that perform relatively well on the prescribed task, intelligent mappings from sensor inputs to agent outputs can be *evolved*, without the need for prespecified training data. In Artificial Life experiments (see Section 7.2), lifelike behaviours can emerge directly from the interactions between AIs within a simulated world, without the need for experimenter-defined tasks.

A crucial component of any AI is its representation *in silico*. If a computer programmer were to design and build an AI by hand, the AI might be represented as a computer program directly, written in one of the many available programming languages. AIs represented in this fashion can also be created autonomously via Genetic Programming (see Section 3.3). Another potential representation for an AI is a Cellular Automaton [112], a spatial lattice of "cells" that each change state depending on its current state and the states of its neighbours. Cellular automata are capable of computation [89] and can be trained as binary classifiers using artificial evolution [105]. More complex classifier AIs can be represented and trained as probability distributions [80]. One of the most common types of AI representation, however, is the Artificial Neural Network (ANN), which is a computational model based on the biological neural networks of animal brains. There exists a plethora of methods to train an ANN using data sets and/or artificial evolutionary techniques such as Evolutionary Robotics.

When looking to design autonomously an AI capable of robust and adaptable behaviour, one must first choose from a variety of biologically inspired ANNs. While certain ANNs, such as Continuous-Time Recurrent Neural Networks (see Section 3.4.3) can theoretically approximate any dynamics, in practice it can be difficult to evolve artificially complex behaviours using these structures. Much of Evolutionary Robotics research focuses on further improving the evolvability and capabilities of the ANNs undergoing evolution by drawing inspiration from biology and neuroscience. The problem with this approach, however, is that we do not yet have a solid scientific understanding of the inner workings of biological neural networks. Furthermore, with greater biofidelity comes greater complexity, adding to the already high computational costs of ER algorithms while further obfuscating the inner workings of the evolved agents. Thus, we look to avoid models of neural networks altogether by instead representing and evolving AIs as *abstractions* of neural networks. We introduce Evolvable Mathematical Models (EMMs), a novel AI paradigm that uses artificial evolution to create state equations autonomously that map an agent's inputs to its outputs.

Our thesis statement is as follows:

> *Using powerful artificial evolutionary techniques, it is possible to evolve mathematical models that modify controller outputs based on current and previous agent/environment information in such a way as to demonstrate learning, adaptability and generalization capabilities similar to those seen in biological agents.*

The work presented here is heavily inspired by the groundbreaking work done by Dario Floreano and Stefano Nolfi [114]. Their work has demonstrated the capacity for Evolutionary Robotics to produce AIs with significant learning and generalization capabilities. Furthermore, they have simulated the emergence and evolution of interagent communication and demonstrated that ANN-based agents can be transferred successfully into robotic hardware. This work is also influenced by Kenneth Stanley's NEAT and CPPN/HyperNEAT algorithms, as they have shown that artificial evolution can produce remarkably lifelike patterns [137] and behaviour [28]. The freely available NEAT source code has been an important and influential resource. Finally, the implementation of our EMMs is based on the work on Genetic Programming and Symbolic Regression by J.R. Koza [85], Riccardo Poli, William B. Langdon and Nicholas Freitag McPhee [126], and Michael Schmidt and Hod Lipson [135].

The ultimate goals of this research are twofold. First, artificial lifeforms can be produced for scientific purposes, such as the study of their evolution and behaviour. This can then be used to inform the study of the evolution and behaviour of life on Earth. The second goal is to create robust AIs that can solve complex problems and adapt to novel scenarios. Applications include space exploration and hazardous work environments on Earth, as a strong AI could perform complex tasks requiring human-like intelligence without the requirements engendered by having to sustain biological life. Furthermore, AI can aid humanity in scientific discovery, a capability that is already being demonstrated by Schmidt and Lipson's Eureqa algorithm (see, e.g., [136]). Eventually, an AGI could help in the design of better AGIs, potentially producing a *beyond human* intelligence.

Part I of this thesis will begin with a literature review covering artificial evolutionary techniques, Artificial Neural Networks and Evolutionary Robotics. Our novel Evolvable Mathematical Models will then be presented and subsequently tested and compared to ANN-based algorithms on two common benchmarking tasks. Part II of this thesis arose out of the desire to demonstrate the capabilities of our paradigm to produce lifelike behaviour by evolving complex artificial organisms. For this task we chose to simulate the emergence and evolution of communication, as it is a notoriously hard problem with a wide gap between simulation results and observed behaviours in nature. Current results in the simulation of the emergence and evolution of interagent communication will first be reviewed. We will then present our simulation world, *NoiseWorld*, in which simulated robots live, reproduce, communicate and die. The emergence and evolution of complex communication schemes from initially noncommunicating EMM-based agents is observed, providing the first, albeit digital, fossil records for the evolution of communication. Furthermore, these evolved agents are embodied in e-puck robots, demonstrating that the evolved behaviours of our EMMs can be successfully ported from the simulation world into our physical reality.

Part I

Evolvable Mathematical Models

Chapter 2

Introduction

The first part of this thesis pertains to employing our novel approach, called Evolvable Mathematical Models, in autonomous tasks previously solved using ANN-based approaches.

The field of Evolutionary Robotics studies methods for automatically generating the artificial brains and/or morphologies of autonomous robots via artificial evolution [114, 50]. These artificial brains are the autonomous robots' controllers and are typically represented as Artificial Neural Networks (ANNs), a class of computer programs that are inspired by the neural networks of biological brains. In its simplest form, an ANN is composed of a collection of identical nodes, or "neurons," where each performs the same computation on its numerical inputs, yielding a numerical output. These nodes are connected via weighted directional edges. The behaviour of an ANN is fully described by the calculation performed by its nodes, how these nodes are interconnected, and the weights of these connections. Data arrive at input nodes, are propagated through the network, and finally reach the output nodes, whose numerical values are interpreted as behaviours.

When evolving controllers for robotic applications, the most obvious metric of success is how well the final controller solves the given task. However, researchers and developers are also very interested in robustness, adaptability and the ability to generalize. Evaluating controller fitness on hardware is usually infeasible owing to time and hardware constraints. Thus, the fitness of individuals, which is usually their ability to accomplish the specified task, is most often evaluated in simulation. This is one of the reasons why the ability for an evolved controller to generalize is so important, as it is very difficult to predict perfectly and simulate the environmental and hardware conditions that the evolved controller will eventually see during real-world applications. Thus, a good algorithm should be able to produce a controller that can be applied successfully in never-before-seen scenarios.

With these goals in mind, researchers have been looking to nature for inspiration as so far it is nature, and not human design, that has produced the most adaptable controllers. How-

ever, natural systems are extremely complex and poorly understood, and so initial research tried to distill the most important elements of the evolution and operation of modern lifeforms in an attempt to reproduce nature's successes. This has yielded the core idea behind Evolutionary Robotics: Use a Genetic Algorithm to evolve the weights of a fixed-structure Artificial Neural Network. Further research then worked to increase the biofidelity of this approach by adding in components inspired by neuroscience, such as neuroplasticity and neuromodulation (see Section 3). We propose a new paradigm that evolves *abstractions* of neural networks, thus allowing the programmer to safely ignore their inner physical workings. Humanity's best tool for abstraction is mathematics, therefore we present our Evolvable Mathematical Models (EMMs) as a novel paradigm in Artificial Intelligence. EMMs are mathematical equations that directly map numerical inputs to numerical outputs. Using only the four basic mathematical operators (addition, subtraction, multiplication and division[1]), these equations can approximate any analytic function. A function is analytic on an interval of the real line iff its Taylor series converges to the value of the function in question at each point on the interval.

The thesis statement of Part I is:

> *Evolvable Mathematical Models can be used to evolve robust, learning-capable controllers for autonomous agents.*

2.1 Objectives

The objectives of Part I of this thesis were to develop, program and test (in simulation) a novel type of controller. This evolvable controller should be highly robust and capable of learning.

2.2 Methodology

We attempt to evolve mathematical abstractions of natural neural systems. The method developed is based on Symbolic Regression algorithms in Genetic Programming (see Section 3.3). A controller is a series of mathematical equations, represented as an equation tree. There is one equation for each controller output, and equations can be composed of four basic mathematical operators (addition, subtraction, multiplication and division), as well as input and output variables and constants. A population of such equations is tested on a given task, and those controllers that perform well relative to others in the population are chosen as parents. Offspring equation trees are generated from the parent trees using genetic operators

[1]More sophisticated operations, such as sine and cosine, can be used as well, although the work presented here uses only the four basic mathematical operators.

such as various types of mutation that modify different aspects of the equation trees and two types of sexual recombination that combine elements from the trees of both parents to form a single offspring's equation trees. Extra equation trees and corresponding extra internal state variables can be added via mutation as well. A new population comprised entirely of the offspring of the previous generation are again tested on the given task. This generational loop continues for a fixed number of generations, and top solutions undergo further testing to determine which are the best overall.

These controllers are tested in two simulation experiments that require adaptability, robustness, generalizability and learning. The first is the most common benchmark in Evolutionary Robotics, the double-pole balancing without velocity information benchmark. Here, controllers are required to keep two poles of different sizes upright by applying a force to the cart which they are attached to, using only pole angle information (pole velocity information is withheld). A solution to this problem is defined as a controller that can balance both poles for 100,000 timesteps from a fixed initial state, and can balance both poles for 1,000 timesteps from at least 200 of 625 other predetermined initial states (this tests generalizability). Algorithms are compared on the number of simulation runs needed to find a solution (i.e., population size $\times$ number of generations) and on the ability of top solutions to generalize.

The second benchmark is the discrete Double-T Maze with homing. This task was chosen for its difficulty and learning requirements. It has previously only been solved using special artificial neural networks with connection weights that can change over the course of a simulation run. Agents must navigate a maze with four ends, three of which contain a low reward, with the fourth containing a high reward. Agents must return home after reaching a maze end. Successful controllers should forage for the high reward and then repeatedly return to the maze end containing the high reward once it is discovered. If the high reward is moved, a successful controller should return to foraging behaviour until the high reward location is rediscovered. This task requires that agents adapt their behaviours as conditions change throughout their lifetime, and thus requires a form of learning. We use this benchmark to see if our method can solve this difficult task, and thus whether or not our algorithm is capable of producing learning behaviours and solutions comparable to those evolved using ANN-based algorithms.

The two described tasks are typically used to compare and contrast Artificial Neural Network based approaches. In this case, we supplant these neural networks with evolvable mathematical models. This one algorithm can solve both of these problems, whereas previously these two different problem domains were solved using different simulated neural network structures. This also demonstrates that these evolved mathematical equations can

abstract a variety of artificial neural networks.

2.3 Outline of Part I

Chapter 3 gives a literature review encompassing the fields of Evolutionary Computation, Genetic Algorithms, Artificial Neural Networks (ANNs), and Evolutionary Robotics. Our novel paradigm, Evolvable Mathematical Models (EMMs), will be introduced and described in detail in Chapter 4. This chapter will also present two benchmarking experiments, including one requiring that agents develop learning capabilities. The performance of EMMs on these tasks will be compared to results from ANN-based experiments. The solutions found by the EMM approach will be compared to the solutions found using ANN-based approaches. Finally, some concluding remarks will be given in Chapter 5.

Chapter 3

Literature Review

The following literature review will present the algorithms and data structures that are most commonly used when applying artificial evolution to the development of artificial intelligence. A full review of the fields of evolutionary computation, artificial neural networks, and nonevolutionary training methods for ANNs are outside the scope of this thesis.

3.1 Evolutionary Computation and Genetic Algorithms

Genetic Algorithms are a type of Evolutionary Computation (EC), where computer simulations of Darwinian evolution are used to solve optimization problems [37]. The idea of simulating evolution in a computer dates back to Alan Turing [153], with some of the first algorithms being implemented by Nils Aall Barricelli soon after [11]. This class of algorithm did not become popular until the late 1970s and 1980s however, owing to a lack of cheap computational power. Since the digital revolution, EC has been used to solve a variety of NP-Hard problems, including the Travelling Salesman Problem [91] and the Knapsack Problem [27, 62] and has produced a myriad of human-competitive results [87].

Genetic Algorithms (GAs) are usually attributed to John Holland [71]. A typical GA proceeds as follows (adapted from [37, 126]):

1. Randomly generate a population of solutions.
2. Evaluate the "fitness" (solution quality) of each solution in the population.
3. Select "parent" solutions from the population using a method based on fitness values.
4. Create "offspring" solutions from selected parents using stochastic genetic operators.
5. Replace current population with offspring population.

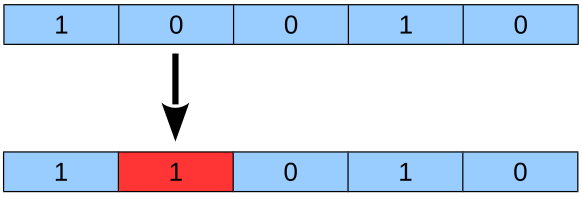


Figure 3.1: An example bit flip mutation for a binary genome containing 5 genes. The mutation point is chosen randomly.

6. Loop back to Step 2 unless stopping condition is met.

7. Return best (i.e., highest fitness) solution.

The key elements here are *selection* (Step 3), which allows for the algorithm to search the solution space around relatively high quality solutions, and the *mutation* and *crossover* operators (applied at Step 4), which allow for local and global jumps in the solution space, respectively.

There are several design decisions to be made when using a GA to solve a given problem. One needs a "genome" that can encode potential solutions. A fitness function is also required to give a numerical value representing how good a particular genome/solution is at solving the given problem. Furthermore, a selection method is needed to choose parents based on fitness values (see [56] for a comparison of popular selection methods). A variety of genetic operators are also required to produce offspring genomes from selected parents. Stochastic mutation operators are needed to produce and maintain variation within the population, while also allowing for incremental hill-climbing within the solution space. A stochastic crossover operation is required if sexual recombination (i.e., where two parent genomes are combined to form one or more offspring genomes) is desired.

Crossover is an important element of a GA, as it improves the rate of convergence to a solution and can help find better solutions overall by allowing for larger jumps through the search space [37]. The types of mutation and crossover operators that can be used depend on how the genome is constructed. For a fixed-length genome (i.e., a fixed number of genes), a mutation is usually accomplished by randomly selecting a gene and changing it to a new randomly selected value (see Figure 3.1 for an example), whereas crossover produces an offspring genome by taking one or more randomly chosen segments from the first parent's genome and combining it with the remaining segment(s) taken from the second parent's genome. An example is shown in Figure 3.2. A stopping condition needs to be provided by the experimenter to choose when to terminate the evolution loop. This usually comes in the form of a fixed number of generations, but could also incorporate solution quality or time since the fitness "high score" was usurped. Programmers must also choose a population size

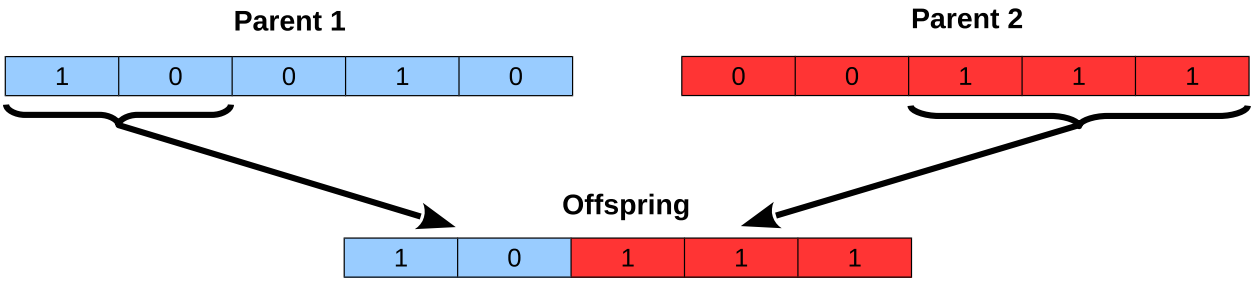


Figure 3.2: An example of a single-point crossover operation creating an offspring genome from two parent genomes. The crossover point is chosen randomly.

(with a trade-off between average solution quality and convergence speed) and mutation and crossover rates (which need to be tuned to the GA being used) [37].

3.2 Parallel Genetic Algorithms

Genetic algorithms are considered to be "embarrassingly parallel" [22], as it is straightforward to parallelize fitness function evaluations (which are the most computationally expensive part of the algorithm) to utilize the full power of multicore processors. Recently, researchers have been achieving massive speed-ups using off-the-shelf graphics processing units (GPUs) which have hundreds of on-board processors running in parallel (e.g., [65], [162] and [7]).

While these speed-ups are important, especially considering that GAs are often computationally expensive, parallel GAs can also provide improvements in solution quality over their serial counterparts. As GAs are stochastic algorithms, runs with the same parameter settings but different initial conditions can produce vastly different results. Experimenters must run a GA multiple times when looking for the best possible solution to their problem. Thus, the simplest way to parallelize a GA is to run several independent instances of the same GA with different initial populations simultaneously. However, if one introduces "migration" into this parallel paradigm, whereby genomes from one instance of the GA can move to a different instance, better solutions can be discovered [21]. This "Island Model" [30] allows for the parallel GA to search multiple regions of the solution space simultaneously, with the combination of migration and crossover allowing for the algorithm to reach new areas of the search space that a single population GA might never attain. Unfortunately, Island Models add more parameters to the already parameter-heavy GA, such as number of islands, migration rate and size of migration groups [139]. Furthermore, there are a multitude of different migration schemes (i.e., topologies) to choose from, although their influence, if any, on final result quality remains unclear [47]. Recent work suggests that different optimization algorithms prefer different topologies [134].

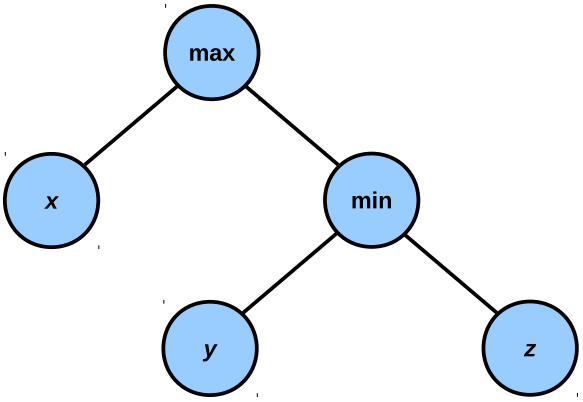


Figure 3.3: An example genome of a GP program. This tree represents the program `max(x,min(y,z))`.

Finally, more complex parallel GAs have emerged that utilize the concept of "species" to further improve results, e.g., [64, 62].

3.3 Genetic Programming and Symbolic Regression

Genetic programming (GP) is the application of Genetic Algorithms to the design of computer programs [85]. Potential solutions (i.e., genomes) are usually stored as tree structures, with nonterminal nodes being selected from a user-defined set of operators and terminal nodes being selected from a user-defined set of operands (see Figure 3.3 for an example). These programs are executed and given a fitness value based on how well they do on a user-specified task. The manner in which mutations and crossover are implemented in GP will now be described (adapted from [126, 60]). Possible mutations fall into two main categories: point mutations and subtree mutations. Point mutations affect a single node in the genome tree. If the mutation is applied to a terminal node, the current operand is replaced with a new, randomly selected operand from the user-defined operand set. If a point mutation is being applied to a nonterminal node, the current operator is replaced with a new, randomly selected operator from the user-defined operator set *with the same arity*[1] *as the current one.*

A subtree mutation selects a node at random on the current tree and replaces it (and all its subtrees) with a new, randomly generated subtree. Random subtrees are usually generated using the ramped half-and-half method, which is a combination of two methods, the "full" and "grow" methods. For both methods, a maximum depth (i.e., the maximum number of

[1]By "arity" we mean the number of arguments that an operator accepts. For example, the "sine" operator has an arity of 1, as it operates on a single argument. Therefore, if a point mutation is changing a node that is currently a sine operation, it could mutate to other operators of arity 1, such as cosine or absolute value, but not operators of other arities, such as addition, which has an arity of 2.

edges that need to be traversed to reach a node, starting from the root node) is specified. In the "full" method, nonterminal (i.e., operator) nodes are randomly generated until the maximum depth is reached. At the maximum depth, only terminal (i.e., operand) nodes are created. In the "grow" method, as in the "full" method, only terminal nodes are created at the maximum depth. The difference is that before the maximum depth is reached, randomly generated nodes can be either terminal or nonterminal nodes, allowing for a wider range of potential tree shapes. The ramped half-and-half method chooses to create a random subtree using either the "full" or "grow" method with equal probability. The ramped half-and-half method is also usually used to generate initial populations.

Crossover between two parent genome trees is implemented in a similar manner to subtree mutation. An offspring genome tree is taken as a clone of one parent, with a randomly selected subtree from the second parent overwriting a randomly selected node on the offspring genome (see Figure 3.4 for an example). Genetic programming has a wide range of applications, including image processing [74], finance and economics [24, 81], data mining [138] and even engineering design, e.g., an antenna for deployment on NASA's Space Technology 5 Mission [93].

One of the most common applications of GP is Symbolic Regression (SR), to which a lot of the aforementioned examples can be reduced. In SR, the GP algorithm is tasked with finding a mathematical function that best fits a data set. At its most basic, the set of operators could be addition, subtraction, multiplication and division, although a large variety of other mathematical operators can be used. The set of operands would include constants and user-selected independent input variables. The fitness function is usually calculated using absolute difference or squared error between known data values at certain input vectors and the output values of the function tree in question when evaluated using these same input vectors.

The best example of SR is the Eureqa algorithm which can automatically discover phenomenological rules from experimental data [135, 136]. For example, Eureqa was able to discover the Hamiltonian of a double pendulum using only data recorded from the pendulum's movements. Eureqa has also been used to model a robot's dynamics autonomously [110].

3.4 Artificial Neural Networks

Artificial Neural Networks (ANNs) are computer programs loosely based on biological neural networks (brains). An ANN is a collection of connected nodes (analogous to neurons in the brain), with each node performing the same calculation on its numerical inputs, yielding

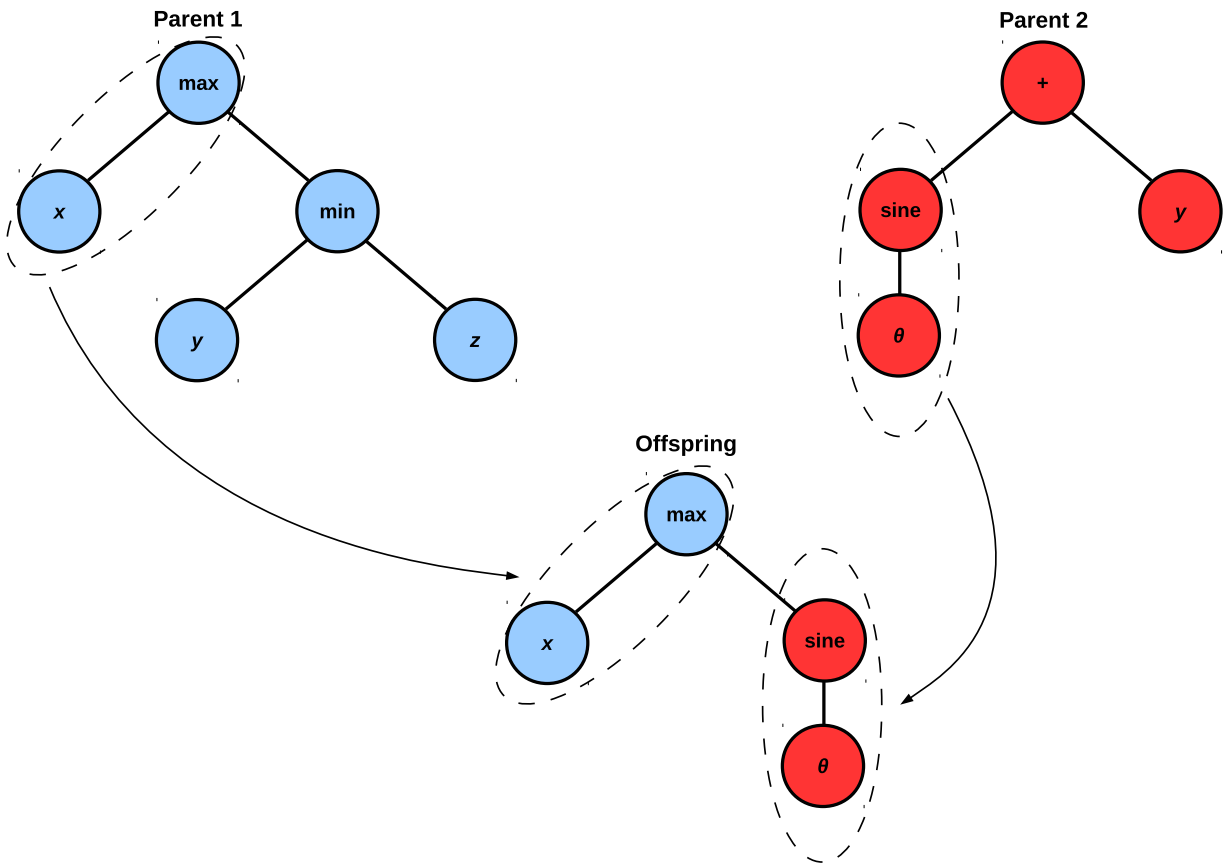


Figure 3.4: An example crossover operation to produce an offspring GP tree. Parent 1 is the same tree as in Figure 3.3 above. Parent 2 represents the program `sin(θ)+y`. The offspring is a tree randomly created by copying the tree of Parent 1 and replacing a randomly chosen subtree on this copy with a copy of a randomly selected subtree from Parent 2. The resulting program is `max(x,sin(θ))`. Note that subtree mutations occur in a similar manner, except that the randomly selected subtree from Parent 2 would instead be a randomly generated subtree.

a numerical output value. A commonly used node calculation (known as the activation function) is the logistic sigmoid:

$$\phi(u_j) = \frac{1}{1 + e^{-u_j}} = v_j \tag{3.1}$$

where $u_j = \sum_{i=1}^{n} v_i w_{ij}$, with n being the number of nodes in the network, v_i being the output of the i^{th} node and w_{ij} being the weight of the connection from node i to node j (set to 0 if node i is not connected to node j). An example ANN is shown in Figure 3.5.

Data enter an ANN via input nodes, propagate through the network via the directional weighted connections and nodes, eventually reaching one or more output nodes, whose numerical values are interpreted as the output of the ANN. Thus the behaviour of an ANN

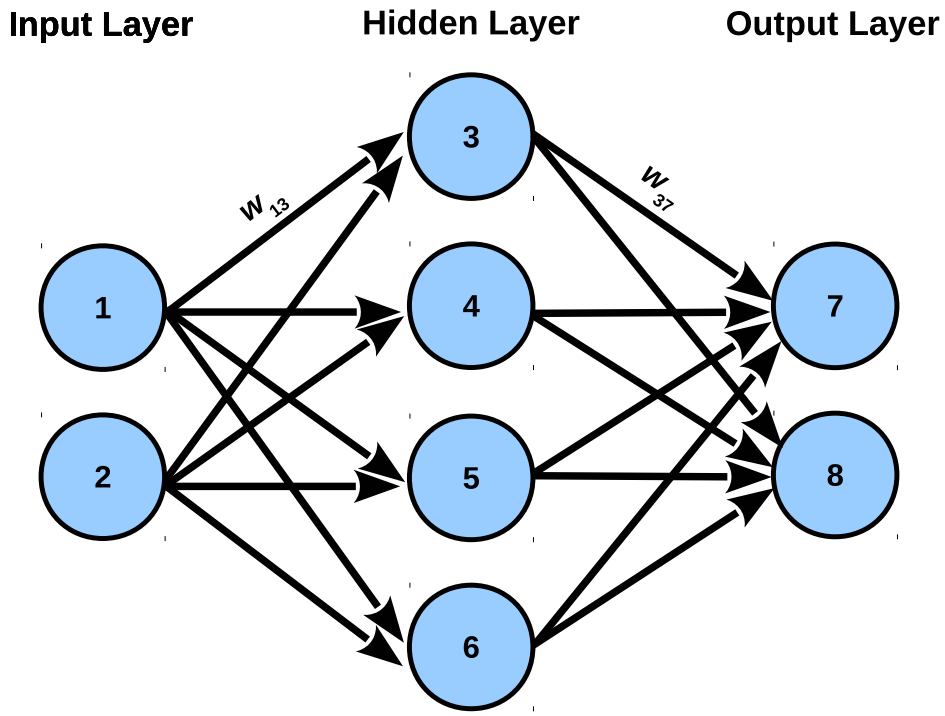


Figure 3.5: An example of a fully connected feedforward Artificial Neural Network. This network has two input nodes, four hidden nodes and two output nodes. Each connection from a node i to a node j has an associated connection weight w_{ij}. Two such connection weights are labelled in this diagram.

is fully defined by the calculation the nodes perform, the topology of the network, and the weights of the connections. An ANN is considered to be "fully connected" when all possible network connections have nonzero weights.

3.4.1 Feedforward Neural Networks

The simplest type of ANN is the feedforward neural network. Nodes are organized into an input and output layer, with zero or more "hidden" layers in between. All network connections run between layers in the forward direction (i.e., from the input layer towards the output layer with no connections allowed between nodes on the same layer). Figure 3.5 shows a simple example of a fully connected feedforward neural network.

To evaluate this type of network for a given input vector, one propagates the input values through the network's node layer(s) sequentially, yielding an output vector.

These types of networks can only be reactive. They have no memory and thus will always produce the same output vector for a given input vector. Furthermore, if the set of possible input vectors is finite, a feedforward neural network can be reduced to a fixed set of *if...then...* rules [8].

Feedforward neural networks have been shown to be capable of approximating any Borel measurable function from one finite-dimensional space to another with any degree of accuracy [73]. Thus, ANNs are a class of universal approximator.

3.4.2 Recurrent Neural Networks

Recurrent neural networks are neural networks that allow connections between any two nodes regardless of their locations in the network, while also allowing for connections from a node to itself. In a fully connected recurrent neural network, each node's output is an input to every node in the network, including itself. Recurrent connections allow the network to have an internal state and thus a form of memory, yielding a much more diverse set of potential behaviours than the feedforward networks.

These networks are evaluated in discrete timesteps, with a node performing its computation at the current timestep using the results (modified by connection weights) of the node computations done at the previous timestep. Nodes can also incorporate current data input values if there is a direct connection between input nodes and the node in question.

3.4.3 Continuous-Time Recurrent Neural Networks

So far, we have discussed examples of ANNs comprised of neurons (nodes) with static functions (e.g., logistic sigmoid). Continuous-Time Recurrent Neural Networks (CTRNNs) [12] are an example of dynamic neuron models. Here, the output v_i of a node i is defined as (adapted from [49])

$$\frac{dv_i(t)}{dt} = \frac{1}{\tau_i}(-v_i(t) + \sum_{j=1}^{n} w_{ij}\phi(v_j(t)) + u_i) \tag{3.2}$$

where τ_i is a time constant, n is the number of neurons in the network, and u_i is an external input to the system, e.g., from a sensor. The neurons in this paradigm resemble biological neurons more closely than in the static neuron cases, as the firings of biological neurons are modelled as a time-series [70] instead of as discrete events.

An important trait of CTRNNs is that they can theoretically approximate any dynamics with arbitrary precision [54].

3.4.4 Spiking Neural Networks

Further biological fidelity is added to the CTRNN model by using spiking neurons. The "Integrate and Fire" neuron is the simplest form of spiking neuron. It is an extension of the neurons of a CTRNN where the output value of (3.2) is fed into a threshold function. If the

output is above the user-defined threshold a spike is emitted, otherwise there is no output [49]. To use a Spiking Neural Network, one must translate numerical inputs into spikes (e.g., using spike frequency) and then translate the output neurons' spikes into usable numerical values [51].

3.5 Evolutionary Robotics

The field of Evolutionary Robotics (ER) uses genetic algorithms to design autonomous controllers (typically represented as Artificial Neural Networks), and sometimes robot morphologies, for experimenter-defined tasks requiring physical or simulated autonomous robots [114]. In its simplest form, an ER algorithm is used to tune the weights of a fixed structure feedforward ANN for a given task. In this case, the genome will be of fixed length, with one "gene" location for each (usually real-valued) weight of the ANN. The size and topology of the ANN will be chosen by the programmer before running the ER algorithm. A "population" will consist of a collection of genomes, each with their own values for the weights of the ANN. To give a fitness value to each genome, a computer simulation of the required task is often devised, as evaluating each genome in hardware could be prohibitively time-consuming and poor solutions could damage the hardware.

As an example, consider a simple task where we need a two-wheeled robot to drive along a straight line that the robot can detect. The ANN in this case would have one input (a binary input that indicates whether or not the robot is on the line), a layer of hidden neurons, and two outputs that encode the two motor speeds. The simulation would then continually have to evaluate the ANN (while providing the correct input value corresponding to whether the simulated robot is on the line) and simulate the two-wheeled robot's movements based on its motor speeds. One could then assign a fitness value to a genome based on how well that genome's ANN drives the simulated robot, e.g., the fitness could be the total distance travelled while on the line. Higher fitness genomes do a better job of driving the robot along the line and will be more reproductively successful than those genomes that miss the line. After generations of simulated evolution, a successful ANN will be produced for this task (assuming a good selection of ANN topology and algorithm parameters). The next step would be to test this evolved ANN on a physical robot to see if the evolved solution is transferrable to the real world.

In ER algorithms, as in Darwinian evolution, selection operates on variation. Obviously if all genomes have the same fitness, the algorithm will have no way to differentiate between good and bad solutions and thus the generational loop will be unable to tune the ANNs for the task. This issue occurs most often at the beginning of the simulation, when all the ANNs

have random weights and thus all perform very poorly on the given task. This is known as the "Bootstrap Problem" and it requires careful consideration and special techniques to overcome, especially when attempting to evolve for complex tasks [111].

While the approach outlined above works well for simple tasks, it does not scale well to more complex tasks requiring larger neural networks. For difficult tasks, there is no easy way to choose an appropriate network topology. Furthermore, with complex ANNs, the number of weights and thus the size of the genomes can be quite large, creating very large search spaces that a GA might have difficulty searching efficiently. Finally, there is a trade-off between computational efficiency and simulation fidelity. Regardless of how well the simulation represents reality, there are always unexpectedly relevant properties of the real world that are omitted from the simulation. Therefore, it can be very difficult to successfully transfer an ANN that operates well in the simulation environment to the real world. A variety of algorithms have been developed to attempt to alleviate the aforementioned issues.

3.5.1 AGE

Analog Genetic Encoding (AGE) is an encoding method based on biological gene regulatory networks [101]. By using AGE to represent ANNs, one can evolve both network topology and weights simultaneously [44]. A genome consists of a variable-length string of characters. Certain combinations of characters, called device tokens, signal the beginning of a section of the genome defining a neuron, while one or more terminal tokens denotes the end of such a section. The characters between a device token and the final terminal token are used to determine which other neurons this neuron will connect to, as well as the connection weights. Characters that come after the final terminal token but before the next device token are ignored, as are the characters that appear after a device token that has no corresponding terminal token (see Figure 3.6 for an example). Thus, any character string is a valid genome. This allows for a variety of mutations, including permutations of random substrings. A mutation to add in a new device token is also implemented, which when combined with the fact that the genome strings are variable-length gives AGE the capability to produce ANNs of any shape and size.

3.5.2 NEAT

Another method that allows for the simultaneous evolution of both network structure and connection weights is NeuroEvolution of Augmenting Topologies (NEAT) [148]. As with AGE, NEAT uses a variable-length genome. There are two components to a genome, a "node" component and a "connection" component (see Figure 3.7 for an example). Both

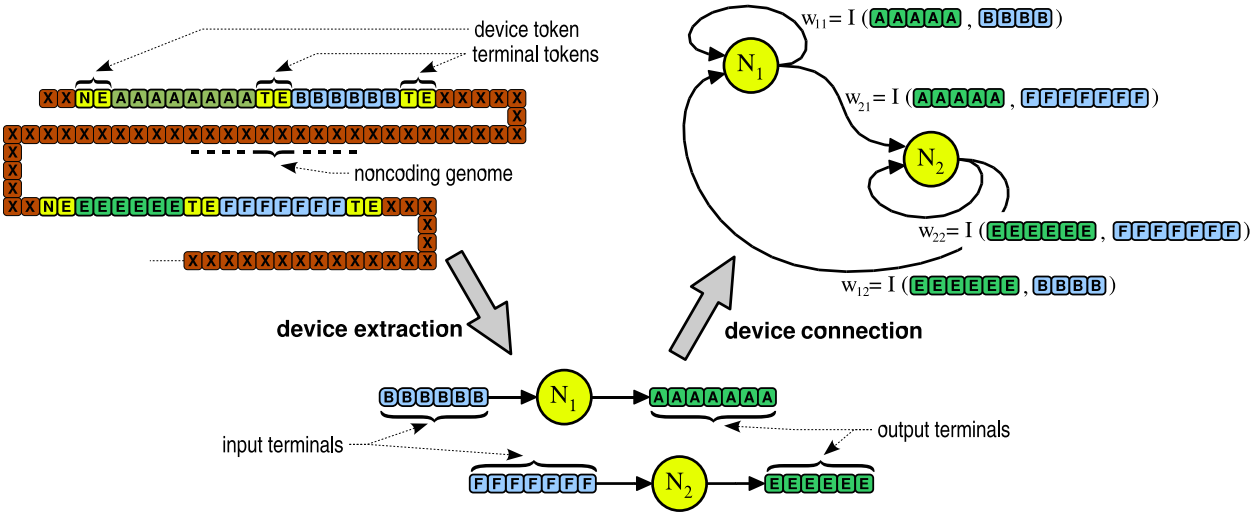


Figure 3.6: An example encoding of a two neuron neural network using AGE; from [44].

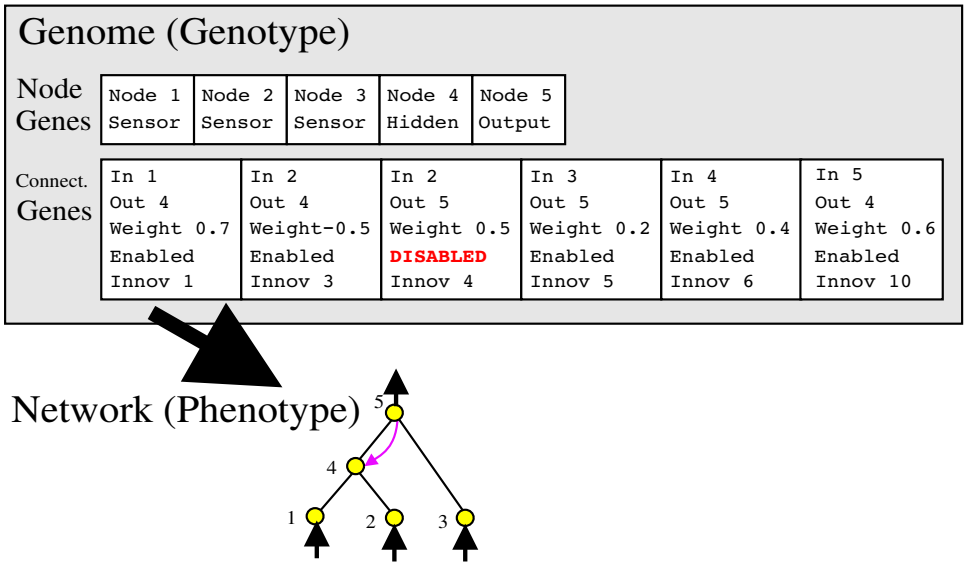


Figure 3.7: An example encoding of a neural network using NEAT; from [148].

of these are variable-length, as the algorithm has mutations that insert new nodes into the genome (and thus into the ANN that it represents) as well as mutations that can add new connections between existing nodes in the network. Connections have associated weights which are also subject to mutation. NEAT also has the capability to produce ANNs of any shape and size.

3.5.3 HyperNEAT

In the methods described thus far, each and every weight of the ANN is encoded in the genome. This becomes problematic when evolving large networks, as a large number of con-

nections will require large genomes and thus very large search spaces. As was previously mentioned, this can negatively impact the GAs performance. An alternative class of encodings, called developmental or generative encodings, uses genomes that contain "programs" for building ANNs. This allows for compact representation of large genomes and is therefore much more scalable. To construct an ANN, one executes the program encoded in the genome. This allows for the reuse of previously evolved ANN substructures, as a simple "loop back" gene on the genome can cause the program to reexecute an earlier section, thus regenerating that section's ANN substructure [72]. Biological systems are a great example of developmental encodings, as each individual neuron of an animal's brain is not encoded in its DNA. Instead, DNA encodes a program whose execution (through the developmental process) produces biological neural networks.

A popular example of a generative encoding is HyperNEAT [147]. In this approach, the original NEAT algorithm is used to evolve a Compositional Pattern Producing Network (CPPN) [145], which is then used to generate the weights of an ANN. CPPNs are basically neural networks where each node is one of a set of user-defined functions (e.g., $f(x) = x$, $g(x) = |x|$, $h(x) = \sin x$, $j(x) = x \bmod 1$, etc., see Figure 3.8 for an example). As in ANNs, the connection weights in CPPNs multiply the output values that are travelling through the connection in question and if multiple connections feed into the same node, that node's function is applied to the sum of the incoming values. In its typical form, HyperNEAT generates a neural network from a CPPN that has four input nodes, x_1, y_1, x_2 and y_2. The nodes of the ANN are arranged on a 2D grid called the substrate. The experimenter defines which nodes are the input and output nodes. Each connection weight in the ANN is then determined by inputting the substrate coordinates of the starting node (x_1, y_1) and destination node (x_2, y_2) into the CPPN. By propagating these four values through the CPPN, a single output value is generated that corresponds to the connection weight between those two nodes. Experimenters must define the substrate structure beforehand, as it does not have to be a 2D grid. Furthermore, the placement of the input and output nodes within the substrate is crucial, as the performance of HyperNEAT is significantly affected by geometric representation [29]. Users must also determine the placement of hidden nodes in the substrate, although Evolvable-Substrate HyperNEAT can automate this task [130]. Finally, the set of user-defined functions available for use in the CPPN must also be defined *a priori*.

HyperNEAT has been successful at evolving large ANNs for difficult tasks involving regularity, such as gaits for quadrupeds [28], simulated driving tasks [42], and autonomous players for the game of Go [55]. A slightly modified version of HyperNEAT can be used to evolve heterogeneous multiagent teams on a single genome [33, 32].

HyperGP [15] is an alternative approach to HyperNEAT that replaces the CPPN with a

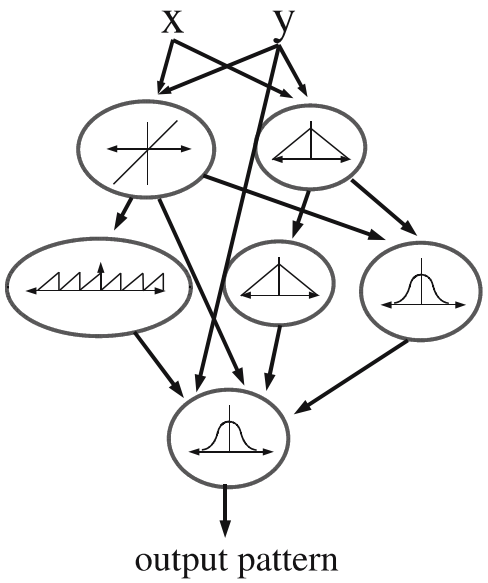


Figure 3.8: A high-level view of a CPPN. The values that traverse the connections are multiplied by the associated connection weight. If multiple connections feed into the same node, that node's function is applied to the sum of the incoming values; from [145].

mathematical function evolved using Genetic Programming.

3.6 Learning

In all of the algorithms discussed thus far, when evaluating an ANN, either in simulation or hardware, the ANN's connection weights are fixed. It is only when generating offspring genomes for the next generation of the GA that genetic operators such as mutation can modify the ANN's weights. This is contrary to biological neural networks where "neuroplasticity" allows for connections between neurons to change during the lifespan of an organism [124]. Neuroplasticity (or "plasticity" for short) is a mechanism that plays a role in an animal's ability to learn, modifying the way it reacts to certain inputs from the environment. For example, it was found that learning to juggle induced changes in the brains of human participants that were detectable through brain scans [41].

When using ER algorithms to evolve controllers for real-world automation tasks, it is difficult to transfer the controller from simulation to real-world hardware. This is due to the fact that no matter how high the fidelity of the simulation, there will always be environmental and hardware details that are not simulated. The real world is notoriously unpredictable. Without the capability for online adaptation, an evolved controller that fails once in a novel scenario will always fail in this scenario. To incorporate successful behaviours for this situation into a controller, the ER algorithm will have to be rerun to reevolve controllers in a

simulation environment that incorporates this novel scenario. This is less than ideal, especially considering the computational complexity of a typical ER algorithm. Therefore, a big component of ER research is the exploration of methods to incorporate "learning" behaviours into evolved controllers.

One of the first attempts to evolve learning was by Stefano Nolfi and Domenico Parisi in [116], where two simple neural networks with no hidden units were evolved simultaneously. The first network was tasked with controlling the motors of a robot, while the second network was considered a teaching network. The outputs of this teaching network were used to modify the network weights of the first network through backpropagation. Thus, the weights of the first network were plastic. The task was for a robot to forage through a 60×20 cm area to find a randomly placed circular target area with a diameter of 2 cm. The robot, which had proximity sensors on all sides, operated in one of two environments. In the first, its proximity sensors were activated when the robot was within 1 cm of a wall, while in the other scenario the proximity sensors were activated when the robot was within 6 cm of a wall. Thus, to search the experimental area fully, the robot had to learn which environment it currently inhabits and adapt its behaviour accordingly.

3.6.1 Hebbian Learning

The most common type of learning used is Hebbian learning, which is based on how learning is thought to occur in biology [68], although some aspects of Hebbian theory are oversimplified (see, e.g., [106]). With this type of learning, rules for adjusting the connection weights of the ANN are evolved; however, initial weights need not be (they can instead be set to small random values at the beginning of a network's lifetime, as in [48]). These evolved Hebbian rules adjust the weights of connections based on the activity level of the nodes at either end of the connection in question.

Hebbian learning was first introduced to Evolutionary Robotics in [48], where the GA could select one of four learning rules Δw_{ij} for each network connection weight w_{ij}. At each timestep t of an ANN's lifespan, each of its connection weights is updated using its associated learning rule:

$$w_{ij}^{t+\Delta t} = w_{ij}^{t} + \eta \Delta w_{ij}^{t+\Delta t} \tag{3.3}$$

where $0 \leq \eta \leq 1$ is the evolvable learning rate and Δw_{ij} is one of four possible learning rules. The "plain Hebb rule" is defined as

$$\Delta w_{ij}^{t+\Delta t} = (1 - w_{ij}^{t}) v_i^t v_j^t \tag{3.4}$$

where v_i^t is the output of node i at the current timestep. The other three learning rules are more complex but they are all functions of only the current weight of the connection w_{ij}^t and the output values of the two nodes in question (i.e., v_i^t and v_j^t). Agent connection weights were not evolved, instead they were set to small random values at the beginning of an agent's lifetime. Top evolved agents were able to learn a wall-following behaviour within their lifetimes to allow them to solve a simple navigation task.

The experiments in [53] used Hebbian learning to evolve controllers on miniature Khepera robots (evolution occurred sequentially on hardware instead of in simulation). Agents had to drive to an area marked in black to activate a light at the opposite end of the arena, and were then rewarded for the amount of time spent underneath this activated light source. It was shown that agents with plastic connection weights significantly outperformed their fixed-weight counterparts. Furthermore, top evolved plastic agents were successfully transferred to a different robot, the Koala robot, whereas the top fixed-weight agents performed much worse on the different hardware. Another experiment demonstrated that when evolution could choose between fixed and plastic connection weights, artificial evolution systematically selected plasticity.

In [146], the NEAT algorithm was modified to allow for plastic connections, as well as the previously implemented fixed-weight connections. Here, the four learning rules from [48] are combined into a single learning rule:

$$\Delta w_{ij}^{t+\Delta t} = \eta_1 \left(W - w_{ij}^t\right) v_i^t v_j^t + \eta_2 W v_i^t \left(v_j^t - 1.0\right) \tag{3.5}$$

where W is the maximum weight magnitude in the network. The experiment used to demonstrate this algorithm was a foraging task where a mobile agent gains fitness by finding food, but loses fitness if it finds poison. Whether or not the environment is populated with food or poison is determined at the start of a simulation run. Agents could feel pleasure and pain and thus must evolve to use this information to determine if their environment has food or poison, and thus whether or not to forage. While NEAT with learning rules was able to solve this task, it is interesting to note that a relatively simple fixed-weight recurrent neural network could also be evolved to solve the task [146].

HyperNEAT can also be modified to allow for plasticity [131]. Three different possible learning rules were compared on a T Maze problem where an agent must visit one of two maze ends. One of the maze ends contains a high reward while the other contains a low reward (T Mazes are described in detail in Section 4.3.2). Agents must try to maximize the number of times they reach the large reward, which is moved once over the course of the series of trials. Successful agents were able to learn where the large reward was located and

then repeatedly visit that part of the maze. They were also able to change their behaviours when the reward was moved.

Hebbian-like learning has also been implemented in evolutionary robotics experiments using spiking neural networks, e.g., [40, 75].

3.6.2 Neuromodulation

In all of the learning paradigms described above, connection weights are adjusted at every timestep throughout the lifetime of an agent. This differs from biological systems which are theorized to use "neuromodulation" to control and stabilize learning [9].

The AGE algorithm was modified to allow for Hebbian learning which could be enabled and disabled via neuromodulatory signals [143, 43, 142]. In this approach, a generalized Hebbian rule from [113] is modified to include a modulatory signal m^t:

$$\Delta w_{ij}^{t+\Delta t} = m^t \eta \left(A v_i^t v_j^t + B v_i^t + C v_j^t + D \right) \tag{3.6}$$

where A, B, C and D are evolvable parameters that determine the importance of the different types of Hebbian learning and $0 \leq m^t \leq 1$ is the current strength of the signal produced by one or more special modulatory neurons. These modulatory neurons operate in a manner similar to regular neurons, taking inputs from previous nodes through network connections and generating their output value m^t using an activation function. AGE with neuromodulated Hebbian learning has been used to solve a single and Double-T Maze learning problem (see Section 4.3.2 and Figure 4.6 for more details on these problem domains), outperforming fixed-weight networks in all cases, and outperforming learning networks without neuromodulation in the Double-T Maze experiment [142]. It should be noted that a few of the fixed-weight evolutionary runs achieved good results in the Single-T Maze, displaying learning-like behaviours with the help of recurrent connections, although fixed-weight networks were unable to solve the Double-T Maze. Furthermore, several of the runs with unmodulated learning were successful at the Double-T Maze task.

3.6.3 Learning in Fixed Weight Neural Networks

As was already noted in several of the experiments described above (e.g., [146, 142]), fixed-weight recurrent neural networks seem to be able to accomplish certain tasks requiring learning, as the recurrent neural connections can act as a sort of memory.

The work in [23] demonstrates how a fixed-weight Locally Recurrent Neural Network (LRNN) can outperform a fully connected recurrent neural network in a dynamic simulation

Figure 3.9: A real-world setup of the simulation world used for the experiments in [78]; from [78].

environment. Simulated agents must visit a food location, then their nest, then a water location, then their nest again. Evolved agents with LRNN structures (basically modular versions of fully connected recurrent neural networks) could adapt successfully to changing food and water locations, finding better solutions than the fully recurrent neural networks. Strictly feedforward neural networks could not solve the task, owing to a lack of memory capabilities. Top LRNN controllers were successfully transferred to robotic hardware.

CTRNNs are also capable of demonstrating learning behaviour. In [161], Yamauchi and Beer were able to evolve CTRNNs that could learn which three-bit sequence to output based on an external reinforcement signal. In [78], CTRNNs are compared to fully recurrent neural networks with plasticity (as in [48]). It was found that the fixed-weight CTRNNs could perform just as well, if not better, than the neural networks with plastic weights on a simple reinforcement learning task. Simulated robots with infrared sensors needed to discover and remain in a goal zone. The goal zone was randomly placed either under a light bulb or a black stripe and was moved once per simulation run (see Figure 3.9). In a second experiment that was the simulation counterpart of the evolution on hardware experiments in [53], simulated robots had to first reach a target zone to activate a light, and then find and remain near this activated light. CTRNNs seemed to perform better on this task, although the networks with plasticity could also solve it. The authors claimed that the difference in performance was owing to the random initialization of weights in the plastic network case, requiring several timesteps to reach beneficial values. It is interesting that the best CTRNNs that were evolved in simulation on this second task could not solve the task when transferred to real robotic hardware, while the plastic neural networks were successfully able to adjust to the real world implementation. However, a method of evolving hardware-ready CTRNNs was proposed in [14]. It was found that by taking an evolved solution and then further evolving with 10% sensor noise, motor noise and variable starting positions and orientations, the final best evolved CTRNN could be successfully transferred to robotic hardware. The experiments in [152] also compared CTRNNs to fully recurrent neural networks with plasticity, although

the problem was more challenging than the one in [78]. In this experiment, robots must learn which direction of a continuous grayscale gradient (positive or negative) to follow towards the goal zone, while in [78] the robot had to decide between driving towards or away from a discrete (on/off) light source. Out of 20 runs, the fully recurrent neural networks with plasticity never found a solution, while the CTRNNs were successful in 14 of their 20 evolutionary runs.

Further experiments requiring learning in a continuous domain were performed in [77]. Here, CTRNNs that can learn to perform a reward-generating behaviour at a given "temperature" value were successfully evolved. Agents could learn and relearn to perform this behaviour for a variety of different temperatures. "Incremental evolution" [57] was used to produce these networks, where populations of agents are evolved on increasingly difficult problems until they can accomplish the final goal task.

Other experiments in [14] have shown that CTRNNs are capable of solving a Single-T Maze problem with varying reward locations. However, the evolutionary algorithm failed to find a solution to the Double-T Maze problem using CTRNNs. It should be noted that the fitness function in this case only rewarded agents if they reached the correct goal zone, whereas in the Double-T Maze experiment in [142] agents were penalized for hitting walls and received at least some reward for reaching any corner of the maze, regardless of whether or not it was the "goal" corner. Therefore, the lack of success for CTRNNs on the Double-T Maze problem may be an issue with the fitness function in the experiments instead of an issue with the neural-network paradigm being used. The experiments in [142] were simplified in a variety of other ways as well, compared with those in [14]. The most notable of these simplifications is that agents in [14] control the simulated robot's two actuators directly, requiring two outputs and the use of proximity sensors to avoid walls. In [142], however, agents simply control the discretized orientation of the simulated robot $(0, +\frac{\pi}{2}, -\frac{\pi}{2})$ requiring a single output and no wall sensors.

3.7 Genetic Programming for Robot Control

Genetic Programming can be applied to Evolutionary Robotics to evolve autonomous controllers as computer programs. Koza and Rice were the first to attempt such an experiment [88]. Controllers were represented as trees containing sensor inputs and four preprogrammed macros, such as "if-less-than-or-equal." Programs that could control a simulated robot to find a box in an irregularly shaped world and push it to a wall from four different starting configurations were evolved.

Nordin and Banzhaf used a linear implementation of GP to evolve machine code to control

a Khepera robot [117, 118]. Each 32-bit node contained a machine code instruction, such as `a=b+c`, which adds the values from registers `b` and `c` and stores the result in register `a`. The types of operations that could be employed were addition, subtraction, multiplication, `or`, `xor`, `and`, and left and right bit shift operations. Integers in the range [0,8192] could also be used. The evolutionary process occurred on a real robot and machine code programs were evolved for object avoidance and object following. This work was recently extended by Burbidge, Walker and Wilson who evolved machine code using Grammatical Evolution [16, 17]. Here, the agent genomes are binary strings that are mapped to their machine code phenotypes via a prespecified generative grammar. Simulated Khepera robots are tasked with driving towards a light source while avoiding obstacles.

GP has also been used to evolve competitors for the RoboCup robotic soccer tournament. Andre and Teller evolved team "Darwin United" using GP with a variety of possible operations, including basic mathematical operators, reading and writing to memory locations and executing a variety of programmer-designed subroutines [5]. Adorni, Cagnoni, and Mordonini used a simple GP approach to evolve robot goalie behaviours [3]. Strong simulation results were obtained by evolving controllers using only addition, subtraction, multiplication, division and sine and cosine operators. Agents had access to the position of the ball relative to the goalie's current position, as well as the current speed and direction of the ball (or, in a second experiment, the ball's position at the previous timestep) and evolvable constants. This work is the most similar to our Evolvable Mathematical Models paradigm presented below, as it is evolving mathematical equations as trees for robot control. However, these experiments were performed on a relatively simple task (especially considering the amount of information provided to the controller), had only one equation tree/agent output, used sine and cosine, and did not allow for extra state variables/equation trees to be evolved.

Chapter 4

Evolvable Mathematical Models

4.1 Introduction

We develop a new Artificial Intelligence paradigm with a heavy focus on applicability. Artificially evolved agents should be capable of the complex learning behaviours demonstrated in their ANN counterparts, as adaptability is an essential property of both autonomous robots and complex lifeforms. Furthermore, agents that have been evolved in simulation should be readily transferrable to physical robotic hardware. Advanced ANNs often act as black boxes, as their internal workings are complex and difficult to fully comprehend. An autonomous controller meant for real-world deployment should be readily examinable and amenable to mathematical analysis. Finally, we hope to develop an AI paradigm that is highly reusable, in the sense that it should be capable of being applied to a wide variety of tasks with little to no tuning/reworking.

Our new Evolvable Mathematical Model (EMM) paradigm focuses on evolving mathematical equations as controllers for autonomous agents. The idea of evolving mathematical equations originated with Symbolic Regression (see Section 3.3), which has been used since the early 1990s to discover automatically mathematical functions that accurately model data sets. In Symbolic Regression, candidate functions are tested on fixed training data sets composed of input and corresponding output values, with fitness being assigned based on how closely a function's outputs match the training outputs when evaluated using the training inputs. In the EMM paradigm, however, candidate functions are used to control autonomous agents. Therefore, the inputs to the EMM vary depending on the (potentially variable) properties of the agent and the (potentially variable) environment in which it finds itself. Furthermore, an EMM's outputs can influence its inputs, as its outputs influence properties of its agent and/or environment, thus modifying the inputs it receives via its agent's sensors. Instead of comparing an EMM's outputs to a fixed training data set, fitness is assigned based

on how well an EMM's agent performs on a given task. Complex behaviours can even emerge from the EMM paradigm without any explicit fitness function whatsoever (see Part II).

Despite a focus on applicability, biological plausibility and scientific benefits have not been abandoned. This new paradigm can produce complex artificially evolved artificial intelligences. By simulating the evolution of such intelligent behaviour, one can examine digital snapshots over time to lend evidence to, as well as inspire the development of, theories on the evolution of these behaviours in nature (see Chapter 9). Furthermore, while experiments run within this paradigm cannot contribute to the understanding of the physical inner workings of biological neural networks, they can contribute to understanding the evolution, organization and behaviours of a brain.

4.2 The Algorithm

Our Evolvable Mathematical Models algorithm is based on Symbolic Regression in Genetic Programming (see Section 3.3) and is similar to the algorithm presented in [135]. An earlier version of this algorithm was presented in [60] and implemented in [61]. The core idea is that one can use mathematical equations as a mapping from an agent's inputs to its outputs.

An EMM-based agent is represented as a system of equations, with one equation for each of the N experimenter-defined outputs v_i in the simulation. Additional "extra" equations that modify a corresponding extra state variable can be added through an "add equation" mutation. An agent's N' extra equations, $0 \leq N' < \infty$, do not have associated agent outputs; however, they modify agent outputs indirectly via the incorporation of their extra state variable into those equations that affect agent outputs directly. These extra equations and the mutations that create them are similar to Koza's Automatically Defined Functions and Architecture-Altering Operations [86], respectively. An agent is fully specified by its system of equations:

$$\mathbf{v}^{t+\Delta t} = \mathbf{f}\left(\mathbf{u}^t, \mathbf{v}^t\right) \tag{4.1}$$

and its evolvable initial conditions $\mathbf{v}^{t=0}$.

Here, $\mathbf{v} = [v_1, v_2, ..., v_N, v_{N+1}, ..., v_{N+N'}]^{\mathrm{T}}$ where N is the number of agent outputs and N' is the number of extra equations, $\mathbf{u} = [u_1, u_2, ..., u_M]^{\mathrm{T}}$ where M is the number of experimenter defined inputs to the agent and $\mathbf{f}(\mathbf{u}, \mathbf{v}) = [f_1(\mathbf{u}, \mathbf{v}), f_2(\mathbf{u}, \mathbf{v}), ..., f_{N+N'}(\mathbf{u}, \mathbf{v})]^{\mathrm{T}}$ are the agent's genetically encoded mathematical functions.

4.2.1 Evaluating an EMM

An EMM-based agent's behaviour over the course of its lifetime is determined as follows:

1. Set $t = 0$
2. Set $v_i = v_i^{t=0}$, $i = 1, ..., N, ..., (N + N')$
3. Update $\mathbf{u}^t$ with current agent inputs (i.e., current sensor values)
4. Evaluate $\mathbf{f}(\mathbf{u}^t, \mathbf{v}^t)$
5. Update agent output and extra state variables $\mathbf{v}^{t+\Delta t} = \mathbf{f}(\mathbf{u}^t, \mathbf{v}^t)$
6. Run agent for Δt timesteps using agent output values $v_i^{t+\Delta t}$, $i = 1, ..., N$
7. Set $t = t + \Delta t$
8. Go to step 3 (unless the end of the agent's lifespan has been reached)

The above steps apply to agents operating in a simulation environment as well as to embodied robotic agents operating in the real world. Steps 4-5 are equivalent to propagating agent inputs through an ANN to produce agent outputs for a single timestep of an agent's lifetime.

4.2.2 Evolving an EMM

The genetic operations used for creating and modifying EMM genomes will now be described. As was mentioned above, one of the goals of this paradigm is to minimize parameter tuning. In this document, a total of three different experiments are described. The first two are standard benchmarking experiments (Sections 4.3.1 and 4.3.2), while the third uses EMMs in an Artificial Life simulation (Part II). Many parameters related to EMMs in the various experiments were kept constant to demonstrate the transferability of the core paradigm to a variety of different applications. Note that this does not imply that these were necessarily the best settings for a given experiment. It should also be noted that a certain amount of parameter tuning between experiments is unavoidable owing to varying search space properties.

The EMM paradigm itself is representation-independent. In the work presented here, our implementation of the EMM paradigm uses a representation based on Symbolic Regression in GP (see Section 3.3), where equations are represented as nonlinear trees. Alternative genetic representations, such as those used in [135], where equations are represented as acyclic graphs or those compared in [123], where genomes are linear strings that encode nonlinear equation tree phenotypes, are left to future studies.

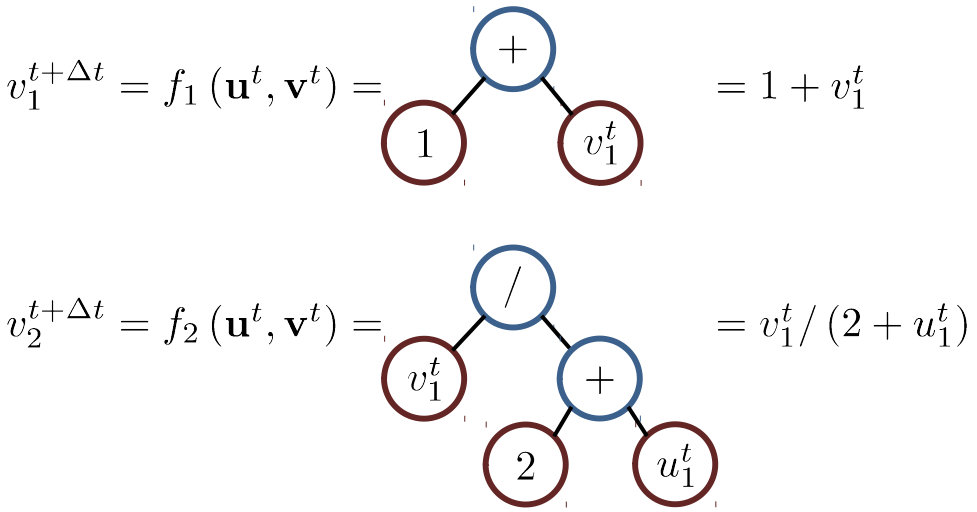


Figure 4.1: An example EMM for an agent with $N + N' = 2$. The two trees, along with the two initial values $v_1^{t=0}$ and $v_2^{t=0}$ (not shown), are how the agent's EMM is encoded in its genome.

Genetic Representation

For all experiments presented here, tree structures are used as genomes to represent the EMMs, as in GP. Only the four basic mathematical operators are allowed (addition, subtraction, multiplication and division), from which any analytic function can be approximated.

A genome contains $N + N'$ equation trees, one for each output and extra state variable of the system. Each tree contains a collection of terminal ("leaf") and nonterminal ("internal") nodes. The set of possible terminal nodes is comprised of all potential constants C and variables (i.e., input, output and extra state), while the set of possible nonterminal nodes is composed of addition, subtraction, multiplication and division. The number of "child" nodes (subtrees) of a nonterminal node is two, as all four of the basic operations have an arity of two. An example genome for an agent with $N + N' = 2$ is shown in Figure 4.1.

Initialization

The random initialization of the N initial equation trees in each initial genome (i.e., at generation or time 0) is done using the ramped half-and-half method as described in Section 3.3. Half of the trees are generated with a maximum depth of 1, while the other half have a maximum depth of 2. When generating trees using the "grow" method, each node below the specified maximum depth has a probability of 0.5 of being a terminal node, with a terminal node being set to a random variable or a random constant with equal probability. Initial genomes do not contain any extra variables, i.e., $N' = 0$. Initially, constants are randomly selected, as are initial output values $\mathbf{v}^{t=0}$.

Sexual Recombination

Sexual recombination is an important element of all Genetic Algorithms, as it allows for large jumps in the search space through combining two partial solutions. Any such genetic operation requires two parents to produce an offspring genome. Otherwise, a single parent's genome is cloned to produce an offspring genome. In EMMs, there are two levels of sexual recombination, tree level and equation level.

If an offspring genome is being produced via equation-level recombination, the two parents' equation sets are compared. For each equation that the two parents have in common (the equations have unique identification tags), either the equation from parent 1 or parent 2 will go to the offspring, along with that equation's associated initial value $v_i^{t=0}$. Which equation is inherited is decided randomly for each equation in common. An offspring must receive at least one equation from each parent. Obviously, this operation can only occur if the two parents have more than one equation in common (which is always the case if agents have at least two outputs). If an offspring receives an equation that contains an extra state variable modified by another equation that is not common to both parents, the offspring will inherit that equation as well. Several examples of equation-level sexual recombination are shown in Figure 4.2.

If a tree in the offspring genome is selected to undergo tree-level recombination, one of its nodes is selected at random and replaced with a randomly selected subtree (this is very similar to crossover in GP, see Section 3.3 and Figure 3.4). If there is a subtree below the selected node, it is discarded. If the tree undergoing recombination originated from the first parent, the subtree is selected from any of the trees of the second parent and vice versa. This allows for partial solutions to be reused and to be copied to different equation trees. This subtree grafting operation is similar to the subtree mutation operation described below. If an offspring receives a subtree containing an extra state variable modified by an equation that is not common to both parents, the offspring will inherit that equation as well.

Offspring genomes that are produced via equation-level recombination are not exempt from tree-level recombination. An offspring genome is still subject to a variety of genetic mutations, even if it has been produced via sexual recombination.

Mutation

Tree mutations (as well as extra equation mutations and sexual recombination, see below and above) can occur when a parent genome is being copied to its offspring. This means that mutations can only occur *between* generations. Agent genomes remain fixed throughout an agent's lifetime.

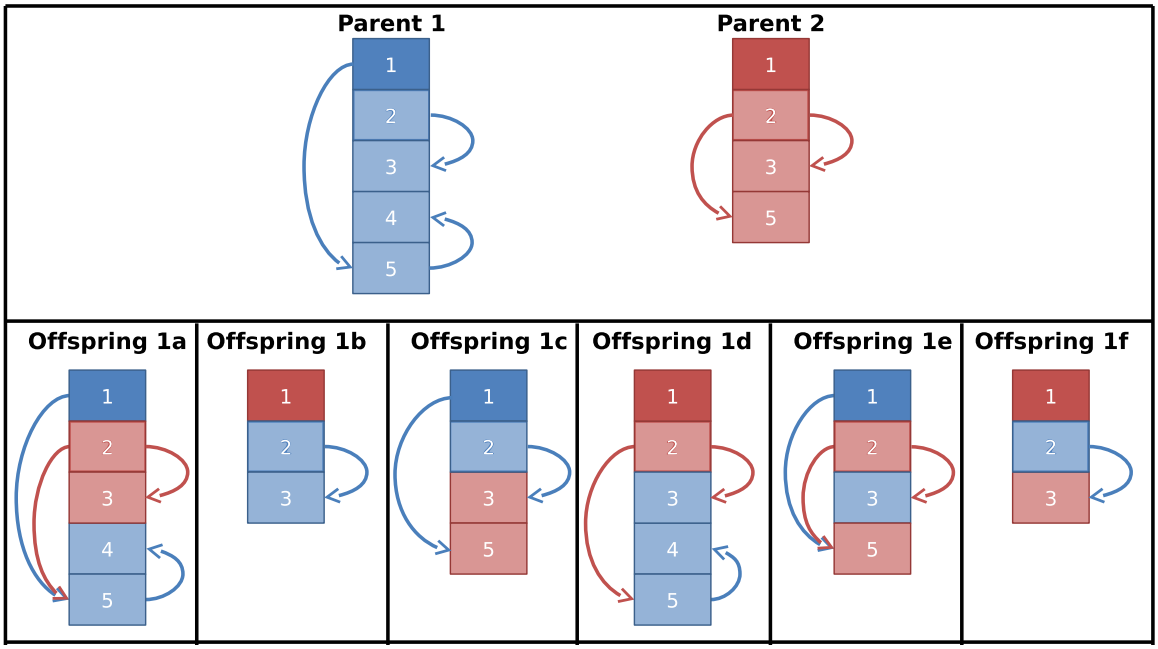


Figure 4.2: Several potential offspring genomes that can be produced from Parent 1 and Parent 2 reproducing. This diagram assumes that the agent controllers have one output. Each rectangle represents a genetically encoded equation and its ID. Lighter coloured rectangles are extra state variable equations that do not affect the output directly. Arrows show which equations depend on which extra state variables. Note that if there are no equations that depend on an extra variable in an offspring genome (excluding the extra variable's own equation), that extra variable and its corresponding equation are discarded.

These mutations are implemented in a manner similar to GP. If a tree is selected to be mutated, one of a variety of mutations is applied:

- **Point Mutation.** A point mutation performs one of several operations on a single randomly selected node in the tree:
  - *Perturbation of a constant.* This operation can only be performed if the tree in question contains one or more constants. The operation adds a random value to a randomly selected constant.
  - *Mutation of a nonterminal node.* This operation is performed if a perturbation of a constant was not done and if a randomly selected node is a nonterminal. A new nonterminal function is randomly selected from the set of addition, subtraction, multiplication and division.
  - *Mutation of a terminal node.* This operation is performed if a perturbation of a constant was not done and if a randomly selected node is a terminal. One of two mutations occurs, with equal probability. The chosen terminal is either mutated to a randomly chosen variable or it is mutated to a randomly chosen constant.

- **Subtree Mutation.** Occurring if there was no point mutation, this operation selects a random node on the original tree and replaces it with a new random subtree. This new subtree is generated in an identical fashion to initial trees (see "Initialization" above). A small variation to the standard subtree mutation was also added. With a small probability, the roles of the subtree and the original tree are swapped, i.e., a random node on the randomly generated subtree is replaced with the entire original tree and this becomes the new tree of the offspring. Our results in [60] suggest that this may improve solution quality.

Add Equation/State Variable Mutation

An offspring is subject to "add equation" mutations, as well as those mutations described above. This mutation produces a new equation tree of depth 1 or 2 using the "ramped half-and-half" method described previously. A new variable v_j, $j > N$ is added to $\mathbf{v}$. This is the extra state variable that the new equation tree will be modifying. The new variable v_j is also randomly incorporated into a randomly selected existing equation, either through a point mutation or a subtree mutation (with equal probability), as described above. Finally, $v_j^{t=0}$ is set to a random value.

Initial Value Mutation

An offspring's initial values $\mathbf{v}^{t=0}$ are subject to mutation as well. If an initial value is to be mutated, it will either be set to a new random value or perturbed by a random value. These two types of initial value mutations occur with equal probability.

Equation Reduction

When an offspring genome is produced, it may be checked for possible equation simplifications. We demonstrated in [60] that equation reduction can improve average solution quality while decreasing wall clock running times. The equation reductions implemented for all experiments presented here are as follows:

- The subtraction, addition, multiplication or division of two constants is reduced to a single constant by performing the encoded operation.
- The sum of two identical subtrees is reduced to $2\times$ a single version of the subtree.
- The subtraction of two identical subtrees is reduced to 0.
- The multiplication of a subtree by 0 is reduced to 0.

- The division of 0 by a subtree is reduced to 0.

Note that if there are no equations that depend on an extra variable in an offspring genome, that extra variable and its corresponding equation are discarded.

4.3 Benchmark Experiments

Now that the algorithmic details have been presented, we look to compare the performance and capabilities of EMMs to their ANN counterparts. As EMMs are abstractions of neural networks, it is expected that the same base algorithm using only addition, subtraction, multiplication and division should be able to solve basic ANN benchmarks that require recurrent neural networks, as well as more difficult benchmarks requiring ANNs with neural plasticity.

The two EMM-based algorithms used for the following two benchmark tasks have several details in common. In both instances, parent selection is done using "tournament selection." For each parent needed, a collection of agents is generated by randomly selecting (without replacement) from the current population. The agent with the highest fitness within this collection, or tournament, is selected as a parent. The parameter controlling the number of agents in such the tournament is called the tournament size. Tournament sizes need to be tuned for each specific task and algorithm, as different population sizes and solution search spaces are best exploited with different tournament sizes.

Whenever random constants are needed, they are selected from the uniform distribution $[-5, 5]$. Random initial values $\mathbf{v}^{t=0}$ are chosen from the uniform distribution $[-1, 1]$.

For both benchmark experiments, offspring genomes were produced through sexual recombination with a probability of 0.7, with equation-level recombination (denoted as "eqn-lvl xover" in result tables) occurring if the two selected parents have at least two equations in common and tree-level sexual recombination occurring on one or more of the offspring's $N + N'$ trees with a probability of $0.5/(N + N')$ per tree. Whenever a subtree needs to be chosen during a tree-level recombination operation (i.e., either as the subtree to be replaced or the subtree being grafted), there is a 10% chance that the root node will be a terminal node, otherwise it will be a nonterminal node.

Tree mutations happen with a probability of 0.1 per tree, whereas an extra equation/state variable tree is also added to a genome with a probability of 0.1 per previously existing tree. Offspring are required to undergo at least one tree or add equation mutation. Elitism, where the best performing agent in the current generation is cloned into the next generation, is used in both experiments as well.

If a tree mutation is to occur, it will be a point mutation with a probability of 0.5; otherwise, it will be a subtree mutation. If a point mutation is to occur and the tree in question contains at least one constant, a perturbation of a constant mutation will happen with a probability of 0.5, adding a random value taken from a Gaussian distribution with mean 0 and standard deviation 0.5 to a randomly selected constant. During subtree mutation, the original tree and the randomly generated subtree are swapped with a probability of 0.05. If an initial value is to be perturbed, the perturbation value will be randomly drawn from a Gaussian distribution of mean 0 and standard deviation 0.25. Offspring have a 10% chance of undergoing equation reductions.

Output values v_i, $i = 1, ..., N$ are capped to the range $[-1, 1]$, however extra state variables v_j, $j = (N+1), ..., N'$ are unbounded. If there is a zero divided by zero operation, or if any variable exceeds the minimum or maximum allowable values of the programming language and hardware being used (in all cases here, $\pm 1.79769 \times 10^{308}$), the current trial is terminated. A maximum genome size of 200 nodes is imposed across all experiments.

The differences between the algorithms used for the different benchmarks will be discussed in their respective sections below.

4.3.1 Common Benchmark: Double-Pole Balancing without Velocity Information

The Double-Pole Balancing without Velocity Information benchmark was chosen as the first benchmark as this seems to be the most reported benchmark in Evolutionary Robotics [148, 44, 58, 63, 79, 76, 82]. A controller must balance two poles by controlling the cart that they are attached to, using only current pole angle information, i.e., no pole velocity information is provided.

Problem Definition

A 1-kg cart has one degree of freedom, x. Two poles of different lengths $l_1 = 1$ m and $l_2 = 0.1$ m are mounted on the cart. The poles have masses $m_1 = 1$ kg and $m_2 = 0.1$ kg. The task is for the controller to balance both poles for a given number of timesteps by outputting a force $-10 \text{ N} \leq F_x \leq 10 \text{ N}$ on the cart, using only the angles of the two poles θ_1 and θ_2 and the position of the cart x as inputs (Figure 4.3). The cart cannot move past ± 2.4 m. The movements of the cart are evaluated via a numerical simulation based on a 4th-order Runge-Kutta integration with timestep $\Delta t = 0.01$ s. Rigid body dynamics are assumed and friction is ignored. Pole i is considered to have fallen over if $|\theta_i| > 0.63$. For the experiments reported here, the cart evaluation code from [148] was used, as it is freely available.

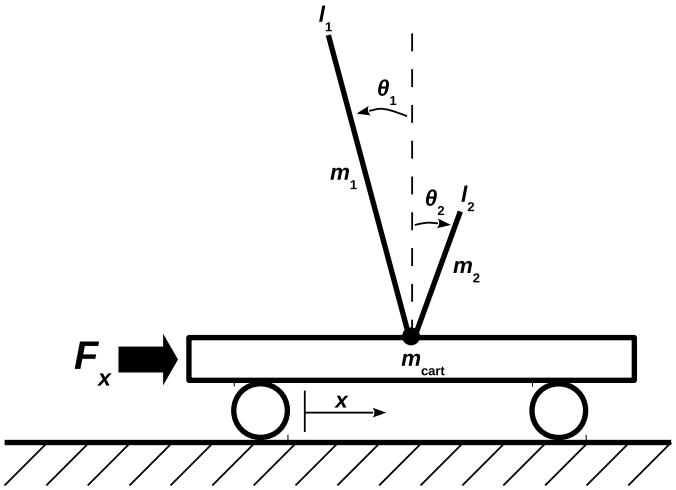


Figure 4.3: The double-pole balancing problem. Agents must balance both poles via a single output, -10 N $\leq F_x \leq 10$ N.

Agents are given a fitness value f based on a single trial with a maximum of 1,000 timesteps and initial conditions $\theta_1^{t=0} = 0.07$, $\dot{\theta}_1^{t=0} = \theta_2^{t=0} = \dot{\theta}_2^{t=0} = x^{t=0} = \dot{x}^{t=0} = 0$. A special fitness function that was originally introduced in [63] is used. The fitness function penalizes oscillations in an effort to prevent controllers from solving the task by moving the cart back and forth quickly, which would not require computing the missing velocity information. Thus, an agent's fitness f is calculated as $f = 0.1f_1 + 0.9f_2$, where

$$f_1 = t/1000 \tag{4.2}$$

and

$$f_2 = \begin{cases} 0 & \text{if } t < 100 \\ \frac{0.75}{\sum_{i=t-100}^{t} \left(|x^i| + |\dot{x}^i| + \left|\theta_1^i\right| + \left|\dot{\theta}_1^i\right| \right)} & \text{otherwise} \end{cases} \tag{4.3}$$

with t being the number of timesteps that both poles remained balanced during the 1,000 total timesteps.

Controllers have access to three scaled inputs at each timestep ($u_1 = \frac{\theta_1}{0.52}$,$u_2 = \frac{\theta_2}{0.52}$,$u_3 = \frac{x}{4.8}$, as in previously reported experiments) and have a single output $v_1 \in [-1, 1]$ that is multiplied by 10 at each timestep to give the force applied to the cart.

At the end of each generation, a single top individual is selected for further testing. If the selected agent can keep the poles balanced for 100,000 timesteps using the same initial conditions as above, it is subject to a "generalization test." This test consists of 625 unique sets of starting conditions (with $\theta_2^{t=0} = \dot{\theta}_2^{t=0} = 0$ and $\theta_1^{t=0}$, $\dot{\theta}_1^{t=0}$, $x^{t=0}$, $\dot{x}^{t=0}$ variable, details can be found in [63]). The agent is tested using each set of starting conditions and a point

is awarded for each time the agent balances both poles for 1,000 timesteps. Thus, an agent's maximum generalization score is 625. An agent is considered to be a solution if it has a generalization score of 200 or more.

This benchmark tests first and foremost whether or not a given ER algorithm can solve this challenging task. Successful algorithms are compared on their average generalization scores and on the average number of fitness function evaluations required to find a solution.

Issues with this Benchmark

As has been previously pointed out (see, e.g., [44] and [82]), high fitness values f do not necessarily correlate with high generalization scores, while low fitness agents can sometimes perform quite well on the generalization tasks. In an attempt to address this issue, an alternative fitness function was proposed in [79] that incorporated the original fitness function as well as generalization performance. Unfortunately, since the point of a generalization test is to see how well agents perform in novel scenarios, evaluating each agent on the generalization set defeats the purpose of such a test.

Thus, we will stick with the original experimental setup owing to the fact that it was used by all previously reported results. However, one must be cautious when drawing conclusions from these data, especially considering that several algorithmic modifications were made solely to deal with this highly unusual search space.

EMM Algorithm Details

For all experiments reported here, a tournament size of 6 is used and initial values are mutated with a probability of $0.5/(N + N')$. Populations are initialized with 1,000 individuals, but subsequent generations are reduced to 100 individuals. A similar approach to initialization was used in [44] to combat the bootstrapping problem. When producing an offspring population of 100 individuals from a parent population of 1,000 individuals, a tournament size of 60 is used. Furthermore, if no improvements in fitness are seen in 15 consecutive generations, the population is discarded and reinitialized. Again, this was also done in [44]. Agents in [148] were all initialized with network connections from each of the three inputs to the output, therefore the EMM initialization method described above is modified to force the output equation $v_1 = f_1(\mathbf{u}, \mathbf{v})$ of all initial agents to contain all three input variables. Preliminary experiments showed that top solutions contained only addition and subtraction, so for the results reported here only addition and subtraction are allowed during evolution. This greatly reduces the search space allowing for significantly faster evolution. Note that the canonical EMM algorithm described in Section 4.2.2 can solve this benchmark task, however

the tuning described above produces better performance.

As in all previously reported results only one agent per generation was tested for generalization capabilities, such testing had to be done wisely. An agent with a new top fitness is tested in the generation in which it first appears *if it can balance the poles for the full 1,000 timesteps.* If the top agent hasn't changed (i.e., through elitism) or it cannot balance the poles for the initial 1,000 timesteps, a randomly selected agent that can pass the original balancing test is subjected to the generalization tests. This was found to work better than selecting alternative agents to test based on fitness values, owing to the issues described above. The algorithm in [148] divides a population into subpopulations, or "species," based on genetic distance. Once per generation, the best *previously untested* elite agent of a species is subjected to the generalization tests. This allows for the testing of a variety of genetically diverse solutions within a single evolutionary run. Our EMM- based algorithm uses only a single population, i.e., there are no subpopulations nor speciation algorithms.

Results

Table 4.1 shows the reported results for various ANN-based ER algorithms, as well as the results obtained evolving EMMs in the manner described above. All results are averaged over 20 runs. The EMM runs restarted an average of 11.65 times. For the AGE results, an average of 10 restarts was reported [44]. Results from the canonical EMM, i.e., with ramped half-and-half initialization and all four mathematical operators allowed, are also reported, as well as results from the EMM algorithm as described above, but with equation-level sexual recombination disabled. Significant improvements have been reported using different evolutionary algorithms [82, 79, 76]; however, these results are omitted as these algorithms can only be applied to fixed-structure ANNs and not to the evolution of variable-size ANNs [44], ANNs with learning, or EMMs.

Table 4.2 shows how results vary depending on the stopping criterion (stop after 2,000 generations and record how many fitness evaluations it took to find the best test score or stop when we find a solution with a generalization score of 200+) and how many agents we subject to the generalization tests (test only one per generation or test all agents who balanced the poles for the full 1,000 timesteps during their fitness evaluation).

The simplified equations (one output equation and five extra variable equation, i.e., $N = 1$

Table 4.1: Fitness evaluation and generalization test results for various ER algorithms on the double-pole balancing without velocity information benchmark. All results averaged over 20 runs. We use μ to denote mean and σ to denote standard deviation.

Method	# Fitness Evaluations		Generalization
	μ	σ	μ
EMMs - canonical	894,716	884,145	298
Cellular Encoding [63]	840,000	N.A.	300
Enforced Subpopulations [58]	169,466	N.A.	289
EMMs - no eqn-lvl xover	117,350	173,206	313
EMMs	80,895	66,719	317
NEAT [148]	33,184	21,790	286
AGE [44]	25,065	19,499	317

Table 4.2: Fitness evaluation and generalization test results for various EMM algorithm runs on the double-pole balancing without velocity information benchmark. All results averaged over 20 runs. We use μ to denote mean and σ to denote standard deviation.

Stop	Test	# Fitness Evaluations		Generalization
		μ	σ	μ
Score 200	One	80,895	66,719	317
2000 Gens	One	103,800	71,808	346
2000 Gens	All	113,540	71,084	399
Score 200	All	48,935	34,944	314

and $N' = 5$) of an evolved agent with a generalization score of 456 are

$$v_1^{t+\Delta t} = -v_1^t - v_2^t - v_3^t - 7\theta_1^t + 6\theta_2^t \quad (4.4)$$

$$v_2^{t+\Delta t} = -v_4^t + 4x^t - 1.75 \quad (4.5)$$

$$v_3^{t+\Delta t} = -v_2^t + x^t \quad (4.6)$$

$$v_4^{t+\Delta t} = -v_5^t + 3.14 \quad (4.7)$$

$$v_5^{t+\Delta t} = v_6^t + 0.11 \quad (4.8)$$

$$v_6^{t+\Delta t} = 3\theta_1^t + 2.80 \quad (4.9)$$

The current force applied to the cart is calculated as $F_x^{t+\Delta t} = 10.0 v_1^{t+\Delta t}$ with v_i, $i \in \{2, 3, 4, 5, 6\}$, being extra state variables with no associated agent output. Evolved initial values are omitted, and any multiplication operators are the result of hand-performed simplifications (e.g., $x + x = 2x$).

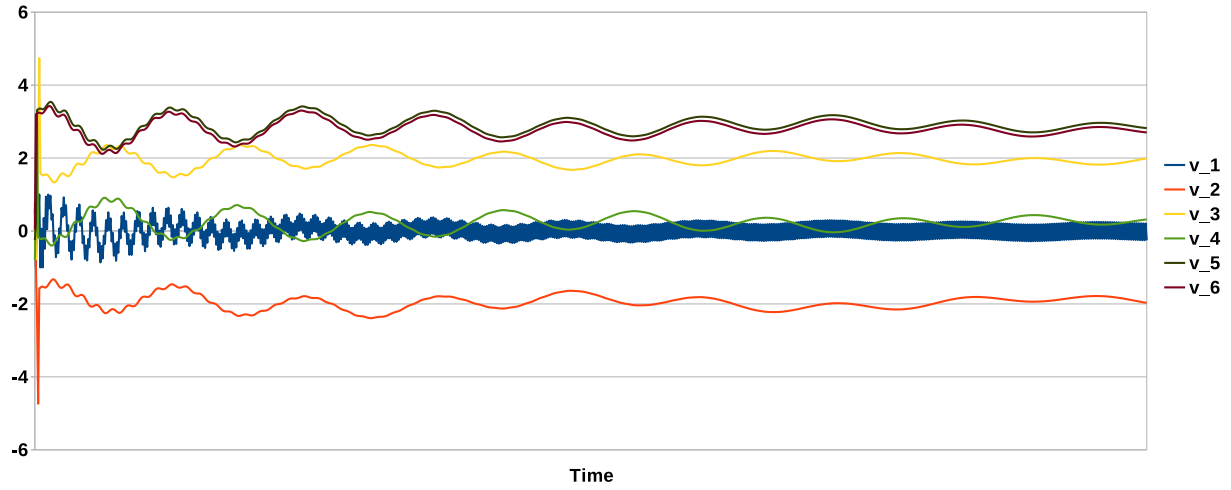


Figure 4.4: Output values of v_i over 1,000 timesteps of double-pole balancing without velocity information.

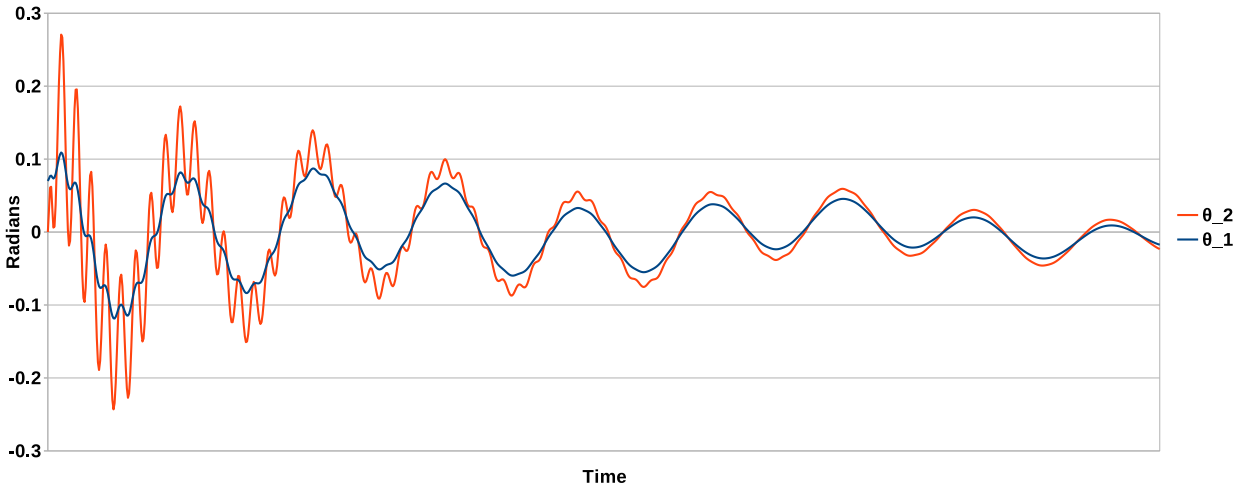


Figure 4.5: θ_1 and θ_2 values over 1,000 timesteps of double-pole balancing without velocity information.

The v_i values of this EMM are shown in Figure 4.4 for the original 1,000 timestep training run, with the corresponding pole angles θ_1 and θ_2 shown in Figure 4.5.

Discussion

The main take-away from these results is that EMMs can successfully produce solutions to the challenging double-pole balancing without velocity information task. Furthermore, solutions are found using a similar, if somewhat higher, number of fitness evaluations as in the top ANN experiments while yielding comparable generalization scores. The differences in

fitness evaluations are most likely owing to EMMs effecting larger search spaces than ANNs by having two types of nodes (addition and subtraction) instead of a single type of neuron. Other contributing factors could be genetic algorithm implementation and parameter tuning.

EMM solutions use equations with references to previous output and extra variable states. This is analogous to the recurrent connections found in the top ANN solutions in [148] and [44] and is necessary owing to the lack of velocity information. The EMM solutions are compact and bloat-free, again comparable to the compact solutions found in the ANN experiments. Equation-level sexual recombination provides performance benefits on this benchmark task.

Further analysis is difficult, owing to the issues with the experimental setup. As was shown in Table 4.2, better solutions did exist within EMM populations; however, the technique used to test one agent per generation often missed them. Future work should look to improve EMM results by implementing NEAT-like speciation, which would allow for more diversity (and thus potentially better solutions) within the evolving population while improving the effectiveness of top agent testing. Furthermore, this benchmark should be redesigned in future studies to improve the correlation between fitness and test scores.

4.3.2 Learning Benchmark: Double-T Maze

In its simplest form, a T Maze test consists of a series of trials where an agent starts in the home position, chooses one of the two "arms" of the maze to visit and collects the reward at the end of that arm. In some cases, the agent is automatically returned home once the end of the maze is reached, whereas in others part of the task is for the agents to find their own way home. For each trial, one arm of the maze contains a high reward, while the other contains a low one. The purpose of this task is to demonstrate learning. A successful agent should search both arms for the high reward, and then return to the high reward arm of the maze in subsequent trials. If the reward is moved, the successful agent should search for and relearn its new position. A Double-T Maze has four arms instead of two, while still only having one high reward (see Figure 4.6). T Mazes have been used to demonstrate and test learning for a variety of algorithms that evolve ANNs with plasticity (see, e.g., [142, 129, 43, 133, 131, 144, 132, 141]) as well as CTRNNs [14].

An important distinction between various T Maze experiments is whether the environment is continuous (e.g., [14, 132]) or discrete (e.g., [142, 129]). In the continuous environment, the agent is a simulated robot with proximity sensors to detect walls and has direct control of two wheels (i.e., two output variables). In the discrete environment however, agents must simply decide to move straight for one unit, or turn left or right 90°. This only requires a single output variable.

The specific T Maze chosen for this benchmark was the discrete Double-T Maze with

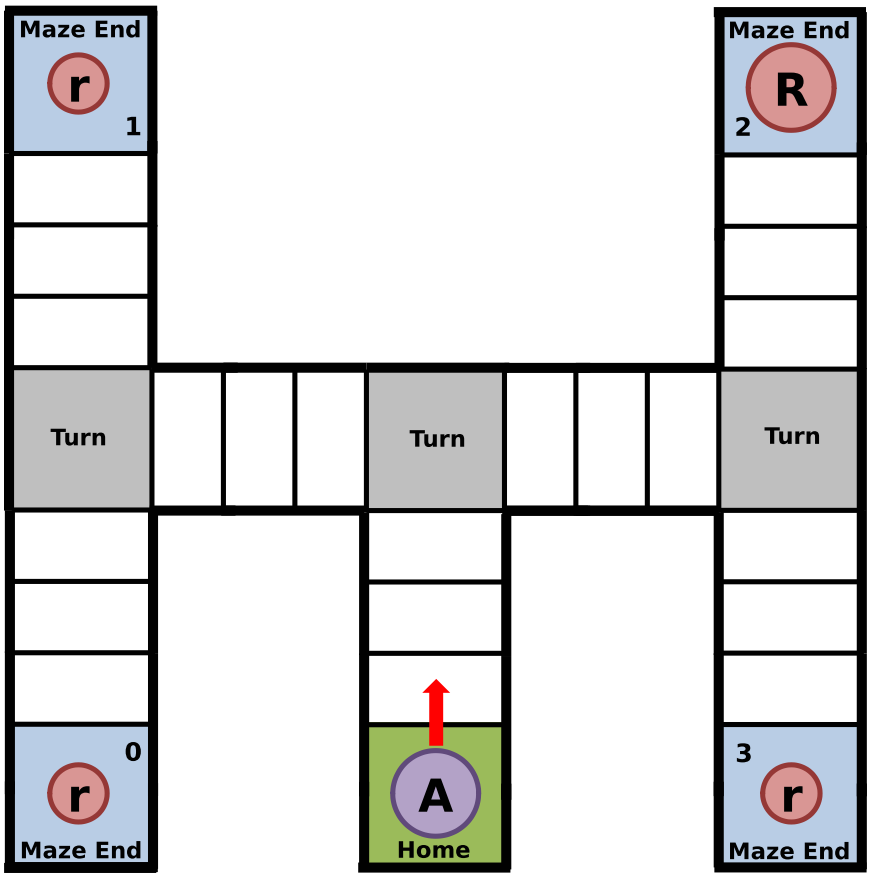


Figure 4.6: A discrete Double-T Maze. "A" is the agent, "r" are low rewards and "R" is the high reward.

homing requirements used in [142]. This version was selected for several reasons. This problem domain has only been solved with plastic ANNs: fixed-weight ANNs have so far been unsuccessful. Furthermore, the Double-T Maze is very difficult, requiring agents to repeatedly choose one of four different movement patterns depending on where they find the high reward. If agents have yet to discover the location of the high reward, they must search up to four different maze arms (requiring four different movement patterns) sequentially. Adding to the difficulty is the requirement that agents must return home after reaching an end of the maze. This doubles the size of each of the four movement patterns. For example, to get to the top left part of the maze an agent must turn left then right. To then return home, the agent must turn left, then right again. There is a unique four-turn pattern for each of the four arms (LL-RR, LR-LR, RL-RL, RR-LL).

The main point of this benchmark is to demonstrate that a given algorithm can solve this challenging task. However, for the sake of comparison, we attempt to tackle this problem using the same number of fitness evaluations as in [142].

Problem Definition

The Double-T Maze used here is the same as the one in [142] (see Figure 4.6). Agent fitness is evaluated over a series of trials. For each trial, the agent is evaluated for a maximum of 35 steps, with each step consisting of one evaluation of the agent's equations and the execution of one move (forward or turn 90°) based on the agent's output v_1. A trial begins with the agent at the "home" position. If an agent executes a turn command while not on a "turn" position (i.e., while on the home position, one of the four reward positions or in a corridor) or executes a "move forward" command on a turn position, this is considered a crash. Crashes end the current trial, returning agents to the home position and subtracting 0.4 from their total fitness. If an agent completes a trial without returning to the home position, a penalty of 0.3 is applied to their total fitness. If the agent reaches one of the three low reward arms of the maze, a score of 0.2 is added to their fitness. The high reward arm yields a fitness boost of 1.0. When an agent reaches the end of a maze arm, it is automatically turned 180°. Corridors and turn points last for three forward steps each.

Agents have access to four inputs, "turn," "maze end," "home" and "reward." The turn input is set to 1.0 when the agent is on a turning point, 0.0 otherwise. The maze end input is set to 1.0 when the agent is at the end of one of the four maze arms, 0.0 otherwise. The home input is set to 1.0 when the agent is at the home position, 0.0 otherwise. Finally, if the agent collects a low reward, the reward input is set to 0.2 for one step. Collecting a high reward sets the reward input to 1.0 for one step. The reward input is 0.0 at all other times. Note that these input values were chosen to be identical to those used in [142].

Agents have one output v_1. If $v_1 < -0.33$, the agent performs a 90° left turn and then moves forward one unit. If $v_1 > 0.33$, the agent performs a 90° right turn and then moves forward one unit. Otherwise, the agent moves forward one unit in its current direction. All inputs and references to variables v_i, $i = 1,..., (N + N')$ are subject to noise by adding a random value taken from the uniform distribution $[-0.005, 0.005]$ at each equation evaluation.

Agent fitness is evaluated on a set of 200 trials, with the high reward randomly positioned for the first trial. The high reward is randomly repositioned after a randomly selected number of trials H_t, with $35 \leq H_t \leq 65$. Reward repositioning happens three to four times per 200 trial run, with H_t being regenerated after every repositioning. During the evolutionary runs, the first four high reward positions are forced to be distinct. This is not enforced for the 100 sets of 200 trials used for testing top agents.

Table 4.3: Double-T Maze results from 50 evolutionary runs. A "successful run" has occurred if an agent scores 189.4 or higher on the test set. Note that all results from algorithms marked with an asterisk are provided for reference only as they are *estimates* taken from visual inspection of graphs and were calculated on a different test set with significantly fewer agents tested. We use μ to denote mean and σ to denote standard deviation.

Algorithm	Test Score			# Successful
	Median	μ	σ	
EMM - 10 isls (canonical)	177	170	21	4
EMM - 10 isls (no eqn-lvl xover)	177	172	18	9
EMM - 40 isls (canonical)	186	183	10	17
EMM - 40 isls (no eqn-lvl xover)	187	186	5	18
ANN - modulatory*	190	N/A	N/A	N/A
ANN - plastic*	117	N/A	N/A	N/A
ANN - fixed-weight*	21	N/A	N/A	N/A

EMM Algorithm Details

For this task, an island model is used (see Section 3.2). Each of the 10 islands has a population of 100, giving a total population size of 1,000. Islands are organized in a ring shape, with the top agent from each island migrating to the left or right island every 20 generations. The direction of migration is constant across all islands and switches after each migration. Each island population is tested on its own set of 200 trials, with all 10 sets being regenerated after each generation. The tournament size used is 15 on each island, and each island's top agent is cloned for the next generation (elitism). Initial values are mutated with a probability of $0.1/\left(N + N'\right)$. All four basic mathematical operators (addition, subtraction, multiplication and division) are allowed, and random initial genomes are generated using the ramped half-and-half method.

At the end of every generation, the top agent from each island is tested on a fixed test set of 100 randomly generated 200 trial runs. Each experiment is run for 1,000 generations, and the final result is taken to be the agent that performed the best on the test set.

Each island is implemented as a separate process so that the algorithm can take full advantage of the parallel architectures of modern CPUs. A Master/Slave parallel implementation is used, where a "master" process handles the synchronization of "slave" processes (i.e., the islands). Islands are synchronized and migrants exchanged after every 20 generations.

Results

A solution's quality is determined by its average score on the test set of 100 randomly generated 200 trial runs. A perfect score of 200 is not theoretically possible as a top solution

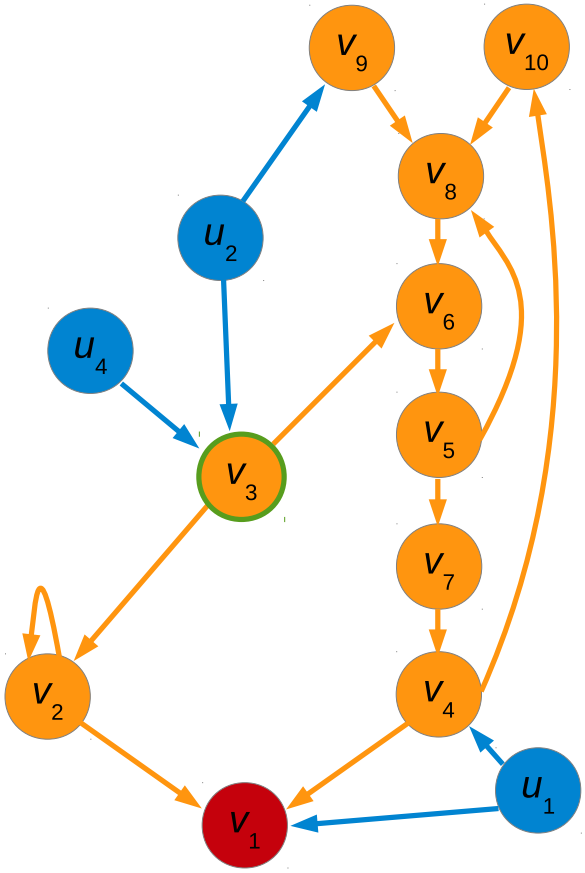


Figure 4.7: Relationship between variables within a successful EMM agent's evolved equations. Inputs are shown in blue, the agent's output v_1 is red and the extra state variables are orange. Data require one timestep to traverse an orange arrow, whereas input data traverse blue arrows instantaneously. The state variable v_3 is emphasized in green as it plays the role of neuromodulator. The calculations performed at each node are shown in (4.10) to (4.19).

will have to collect occasionally a low reward to detect that the high reward has been moved. Practically, top agents will not be able to predict the new location of the high reward after a move and therefore may have to collect several low rewards as the T Maze is explored. The best observed score on our test set was 193.344. For the test set used in these experiments, the "worst-case perfect test score" was calculated to be 189.4. This value is the average score of a perfect agent across the 100 test runs, assuming the agent *always* searches each low reward arm once before finding the high reward (hence "worst-case") and always returns to the high-reward arm once it is discovered (hence "perfect agent"). Thus an agent with a test score of 189.4 or higher is considered to be a solution to this Double-T Maze. Table 4.3 shows the results from 50 runs using the same test set, but with different initial populations and different training sets. The runs with 10 islands use the same number of fitness evaluations as in [142]; however with significantly more agents tested. This is because

we test the top agent from each island at each generation, while in [142] only the top agent from the final generation was tested. Experiments with 40 islands are also reported, demonstrating performance improvements given more fitness and test evaluations. Experiments with equation-level sexual recombination disabled are reported as well, demonstrating slight *improvements* over the canonical runs. This suggests that equation-level recombination is not always beneficial (although it was beneficial in the double-pole balancing benchmark, as well as in the communication experiments reported in Part II). Note that all results from [142] are provided for reference only as they are *estimates* taken from visual inspection of graphs and were calculated on a different test set with significantly fewer agents tested.

The full system of equations (with rounding and simplifications) of a top agent with a test score of 191.884 is

$$
\begin{aligned}
v_1^{t+\Delta t} &= u_1^t v_2^t v_4^t && (4.10)\\
v_2^{t+\Delta t} &= 5.92 v_3^t / v_2^t && (4.11)\\
v_3^{t+\Delta t} &= u_2^t - u_4^t - 0.69 && (4.12)\\
v_4^{t+\Delta t} &= -0.41 u_1^t v_7^t && (4.13)\\
v_5^{t+\Delta t} &= -0.33 v_6^t && (4.14)\\
v_6^{t+\Delta t} &= 0.65 - v_8^t - (1.18/v_3^t) && (4.15)\\
v_7^{t+\Delta t} &= v_5^t - 0.48 && (4.16)\\
v_8^{t+\Delta t} &= v_9^t(0.02 v_9^t + 0.05 v_{10}^t - 0.23) && (4.17)\\
&\qquad - v_5^t - 0.14 v_{10}^t + 0.48 &&\\
v_9^{t+\Delta t} &= 56.11 u_2^t && (4.18)\\
v_{10}^{t+\Delta t} &= 0.80 - v_4^t && (4.19)
\end{aligned}
$$

Note that v_1 is the agent's output variable, u_1 is the "turn" input, u_2 is the "maze end" input, u_4 is the "reward" input and evolved initial conditions $\mathbf{v}^{t=0}$ are omitted. Figure 4.7 shows the relationships between variables within this agent's evolved equations.

The only equation containing the variable u_4 (the "reward" input) is (4.12)[1], and its corresponding state variable v_3 seems to be modulating learning. By examining these equations and their behaviour (see Figure 4.8), it is apparent that this equation plays the role of learning modulator. The value of v_3 is -0.69 at all times, except for when a small reward is collected. A small reward causes v_3 to spike (as in this case $u_2^t = 1.0$ and $u_4^t = 0.2$, therefore $v_3^{t+\Delta t} = 1.0 - 0.2 - 0.69 = 0.11$), which in turn causes the agent to change movement patterns (v_1 patterns). Of the 48 successful runs across the Double-T Maze experiments, 20 had top

[1] note that it also appears in v_1 but is divided by ∞

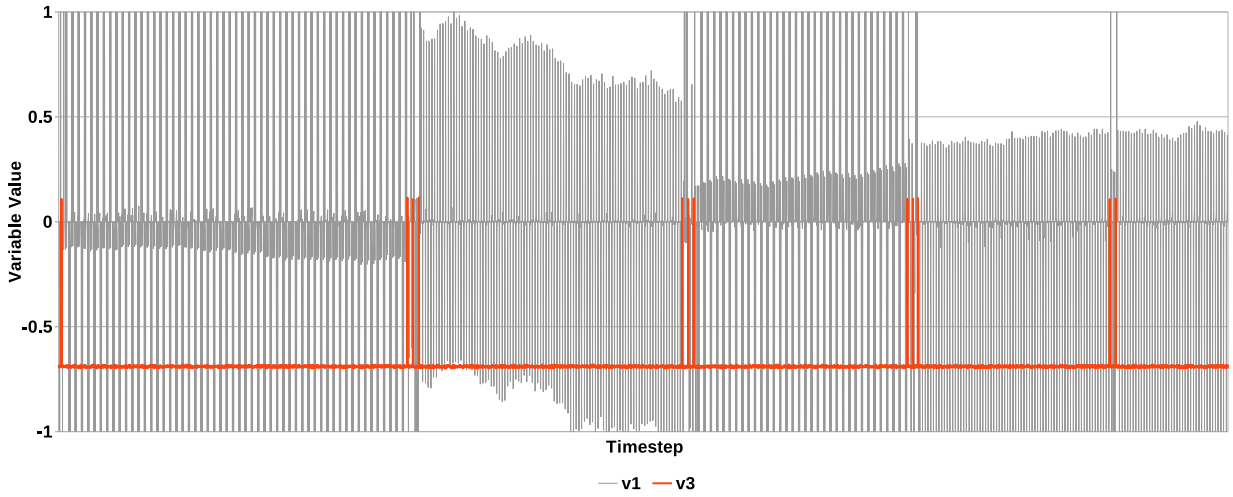


Figure 4.8: Values of v_1 and v_3 of a top agent during a test set of 200 trials.

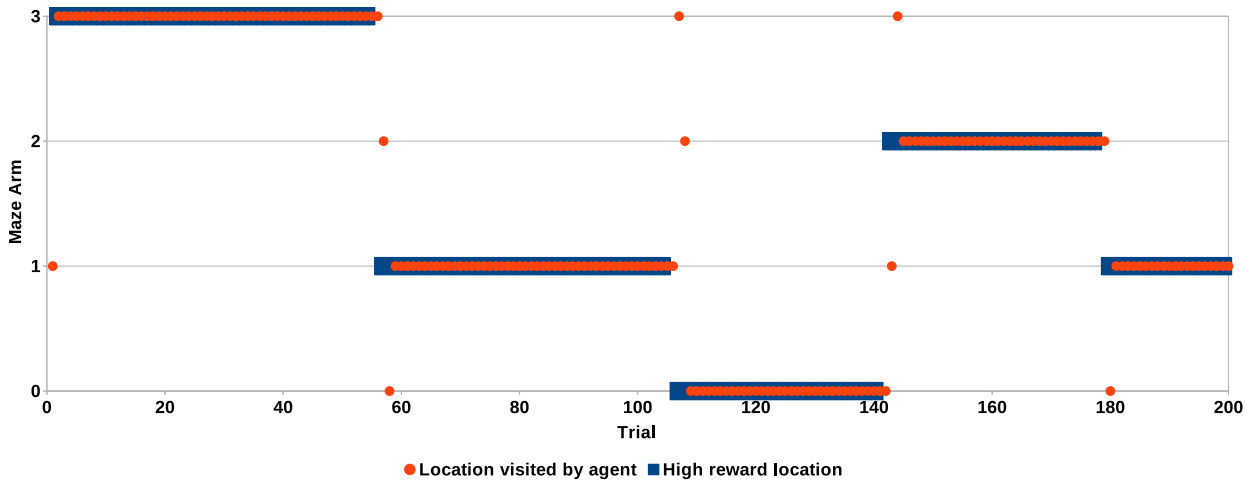


Figure 4.9: Position of high reward and position visited by a top agent during a test set of 200 trials.

agents that contained functions similar to the right hand side of (4.12), including the agent described above.

Figure 4.9 shows which arm of the T Maze contains the high reward, as well as which arm the agent described above visits for each trial in the same 200 trial test run as in Figure 4.8. When the agent receives a low reward, it visits a different arm on the next trial. When the agent receives a high reward, it returns to the same arm on subsequent trials until it receives a low reward. The "forage" pattern of the agent seems to be constant: arm 1, arm 3, arm 2, arm 0, then back to arm 1 and the pattern repeats.

Discussion

These results demonstrate that EMMs can successfully solve a difficult task requiring learning, namely the discrete Double-T Maze with homing requirements. The EMMs performed significantly better than plastic ANNs without neuromodulation, and slightly worse than the plastic ANNs with neuromodulation. There are numerous factors that could contribute to these variations in performance, including differences in test sets, evolutionary algorithms and implementations. Differences in evolvability may again be playing a role as well. Regardless, EMMs were able to produce several top solutions to this challenging task, using the same number of fitness evaluations as were used to evolve solutions with ANNs. An interesting result was seen in a top EMM agent with a test score of 191.884 as it contained an extra equation that produced neuromodulation-like behaviour, *without the experimenter having to explicitly implement it.* A similar mathematical function appeared in over 40% of all successful top agents. This provides further evidence that neuromodulation may indeed play a role in learning in biological brains. It seems as though neuromodulation is not required to solve this Double-T Maze, however, as many successful solutions did not employ it. Several ANNs with neuroplasicity enabled but neuromodulation disabled also solved the Double-T Maze in [142].

These results demonstrate the abstraction power of EMMs: complex learning with neuromodulation can be evolved using only the four basic mathematical operators, requiring no *a priori* development and programming of special artificial neural structures and their associated behaviours.

4.4 Conclusions and Future Work

Thus concludes the description and benchmark testing of our novel AI paradigm, Evolvable Mathematical Models. This algorithm evolves artificially mathematical abstractions of neural networks. Testing on the double-pole balancing without velocity information benchmark demonstrated that EMMs could achieve comparable solutions to their ANN counterparts on a difficult task, albeit requiring roughly three times more fitness evaluations. It was shown that better results could be achieved by testing all potentially successful solutions. Even in the best case scenario however, the EMM algorithm as described still required more fitness evaluations to find a solution than the top results using ANNs. This is likely due to EMMs employing more types of nodes than standard ANNs (i.e., addition and subtraction in the best EMM solutions versus a single type of neuron in the ANN experiments), which produces a larger search space. Another factor that might contribute to differences in required fitness function evalutions is the differences between the evolutionary algorithms used to search

for solutions, as the search method is not standardized for this benchmark. For example, NEAT employs a complex speciation algorithm that requires additional parameter tuning and computational complexity that is not taken into account by the fitness evaluation metric.

Future work should look to improve the evolvability of EMMs. One option would be to use alternative genomic representations to produce and evolve the equations (a comparison of the performance of several linear representations on Symbolic Regression tasks can be found in [123]). Another option might be to modify an ANN-based algorithm, such as NEAT, to produce EMMs.

The second benchmark was the discrete Double-T Maze with homing requirements. This difficult task requires agent learning and has only been solved by a handful of ANN-based algorithms using connection weight plasticity. Using the same number of fitness evaluations as used to find the top ANN solutions, several EMM runs were successfully able to solve this task. It is interesting that while the ANNs that solved the maze have variable structures (as their connection weights vary over time), EMMs are able to accomplish this learning task with fixed structures, as their equations do not change over the lifetime of an agent. Furthermore, this benchmark demonstrates the abstraction power of EMMs, as the ANNs that solve this task use connection weight plasticity and (sometimes) neuromodulation. As was shown in the results above, neuromodulation-like behaviours emerged from the EMMs in over 40% of successful runs, without the programmer having to specify the specific neural mechanisms. Contrary to the results from the double-pole balancing experiments, disabling equation-level sexual recombination slightly *increases* solution quality on the Double-T Maze task, demonstrating that the usefulness of this genetic operation varies between problem domains. Future work should look to extend these discrete Double-T Maze results into the continuous Double-T Maze domain.

Finally, it should be noted that the canonical version of our EMM algorithm can solve both benchmark tasks without needing any modifications or complexification (although adjustments to the GA and parameter tuning were required). On the other hand, ANN-based algorithms such as AGE require the additional complexity of neuroplasticity in order to tackle the Double-T Maze.

Chapter 5

Conclusion of Part I

The above experiments have proven our hypothesis correct. Evolvable Mathematical Models can indeed be used to develop robust, learning-capable controllers.

We have demonstrated that evolvable mathematical equations can be used to abstract a variety of artificial neural network structures using only the four basic mathematical operators: addition, subtraction, multiplication and division. While these initial experiments indicate that evolving EMMs requires more fitness function evaluations than evolving ANNs, our results suggest that the benefit of using EMMs is that one need not worry about what type of agent representation to use for a given problem, as the canonical EMM algorithm was able to find solutions for both problem domains presented here. It should be noted, however, that by deviating from the canonical implementation in the double-pole balancing domain (by restricting potential mathematical operators to addition and subtraction only and by changing the way populations of equations were randomly initialized), we were able to discover solutions more rapidly. Since this is the first version of an EMM implementation, it is hoped that future work will increase the evolvability of these equations and demonstrate their applicability to other problem domains.

Some interesting questions arise from the presented results. If EMMs can successfully abstract various types of ANN, which in turn are simplifications of what we know about biological neural networks, can EMMs then abstract what we *don't know* about biological neural networks as well? To put it another way, can EMMs model the behaviours of a biological brain, despite our lack of scientific understanding of how such a brain physically operates? If not, why not? What is missing in our mathematical toolkit? Or are complex brains, such as those of humans, fundamentally unmodelable? These are big, exciting questions and it is hoped that future work with EMMs will help to answer them.

Part II

Simulating the Evolution of Communication

Chapter 6

Introduction

Now that Evolvable Mathematical Models have been presented and successfully tested on two benchmark tasks, we look to demonstrate further the power of EMMs as evolvable artificial intelligences in a novel simulation. For this task we choose to simulate the emergence and evolution of communication, as it is a notoriously hard problem with a wide gap between simulation results and observed behaviours in nature. For all results and discussion here, we will be using the definition of communication from [96]:

> *Communication occurs when the action of... one organism is perceived by and thus alters the probability pattern of behaviour in another organism in a fashion adaptive to either one or both of the participants.*

Simulations within this problem domain are an important precursor to the understanding of the emergence and evolution of natural language, a highly controversial open problem within the scientific community which has a notable lack of experimental data on which to validate theories. The debate has heard from many researchers, including Darwin himself who mused on the human faculty of language in *The Descent of Man* [34]. Other scientists interested in the evolution of language include Hauser and Chomsky, who argue that human language arose as a "spandrel" (a term coined by Gould and Lewontin [59]), with natural selection promoting the evolution of language-ready neural pathways for reasons other than language (e.g., reasoning and tool-making) [67]. Pinker and Bloom, on the other hand, argue that language did indeed arise through direct natural selection [125], as natural selection is the only mechanism that we know of that can produce such complexity without a designer. Finally, there is a third notion, championed by Christiansen, Kirby and others, that suggests that language itself is what is undergoing evolution, self-adapting to the human brain (see, e.g., [25]). We wouldn't be surprised if the truth contains elements of all three of the aforementioned suppositions.

Unfortunately, while there is an abundance of theories on the topic, there are few scientific truths. This is largely owing to the fact that communication leaves no fossil record and thus researchers must instead use cross-species comparative analysis [66, 67]. The study of the evolution of language is so difficult that in their book *Language Evolution*, Christiansen and Kirby titled the first chapter "Language Evolution: The Hardest Problem in Science?"

Within this chapter, Christiansen and Kirby describe one of the few consensuses of the field [26]:

> *[An] area of consensus is the growing interest in using mathematical and computational modelling to explore issues relevant for understanding the origin and evolution of language.*

Indeed, there have already been several notable computer simulations investigating the evolution of language, including Kirby's Iterated Learning Model [83] that demonstrates how language itself can adapt (i.e., without biological evolution) via pressures arising from language transmission and learning. The simulations and analysis by Cangelosi and Parisi [19] suggests that language may have evolved as a byproduct of cognitive abilities, lending evidence to the spandrel theory. Cangelosi also simulated the emergence of a simple syntax [18] and simple nouns and verbs (with Marocco and Nolfi, [97]). Steels and Vogt [149] studied language evolution via robotic agents playing language games. Finally, a mathematical framework for the evolutionary dynamics of grammar learning was developed in [119] by Nowak, Komarova, and Niyogi. The reader is referred to Cangelosi and Parisi's book *Simulating the Evolution of Language* [20] for further reading on the subject.

A commonality among all the aforementioned theories, models, and simulations is that there are a number of "preadaptations" that are assumed to have occurred prior to the emergence of language. While Hauser and Chomsky do not require that such preadaptations be communicative in nature, most other theories are based on the assumption that some form of communication must exist before language can emerge.

We look to develop a simulation in which communication emerges from initially non-communicating agents with the hope of not only demonstrating the strength of the EMM paradigm, but also of helping inform the debate about the emergence and evolution of communication—and ultimately language—in nature.

The thesis statement of Part II is:

> *It is possible to construct an artificial world populated with EMM-based agents where complex, dialogue-based cooperative communication repeatedly emerges and evolves from initially noncommunicating agents.*

The objectives of Part II of this thesis are to produce novel results in the simulation of the emergence and evolution of a cooperative communication scheme using Evolvable Mathematical Models by evolving agents that use an evolved communication scheme to transmit information about two independent variables over a one-dimensional channel, and to investigate the transferability of EMMs from simulation to robotic hardware.

6.1 Methodology

We look to develop a simulation of the emergence and subsequent evolution of interagent communication. The results should be readily examinable, so that the communication schemes and their evolutionary trajectories can be observed. Agents from this simulation world should be transferrable to robotic hardware.

With these goals in mind, we develop an Artificial Life (ALife) simulation where EMM-based agents are born, reproduce and die. To reproduce, two agents must meet with no information *a priori* on the whereabouts of each other. Only a one-dimensional, nondirectional real-valued communication channel is available to them. This ensures that any information that an agent has about its neighbour must have been received specifically via the interpretation of the *content* of the communication channel. Agents have two outputs: they can control their own orientation and what values to communicate. Agents automatically move in their current forward direction at each timestep. Agents can "hear" the communication output value of their nearest neighbour. When two agents reproduce, they produce one offspring agent and one randomly selected agent in the simulation is removed. Thus, selection pressure arises not from a fitness function, but instead from the fact that reproductively viable genomes remain in the simulation within offspring agents, while less reproductively viable genomes are pushed out before reproduction can occur. Simulations are run for many millions of timesteps and the reproduction rates (number of reproduction events per timestep) of agent subpopulations are tracked.

Digital snapshots of the populations are saved at fixed intervals over the course of a simulation run. Once a run is complete, the snapshot from the interval with the highest reproduction rate is examined. The top agent (in terms of number of offspring produced) from this snapshot is isolated, its EMM is displayed and its behaviour and performance are tested within a separate simulation. This test program is the same as the main simulation, except that only two clones of the agent undergoing testing are present. These two clones are run from a variety of initial positions to allow the experimenter to determine which initial configurations lead to the agents successfully finding each other. It also allows for the examination of the communication schemes employed by the agent, if any exists/emerges.

Entire population snapshots are also tested within a different test simulation. Again, the program is the same as the main simulation, except with reproduction disabled. This allows for the properties of a given snapshot to be examined without evolution (i.e., without these properties changing over the course of the test). Experiments run using this testing framework are used to compare the reproductive viability of a population with and without the use of its communication channel, as well as to generate communication data for further analysis.

The evolutionary history of top agents is examined via digital snapshots of populations that existed before these agents emerged. These snapshots are tested using the same test simulations described above.

Finally, a top agent's EMM is selected for hardware testing. A method for transferring agent inputs and outputs from the simulation world to the real world is devised and the embodied EMMs are run from a variety of initial configurations, as in the test simulation above. These experiments are used to determine the reproductive viability of these evolved EMMs when embodied in hardware and the overall amenability of EMMs to hardware transfer.

6.2 Results and Contributions

We developed a novel Artificial Life simulation called *NoiseWorld* in which EMM-based agents evolve. The best agents to emerge from these simulations could communicate information to each other about both their x and y positions, allowing them to locate one another from any starting configuration. As the provided communication channel was one-dimensional and nondirectional, this implies that agents were encoding their 2D position information in some fashion prior to transmission and subsequently decoding it upon message receipt to extract meaningful information about both position dimensions. The evolutionary trajectories leading to these top agents were examined, yielding several insights into the emergence and evolution of cooperative, dialogue-based communication. A top agent was embodied in robotic hardware, demonstrating a successful transfer of behaviours evolved in simulation to real-world situations and highlighting the transferability of EMMs to physical systems.

6.3 Outline of Part II

A literature review of previous work on the simulation of the emergence and evolution of interagent communication is presented in Chapter 7. This will include experiments from Evolutionary Robotics, as well as experiments from the related field of Artificial Life (ALife),

which will be defined. After the literature review, our novel simulation world, *NoiseWorld*, will be presented (Chapter 8). Details about the world itself, as well as the EMM settings used and other experimental details, will be given. This will be followed by the presentation and discussion of results from multiple *NoiseWorld* runs and additional test simulations (Chapter 9). Chapter 10 will present the hardware experiments and results, and concluding thoughts will be given in Chapter 11.

Chapter 7

Literature Review

The following literature review will cover the most common experimental paradigms used in the simulation of the emergence and evolution of communication, as well as those that are deemed the most relevant to our experiments. For a detailed comparison of previous work (up until and including 2002), the reader is referred to a paper by Wagner et al. [156]. For an in-depth analysis of more recent experiments, as well as a comprehensive literature review encompassing work up until and including 2009, the reader is referred to *Evolution of Communication and Language in Embodied Agents* by Nolfi and Mirolli [115].

7.1 Communication in Evolutionary Robotics

A number of Evolutionary Robotics experiments have successfully simulated the emergence of interagent communication.

The experiments by Floreano, Keller, and others described in [52], [107], [108], and [159] are all based on a similar setup. A group of simulated robots must forage in a 2D environment for food and are rewarded for the number of timesteps spent on or near the food source. In some cases, a poison source was also included in the environment, and agents are penalized for being near it. Each simulated robot can control an onboard light source and possesses directional light detection capabilities. The connection weights of a fully connected feedforward neural network controller with a predetermined connection structure were evolved. A signalling strategy emerged in a variety of experimental scenarios where agents signal to one another via their light source when they find the food (or poison in some cases). Agents evolve to be attracted to (or, in the case of poison signalling, repelled by) each other's light source. This type of behaviour is most likely to arise when groups are composed of clones of a single agent and can be successfully transferred to physical robotic hardware [52].

A similar type of scenario as the one described above was used by Wischmann and Pase-

mann in [160]. Simulated robots must find a food source and groups are composed of clones of a single genome. The main differences from the runs above were that the neural network structure was evolvable, and the agents used sound (with directional sound sensors) instead of light for communication. Furthermore, an incremental evolution approach was used. Robots evolved to emit sounds upon discovering the food source and to be attracted to these sounds when not on a food source.

Another common experimental setup used by Marocco, Nolfi, and others appears in [100], [154], [99], [98] and [36]. A team of simulated robots must coordinate between themselves so that half of the team resides in one of two target areas, while the other half of the team stays in the other target area. Variations include having two agents coordinate to be in the same target area (e.g., [154]) and requiring that two agents occupy separate target areas while earning additional rewards for swapping target areas [36]. Teams are composed of clones of a single genome. Agents are recurrent neural networks with a predetermined structure and can output numerical communication values. Agents also have four directional communication inputs that can receive communication values from other nearby agents. A communication scheme emerges in these simulations where several distinct signals are used for interagent coordination. Results from [36] were successfully transferred to physical robotic hardware.

A commonality across all of the experiments described thus far is that agents have *directional* sensors for receiving potential communications. Thus, it is difficult to determine how much information is being transmitted using an evolved communication scheme (i.e., via the information content of the signal) versus how much information is being extracted from the directional properties of the signal itself.

A binary, nondirectional communication channel was used in experiments by Trianni and Dorigo [150] where four clones of a feedforward neural network with prespecified structure were tasked with moving their physically connected simulated robots in a coordinated fashion while avoiding holes in the terrain. A signal emerged that was used to indicate that one of the robots had detected a hole, and thus the group should change directions. It was also found that evolved solutions out-performed hand-coded communication protocols. Successful controllers were ported to s-bot hardware.

In similar work by Ampatzis, Tuci, Trianni and Dorigo [4], a binary, nondirectional communication channel was employed in an experiment where two clones had to adapt their behaviour depending on which environment they found themselves in. Agents were CTRNNs with prespecified structure that operated simulated robots. The environment contained either a full painted circle on the ground, or one that was broken by a "way in." Agents could detect the colour of the ground and thus had to integrate their sensor inputs over time to determine whether or not there was a "way in." Agents evolved to signal if they have deter-

mined that there was no way in, sharing this informing with the other agent (who had not yet come to this conclusion). Furthermore, it was concluded that the evolution of signalling was based on already evolved cognitive functions. Successful controllers were again ported to s-bot hardware.

Experiments by Pugh, Goodell and Stanley in [127] compared the evolvability of communication using signals with and without directionality. A group of five clones evolved using HyperNEAT were tasked with collecting food in a 2D world. Food would only be collected if at least three agents were touching it simultaneously. The results demonstrated that communication regularly evolves when the communication signal contains inherent directional information, but no interesting communication schemes evolved in the cases without signal directionality. This is not surprising, however, as the agents did not know their own locations in the world, nor were they allowed recurrent neural connections, so they could not remember how far they have moved, nor in what direction. Thus, in the nondirectional cases, it is difficult to envision what useful information the agents could communicate to one another. In a separate set of experiments by D'Ambrosio et al. [31], however, a simple nondirectional signal to determine when a group of clones should leave their positions at a wall (in an effort to achieve synchronization behaviour) was evolved using HyperNEAT. Top solutions from both papers were embodied on real Khepera III robots.

"Acoustic coupling" experiments were done by Di Paolo in [39]. Agents are simulated robots controlled via fully connected CTRNNs with a prespecified structure. Sound is modelled as an instantaneous, additive field with intensity that decreases with the square of the distance from the source and agents possess two nondirectional, physically separated sound sensors. The task is for two simulated robots to approach one another and remain within four body radii of each other. Agents are given individual fitness values averaged over interactions with 10 randomly selected partner agents. A behaviour emerges from these simulations where an agent rotates while moving to determine the direction of a partner's sound. After determining sound direction and using this information to approach one another, the two partner agents engage in "acoustic coupling" where their signals become phase-locked at near anti-phase, with corresponding movement coordination.

Tuci, Ampatzis, Vicentini and Dorigo used the same model of sound as Di Paolo, described above, in experiments where a heterogeneous team of three simulated robots were evolved as one homogeneous controller [151]. A single fully connected CTRNN with prespecified structure was copied into two robots that could detect obstacles but not light, and one robot that could detect light but not obstacles. All robots could emit and detect sound. The team was tasked with reaching a light source that was placed in one of two ends of a simple maze and were penalized for collisions and for not staying close together. Teams evolved such

that all three robots emitted high intensity sounds and then exploited the physics of the simulation of these sounds to coordinate their actions.

In [128], Quinn devises a scenario where two simulated robots must move at least 25 cm from their initial positions without straying more than 5 cm from one another and without colliding. Agents are tested across many different initial conditions and many different partners, with a fitness function penalizing collisions and timesteps where the distance between the two robots exceeded 5 cm. A communication strategy based on the robots' proximity sensors emerged whereby agents would move towards and away from each other to determine "leader" and "follower" roles. These experiments differ from many of the others in that no explicit communication channel was provided *a priori*. Instead, agents evolved to communicate via proximity sensors whose primary purpose was maintaining interagent distances.

Other experiments in which communication emerges without a dedicated signalling mechanism are presented by Williams et al. in [158], with further work done by Manicka in [95]. Two agents, a sender and a receiver, are encoded on a single genome. Both agents exist on a one-dimensional circular track and can pass through each other. Agents are represented as CTRNNs with a prespecified structure and can detect each other through angular distance sensors. Only the sender can detect the target area on the circle, however the fitness of the sender/receiver pair comes from the ability of the receiver to find and remain in the target area. Distinct behavioural patterns emerge to allow the sender to communicate target locations to the receiver via the agents' proximity sensors.

Many of the experiments described above use explicit group selection; that is, the GA selection mechanism is applied to groups of agents instead of individuals. Whether this form of selection plays a part in natural Darwinian evolution is a hotly debated issue (see, e.g., the article by Nowak, Tarnita and E.O. Wilson in [120] and the reply with 136 scientists listed as authors in [1]).

7.2 Communication in Artificial Life

The field of Artificial Life (ALife) is broad and highly interdisciplinary. The term is attributed to Chris Langton [90], who organized the first workshop on the topic in 1987. In announcing the workshop, Langton defined Artificial Life as follows [10]:

> *Artificial life is the study of artificial systems that exhibit behaviour characteristic of natural living systems. This includes computer simulations, biological and chemical experiments, and purely theoretical endeavours. Processes occurring on molecular, cellular, neural, social, and evolutionary scales are subject to investigation. The ultimate goal is to extract the logical form of living systems.*

Here, we will focus on the computer simulations aspect of ALife.

The following ALife computer simulation experiments differ from the Evolutionary Robotics simulations described above in two notable ways: There are no fitness functions and no discrete generations. Agents are continually being born and dying off with reproduction integrated directly into the simulation world. ALife simulations are generally more focused on scientific rather than engineering results.

There have been a few ALife simulation experiments exploring the emergence and evolution of communication. In the work of Werner and Dyer [157], blind but mobile "male" agents and stationary but sighted "female" agents live in a grid world and must reach the same grid location to produce one male and one female offspring. Two random agents are removed from the simulation to make room for the new offspring. Agents are represented as fully connected recurrent neural networks with prespecified structure. Female agents can send three communication bits per timestep to nearby males and eventually a signalling scheme evolved whereby females signal to a male to turn left or right based on his location relative to hers. The signals are nondirectional and not attenuated by distance, so males cannot extract any useful information from the properties of the signal itself; all information must come from their interpretation of the content of the signal.

More recently, the AVIDA platform [92, 121] has been used by Knoester et al. to simulate the emergence of communication [84]. The agents are evolvable computer programs (i.e., instruction sets) arranged in a grid pattern that must compete for virtual CPU cycles (which can be used for reproduction) by accomplishing simple experimenter-specified communication tasks. Although no fitness function is used, the evolution is directed by the choice of communication task.

Chapter 8

NoiseWorld

8.1 Introduction

We now present a novel simulation world, *NoiseWorld*. The idea behind *NoiseWorld* is to develop a simple simulation within which complex communication emerges from populations of initially noncommunicating agents. One of the key benefits of such a simulation would be the ability to study the emergence and evolution of communication via digital fossil records, making up for the lack of fossil records of the emergence of communication in nature.

The experiments reported here focus on simplicity, as it is only after determining the fundamental mechanisms of the emergence of communication that more complex simulations, e.g., involving predator/prey models, food and energy, and higher social capabilities, should be attempted. *NoiseWorld* is provided as open-source software so that the effects of additional complexities can be studied in the future.

The emergence and evolution of interagent communication is a complicated process, as a mutation that increases the information content of a sender's signal will only be beneficial if a separate listener can interpret the new information in a reproductively beneficial manner. This is especially complicated when communication has yet to emerge within a population. What benefits can a sender receive from producing meaningful communications if no one is listening? Conversely, it would likely be detrimental for an agent to react to what it is hearing if no meaningful communications are being produced. In the words of John Maynard Smith, "it's no good making a signal unless it is understood, and a signal will not be understood the first time it is made" [140]. Simulation experiments often avoid this issue by having interacting agents be clones from the same genome. Thus, a mutation in a sender will simultaneously appear in the receiver. This is not biologically plausible, however, as in nature interacting agents are rarely genetically identical. Furthermore, in many results in the literature (see Chapter 7), a certain amount of information is inherent in the signal itself,

such as how far it has travelled from the sender. This makes it difficult to determine how much information is being gleaned by a listener from the *content* of a signal and thus the level of complexity of the evolved interagent communication scheme. Finally, most simulated evolution thus far has occurred in discrete generations. This removes the requirement that communication schemes evolve in a continuous fashion, as the communications of the offspring need not be understood by their parents, and vice versa.

Biological plausibility is difficult to define and achieve when dealing with ALife simulations. There is one obvious component, however, and that is the transferability of the simulation world into physical reality. This is addressed through the embodiment of simulated agents in physical robotic hardware (see Chapter 10). Unfortunately, the technology to create self-replicating robots does not yet exist (although there have been some interesting advances recently, see, e.g., [163, 164]), therefore physical replication cannot be implemented at this time.

Another important element in the quest for biological plausibility is the minimization of experimenter bias. This is achieved in several ways in *NoiseWorld*. There is no fitness function, therefore the experimenter has no direct influence on the reproductive viability of the agents. Furthermore, EMMs are used, thus removing the experimenter bias that arises from prespecifying the artificial neural structures of the agents. The group selection debate is also side-stepped in the *NoiseWorld* simulation, as agents are not forced to interact with clones of themselves. Agents interacting within *NoiseWorld* are usually genetically similar (or identical), which is similar to local subpopulations of biological agents and thus *NoiseWorld* may help inform the group selection debate without explicitly choosing a side. Finally, the lack of discrete generations offers another improvement over the biological plausibility of the ER experiments, as the evolving communication scheme must remain simultaneously understandable by both parents and offspring.

As we are creating an artificial world, some experimenter bias is unavoidable. For example, we need to specify a coordinate system. Giving agents access to their absolute position information is detrimental to the biological plausibility of the simulation, however, it is not without precedent in nature. Honey bees, for example, are able to determine the distance travelled from the hive and their direction relative to the sun [155]. Furthermore, we allow for nondirectional signalling, a physical impossibility and thus another point against the biological plausibility of the simulation. This was deemed necessary to avoid having agents home in on signals directly, and thus to encourage the emergence of a more complex communication strategy. We must also define the size of an agent and the speed at which they move. Obviously, these and other parameters (see Section 8.3 below) will influence the evolution of the agents within this environment. Avoiding these biases is not straightforward.

The artificial world will now be described, followed by the simulation details and experimental parameters chosen for the experiments presented here. Results from these experiments will then be presented, with several top evolved agents that emerged from these simulations being described and two different digital fossil records being examined. Embodiment (i.e., hardware) experiments will be presented in the following chapter.

8.2 The Artificial World

NoiseWorld is a two-dimensional toroidal simulation world containing a number of islands (Figure 8.1), each with a fixed number of agents that can move freely in any direction on their island. Islands are infinite in both directions in both dimensions (i.e., $-\infty < x < \infty$ and $-\infty < y < \infty$) and are quasi-reproductively isolated, ensuring that agents are interacting with kin, which is theorized to be important for the emergence and stability of cooperative communication (see, e.g., [2, 122, 97, 103, 52], although Di Paolo warns against assuming that this is owing to kin selection [38]). If two agents meet (i.e., come within a specified distance of one another, the "reproduction distance"), they reproduce, yielding one offspring agent (Figure 8.2). To keep the population sizes constant (which in turn keeps other simulation parameters constant, such as expected distance between neighbours), one randomly selected agent "dies" for each agent "born." There is a small chance that offspring will be born on a neighbouring island. Agents are born on the timestep after their parents reproduce (except in the case of migration, see Section 8.3) and die when they are randomly chosen to be replaced by a newly created agent. An agent is born at a randomly assigned position on its parents' island (or a neighbouring island in the case of a migration event) with an enforced minimum distance between the new agent and its nearest neighbour (this distance is the same as the "reproduction distance"). After a reproduction event, both parent agents are moved to new random positions on their island and restarted. Selection pressure arises from the fact that higher reproductive rates lead to lower average agent lifespans, since each birth leads to a randomly selected agent's death. This decrease in average lifespan will force less reproductively viable genomes out of the population. Thus, there are no discrete generations, no explicit fitness function and no explicitly enforced group selection.

Each agent is represented by an Evolvable Mathematical Model (EMM), a series of difference equations, embedded as an equation tree in its genome. Each agent is moreover aware of its geographical position on its respective island, given by orthogonal coordinates x and y, as well as its orientation θ relative to a specified direction, which defines its instantaneous direction of motion. It has no information about any other agent. However, it can "listen" on the real-valued communication channel from its nearest neighbour as well as emit a signal

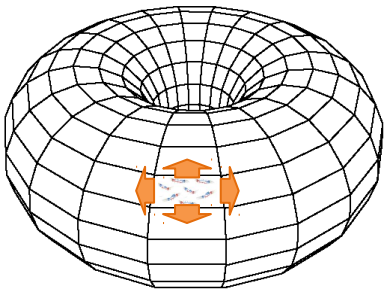


Figure 8.1: Arrangement of 100 islands in the toroidal *NoiseWorld* simulation. Arrows show possible migration routes outward from a given island for the experiments presented here.

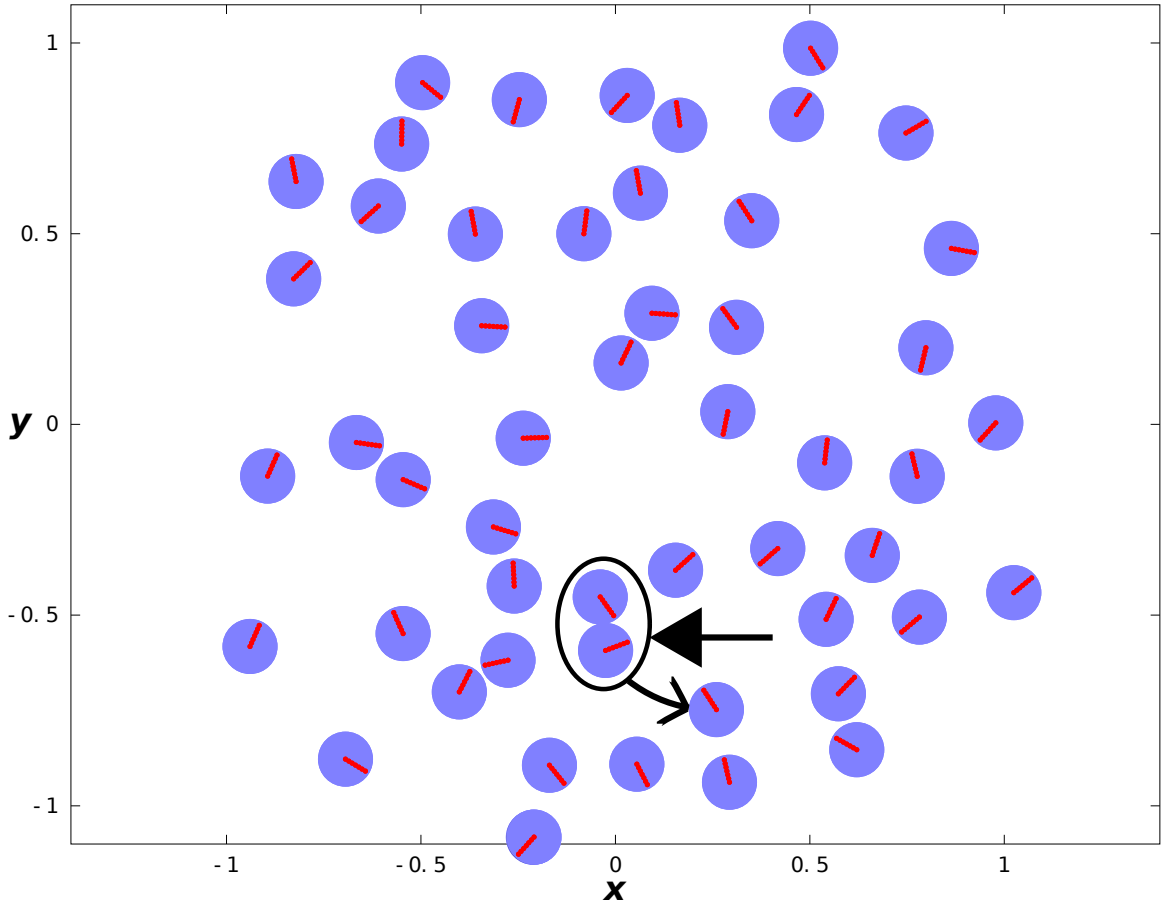


Figure 8.2: A reproduction event on a single 50 agent island in a *NoiseWorld* simulation. The distance between the centres of the two circled agents is less than the reproduction distance, so the two agents become parents and produce one offspring. After the reproduction event, all three agents (the two parents and their one offspring) are moved to new random locations.

ω on its own channel (Figure 8.3). Agents cannot determine the direction from which signals are emitted nor are signals attenuated by distance. This is to prevent agents from being able to home in on communication signals directly.

Time in *NoiseWorld* is discrete and, at each timestep, an agent moves one unit of dis-

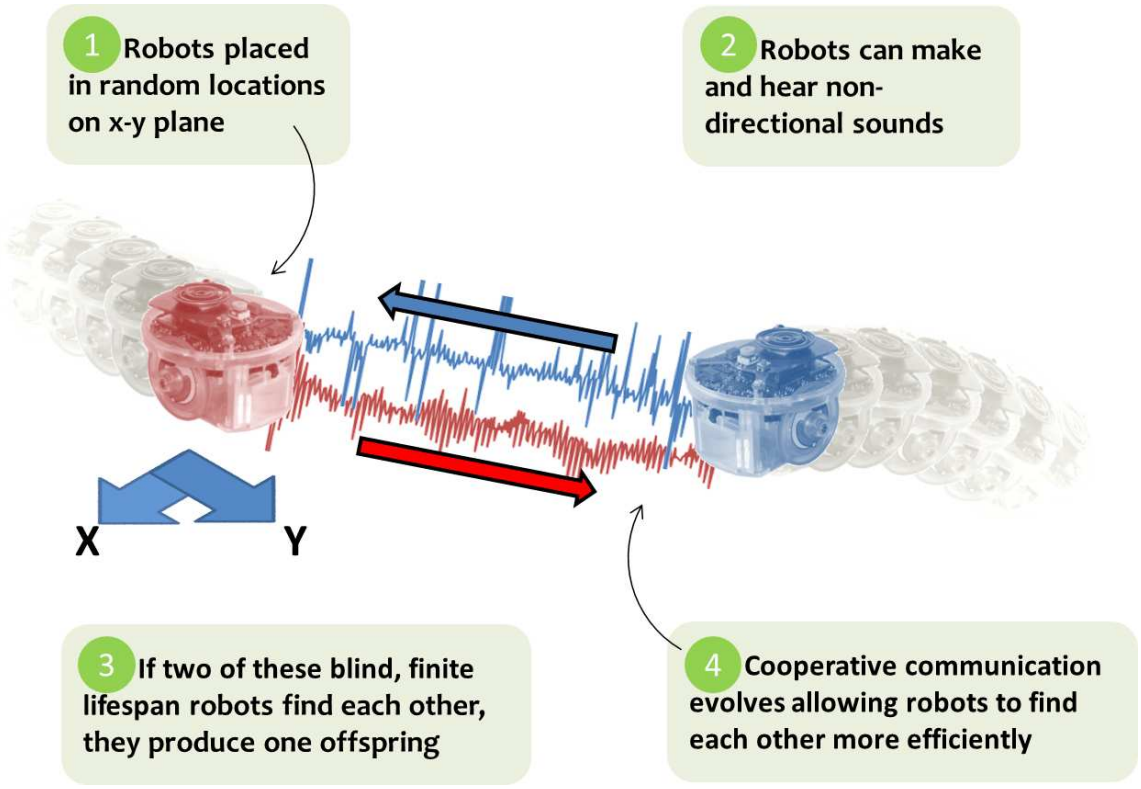


Figure 8.3: Details of the *NoiseWorld* simulation world. The robots shown in this diagram are e-puck robots, which are used in the hardware experiments.

tance (modified by Gaussian noise) in direction θ (also modified by Gaussian noise). For convenience, we define an *era* as 100,000 timesteps. Offspring are produced by randomly combining the genomes from the two parents, with various mutations occurring with small probabilities. These recombination and mutation operations, as well as the initialization method, are the same as those described in Section 4.2.2 and will be summarized below.

Initial agents, as well as random subtrees for subtree mutations, are generated using the ramped half-and-half method using maximum depths of 1 or 2 with equal probability. If a tree mutation is to occur, it can be a point mutation or a subtree mutation. If a point mutation is to occur and if the tree in question contains one or more constants, there is a probability that the mutation will be a perturbation of a constant. In this case, a random value will be added to a randomly selected constant. If a perturbation of a constant was not performed, a random node in the tree undergoing mutation will be selected. If this node is a nonterminal node, it will be changed to a new, randomly chosen operation (e.g., from addition to multiplication). If this node is a terminal node, it will either be changed to a randomly selected variable from $\mathbf{u}$ and $\mathbf{v}$ or it will be changed to a randomly selected real-valued constant.

If a subtree mutation is to occur, a randomly selected node on the tree in question is

replaced with a randomly generated tree. With a small probability, the roles of the subtree and the original tree are swapped, i.e., a random node on the randomly generated tree is replaced with the entire original tree and this becomes the new tree of the offspring.

Every offspring is produced via equation-level sexual recombination, receiving at least one equation tree from each of its two parents. This is always possible, as agents have two outputs and thus always have at least two equations in common (the two equations that modify these two outputs). This differs from the implementations in Part I where equation-level sexual recombination would only occur with a certain probability and only if the two parents had more than one equation in common. This modification was made to improve biological plausibility in our *NoiseWorld* simulations, as offspring in nature contain genetic material from both of their parents. Tree-level recombination cannot be employed for this purpose, as it is a disruptive operation that is usually highly detrimental for the reproductive viability of the offspring, and thus must be applied sparingly to maintain reproductively viable populations.

Tree-level sexual recombination will occur in an equation tree with a given probability. Here, a randomly selected node will be replaced by a subtree randomly selected from all subtrees in a parent genome. When choosing subtrees during tree-level recombination operations, there is a chance that the root node will be a terminal node, otherwise it will be a nonterminal node. Since this subtree can come from any one of a parent's equation trees, this operation allows for genetic material to travel between trees.

An offspring is subject to an "add equation" mutation with a certain probability. This mutation produces a new randomly generated equation tree with a randomly generated initial condition. A new variable v_j, which is what this new equation tree will be modifying, is added to $\mathbf{v}$. The new variable v_j is also randomly incorporated into an existing equation, either through a point mutation or a subtree mutation, as described above. Finally, $v_j^{t=0}$ is set to a random value.

An equation's initial value $v_i^{t=0}$ is also subject to mutation. When such a mutation occurs, the value is either set to a new random value, or it has a random value added to it.

A multiplier (β) is used to scale mutation and crossover probabilities to keep the expected number of genetic operations per unit time on each island constant. Without using β, an island population with a high birth rate will experience more mutations and tree grafting operations per timestep than a neighbouring island with a lower birth rate, owing to the fact that genetic operators are applied when creating offspring genomes.

An agent's genome is allowed to have an experimenter-defined maximum number of nodes across all of its equation trees. If an offspring is born with more than the maximum number of nodes, it dies immediately. If one or more of an agent's output variables exceed the minimum

or maximum representable floating-point number, the agent will have that output set to a random value and will be selected to die when the next birth occurs.

In the simulation world, agent controllers have three inputs and two outputs, all real-valued and unbounded. The inputs are the agent x and y positions, and the communication channel input ω_{in} (i.e., the unbounded communication value from the agent's nearest neighbour). The two unbounded outputs are θ (in radians) and the communication channel output ω_{out}. All inputs and outputs have Gaussian noise applied to them at each timestep. Thus at each timestep, an agent knows its current x and y position, as well as the communication output of its nearest neighbour from the previous timestep. It can then use this information to update its outputs (via its EMM) to determine its orientation for the next Δt step, as well as what it will "say" next on its communication channel. The new x and y positions of an agent are determined from its outputs as follows:

$$x^{t+\Delta t} = x^t + \Delta t(S\cos\theta^t) \tag{8.1}$$

$$y^{t+\Delta t} = y^t + \Delta t(S\sin\theta^t) \tag{8.2}$$

where S is the fixed, experimenter-defined agent speed. Gaussian noise is applied to S and θ^t for each agent at each timestep. Note that the agents do not have any information as to the whereabouts of their neighbours; they are in essence "blind."

8.3 Experimental Parameters

Hundreds of *NoiseWorld* simulation runs were executed to determine parameter settings that would repeatedly produce high quality results. The parameters that we settled upon will now be presented. Note that while high-level mutation and recombination rates are different owing to the vast differences in the experiments, many of the parameters affecting the internal workings of these operations are identical to those used in the benchmarking runs (see Section 4.3).

For all experiments presented here, the *NoiseWorld* simulation contains 100 islands of 50 agents each. Thus, the total number of agents per *NoiseWorld* run is $100 \times 50 = 5{,}000$. An offspring is born on a neighbouring island with a probability of 0.001 (migrants are not born until after an island synchronization/migrant exchange, see below). The island for migration is randomly selected from the four islands that are connected to the island of the offspring's parents. Diagonal migrations are not allowed (see Figure 8.1). Whenever random positions are assigned to agents, they are always in the range $-1 < x < 1$, $-1 < y < 1$ and the position must be at least 0.139 units from all other agents. This distance of 0.139 units is

the "reproduction distance," i.e., if the centres of two agents are less than 0.139 units apart, the two agents will reproduce. We set the speed of all agents in our simulation runs to be $S = 0.0005$ perturbed with random values drawn from a Gaussian distribution with mean 0 and standard deviation 1.25×10^{-5}.

Random constants are chosen from the uniform distribution $[-5, 5]$ and random initial values $\mathbf{v}^{t=0}$ are selected from the uniform distribution $[-1, 1]$. When applying noise to inputs and outputs, random values are drawn from a Gaussian distribution with mean 0 and standard deviation 0.025.

A mutation will occur in an equation tree with a probability of $P_\mu = \beta\,[0.0025/(N + N')]$, where $N + N'$ is the number of trees in the genome and β is independently calculated on each island every 10,000 timesteps as $500/\gamma$, where γ is the number of births in the past 10,000 timesteps on the island in question. The β multiplier has an enforced maximum value of 100 and no minimum value.

A tree mutation will be a point mutation with a probability of 0.5; otherwise, it will be a subtree mutation. Assuming a tree chosen for a point mutation has at least one constant, the point mutation will perturb a randomly selected constant with a probability of 0.5 and using a random value drawn from a Gaussian distribution with mean 0 and standard deviation 0.5. If a terminal node is to be mutated, it will be changed to a new variable or a new constant with equal probability.

During subtree mutation, the roles of the two trees are swapped with a probability of 0.05.

Tree-level sexual recombination occurs with a probability of P_μ. When choosing root nodes for this operation, there is a 10% chance that a root node will be a terminal node, otherwise it will be a nonterminal node.

An offspring is subject to an "add equation" mutation with a probability of 0.0025β and has a 10% chance of undergoing equation reductions.

An equation's initial value $v_i^{t=0}$ is mutated with a probability of P_μ. When such a mutation occurs, 50% of the time the value is set to a new random value, otherwise it has a random value added to it. This random value is taken from a Gaussian distribution with mean 0 and standard deviation 0.25.

Agents can have a maximum of 200 nodes across all of their equation trees.

All simulation experiments were run for 48 wall clock hours on a dedicated Linux server with an Intel Xeon E5540 at 2.53GHz. Owing to the stochastic nature of the simulation and to variable genome sizes (larger genomes take more computation time to evaluate), the number of timesteps per 48 wall clock hour simulation varies. Each island is implemented as a separate process so that the algorithm can take full advantage of the parallel architecture

of the Intel Xeon CPU (8 cores/16 threads). A master/slave parallel implementation is used, where a "master" process handles the synchronization of "slave" processes (i.e., the islands). Islands are synchronized and migrants exchanged every 10,000 timesteps. Islands introduce incoming migrants into their subpopulations at a rate of ξ/10,000 migrants per timestep, where ξ is the number of migrants received at the previous synchronization. Computations were performed on the GPC supercomputer at the SciNet HPC Consortium [94].

Chapter 9

Results and Discussion

9.1 Experimental Results

Three sets of 20 simulation runs were executed using the parameter settings described above. Each of the 20 runs in a given set had different initial populations, but initial populations were consistent across the sets of 20 runs (i.e., the initial population of run 1 in the first set of experiments is the same as the initial population in the first run of the second set of experiments, etc.). The first set of 20 runs was the canonical run, set up as described in previous sections. The second set of runs was identical to the first, except that $\theta = 0$ was aligned with the positive y axis instead of the positive x axis. Finally, the third run is identical to the first, except that equation-level sexual recombination was disabled. We call any given simulation run a "successful run" if one or more islands achieve a maximum reproductive rate of more than 22,000 births per era at some point within its allotted 48 wall clock hours. Over the course of many simulation runs, only subpopulations that explicitly communicated both their x and y positions were able to exceed this reproduction rate, i.e., a subpopulation that only explicitly communicates about either its x or y position was never seen exceeding 22,000 births per era. Table 9.1 summarizes the results from these experiments.

Table 9.1: Summary of *NoiseWorld* experiments. All results are from 20 runs with different initial populations. We use μ to denote mean and σ to denote standard deviation.

Experiment	Best Repro. Rate		# Eras		# Success
	μ	σ	μ	σ	
Canonical	20,949	2,205	1,341	247	4
$\theta = 0$ aligned with +ve y	20,290	1,798	1,330	323	2
No Eqn-level crossover	18,057	3,233	1,090	194	2

While the runs using equation-level sexual recombination had high top reproduction rates

on average, only 15% of them managed to be successful, i.e., to exceed a reproduction rate of 22,000 births per era. This suggests that communication regularly emerges within *NoiseWorld*, but that the transition from communicating about a single variable to communicating about two variables is rare. This should come as no surprise, however, as agents are communicating over a 1D channel and thus significant complexification is required to transition from simply broadcasting one of your position variables on your channel (see, e.g., snapshot II from run 1 and snapshots IV and V from run 2, below) to being able to encode and decode two-dimensional position information (see, e.g., snapshot III from run 1 and snapshot VI from run 2, below).

Even though equation-level sexual recombination was implemented primarily for improved biofidelity, these experimental results also demonstrate another benefit: smaller genomes and thus lower computational complexity. Over the course of 48 wall clock hours, simulation runs with equation-level sexual recombination disabled execute only about 81% of the number of timesteps as the canonical runs, on average. Thus there is increased computational complexity within these runs, and seeing as they average less reproduction events and perform less operations per reproduction event (as no equation-level recombination is being performed), the increased computational complexity must be coming from within the agents themselves. This makes sense, as in the cases with forced equation-level recombination, if two parents have large genomes and one or more different extra equation trees, their offspring could potentially inherit a large genome as well as some or all of the extra equations unique to only one of the parents. Thus, such offspring run the risk of exceeding genome size limits and dying prematurely. This engenders a selection pressure for agents to have compact genomes so that they can produce reproductively viable offspring. No such pressure exists in the runs without equation-level recombination, as offspring will only inherit the extra equations of a single parent. Agent genomes are thus free to grow to near the maximum allowed size.

9.2 Emergence and Evolution of Communication

We show data from two different simulation runs of *NoiseWorld*, one from a run with $\theta = 0$ aligned with the positive x axis (which we will henceforth call "run 1") and one from a run with $\theta = 0$ aligned with the positive y axis (henceforth called "run 2"). The reproduction rates over the course of these two runs are shown in Figures 9.1 and 9.2, respectively. The reproductive viability of the population is increasing over time; thus we have evolution. Furthermore, the island populations seem to be evolving in manner consistent with the theory of punctuated equilibria, which proposes that species will exhibit little evolutionary change for most of their history, punctuated with rare speciation events that occur relatively rapidly

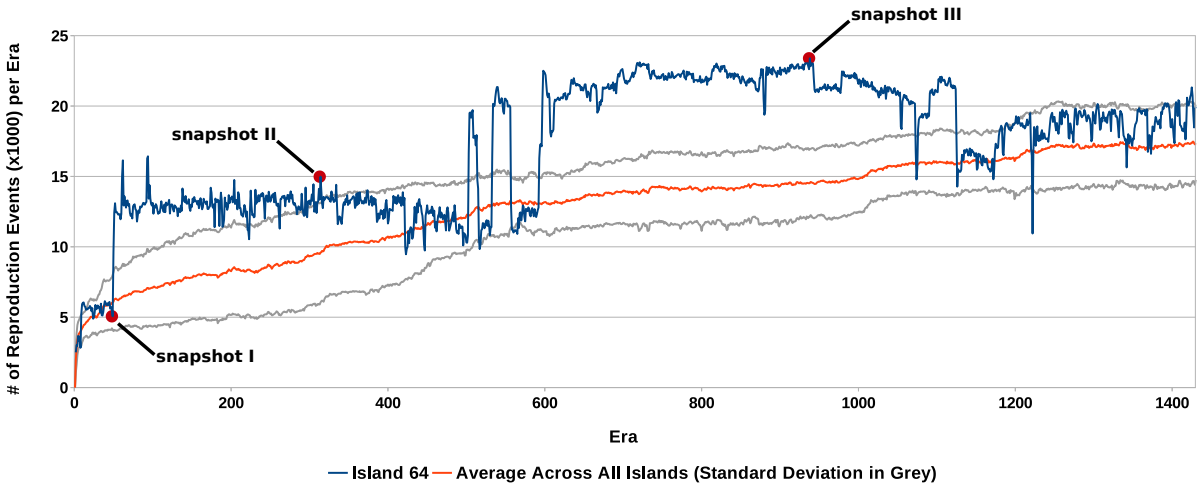


Figure 9.1: The reproduction rate (number of reproduction events per era) of the top island from run 1 over the course of the simulation. Note that all major plateaus (i.e., 1-49 eras, 50-591 eras, 592-1124 eras and 1125-1430 eras) are statistically distinct (pairwise Wilcoxon rank-sum tests, all $P < 0.00001$). Also shown is the average and standard deviation of the reproduction rate across all 100 islands in the simulation. The three red dots (I, II, III) show the three population samples that were selected for further analysis.

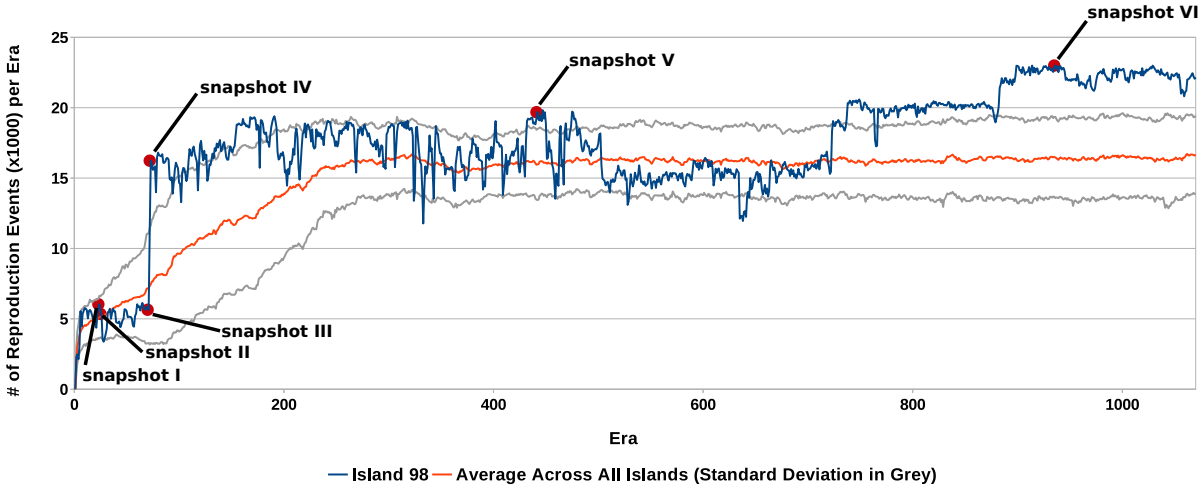


Figure 9.2: The reproduction rate (number of reproduction events per era) of the top island from run 2 over the course of the simulation. Also shown is the average and standard deviation of the reproduction rate across all 100 islands in the simulation. The six red dots (I, II, III, IV, V, VI) show the six population samples that were selected for further analysis.

[46]. This is contrary to the theory of phyletic gradualism, which proposes that speciation is slow and uniform.

While increased reproductive viability demonstrates that the agents are evolving over time, it does not necessarily indicate that communication has evolved. In Figures 9.3 and

9.4, snapshots of the same island populations from various epochs (marked in Figures 9.1 and 9.2 with red dots) are run for an era during which evolution is suppressed (i.e., there are no births or deaths, although to simulate the effects of births and deaths, there is a small probability that an agent will be moved to a new random position and restarted) and the communication channel is, by turn, disabled (signals are zero with Gaussian noise), randomized (signals are randomly chosen from the range $[-1, 1]$ at each timestep) and fully enabled.

These data were collected from a test simulation. Test runs last for 1 era (100,000 timesteps) and use genomes from a snapshot of an island population (i.e., an island's 50 genomes at a given timestep during the main *NoiseWorld* simulation). The agents are initially placed randomly in the test world and initialized. If during the test simulation two agents encounter one another, the event is counted as a reproduction event, but no offspring genome is created. Instead, the two parent agents are moved to new random locations and reinitialized. This prevents any evolution during the test simulations. The effects of births and deaths were simulated by moving an agent to new random position and reinitializing it with a probability of 0.001 per agent per timestep.

For any given snapshot test reported, the above test simulation was run 100 times with different random initial agent positions.

At snapshot I in run 1 (Figure 9.3), it is obvious that there is no communication on the island, as an enabled communication channel does not provide any reproductive benefits. However, as time progresses, the channel becomes significantly more important to the population's reproductive success. A beneficial communication scheme has emerged by snapshot II and continues to evolve through snapshot III.

At snapshot I in run 2 (Figure 9.4), it is again obvious that there is no communication on the island, as an enabled communication channel does not provide any reproductive benefits. Interestingly, there are slight differences in reproductive success across the various channel settings at snapshot II, indicating that some agents are modifying their behaviours based on their communication input. By snapshot III, however, this listening behaviour seems to have evolved away, as modifying the communication channel again produces no changes in the island's reproductive success. As time progresses, a beneficial communication scheme emerges and the channel becomes significantly more important to the population's reproductive success.

In an attempt to quantify the amount of information being transmitted between agents, Shannon's information entropy of the top agent's communication output variable was calculated for the various simulation snapshots. For these entropy calculations, the communication outputs of the top reproducing agent in the snapshot are recorded throughout the test sim-

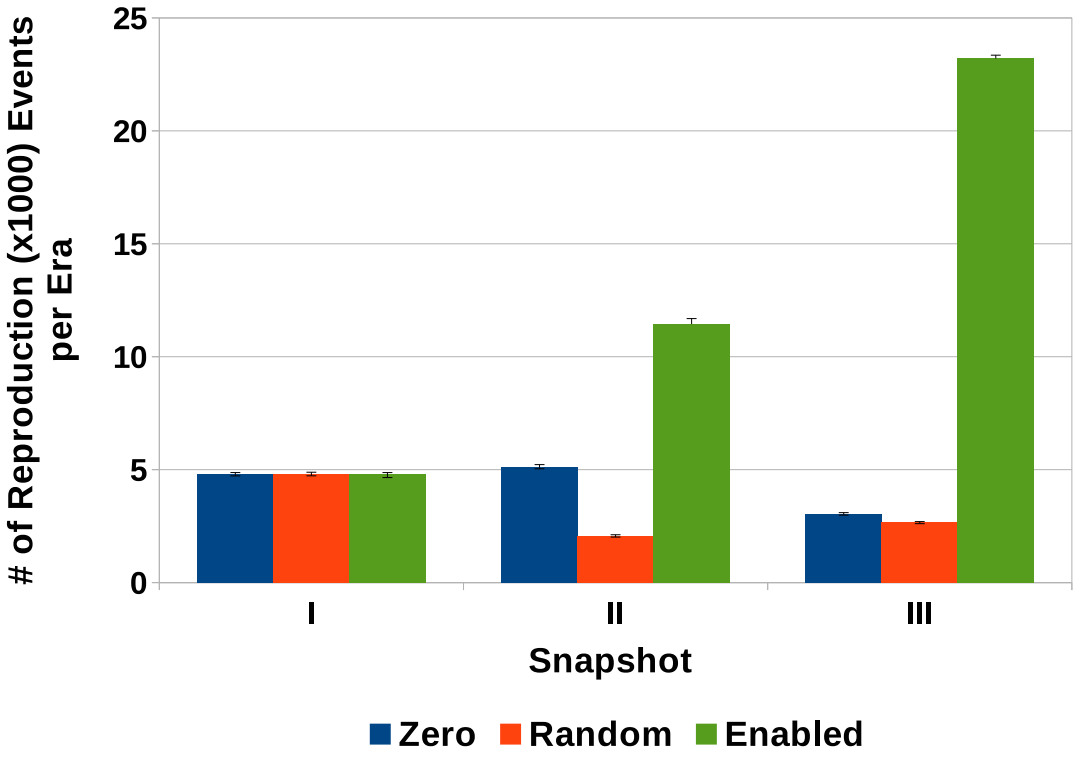


Figure 9.3: Performance (in terms of reproduction rate) of the three population snapshots from run 1 with the communication channel set to 0 with Gaussian noise, uniform random values from $[-1, 1]$ and with the communication channel enabled as in the original simulation. Error bars show standard deviation across 100 runs with different initial agent configurations.

ulation described above, yielding $100,000 \times 100 = 10{,}000{,}000$ real-valued communication output values per test set. The equation for entropy $H(\Omega)$ is:

$$H(\Omega) = -\sum_i P(\omega_i) \log_2 P(\omega_i) \tag{9.1}$$

where Ω is the set of all possible communication outputs and $0 \log_2 0 = 0$.

To evaluate this equation using real data, one must create a probability distribution from the real-valued communication outputs. To do this, a binning technique was used whereby communication values were rounded down to the nearest tenth of an integer. For example, for these calculations, 1.00 and 1.09 are considered the same communication output ω_i, while 1.10 and 1.11 would both be considered communication output ω_{i+1}. All recorded communication output data are rounded in this fashion and then the number of occurrences of each ω_i is tallied. Finally, these tallies are divided by the total number of recorded communication outputs to give the probability distribution $P(\Omega)$.

Figure 9.5 shows the results of these information entropy calculations for the three snap-

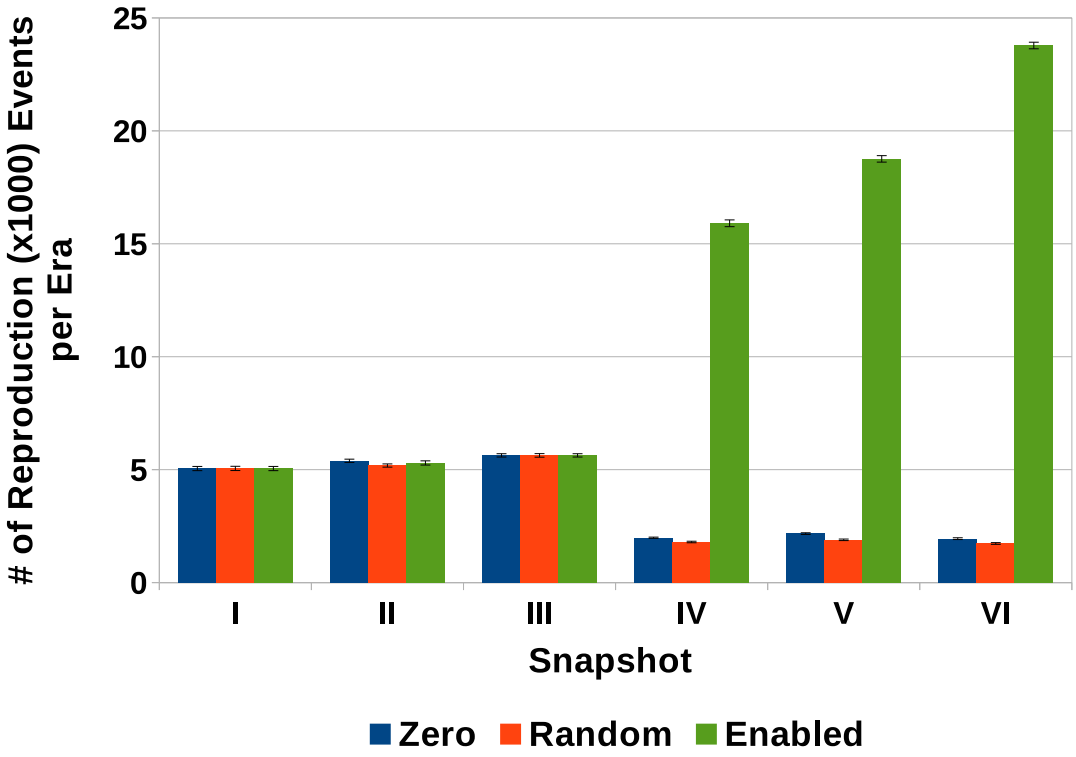


Figure 9.4: Performance (in terms of reproduction rate) of the six population snapshots from run 2 with the communication channel set to 0 with Gaussian noise, uniform random values from $[-1, 1]$ and with the communication channel enabled as in the original simulation. Error bars show standard deviation across 100 runs with different initial agent configurations.

shots taken during run 1. The information entropy of the agent's signals increases as the communication scheme evolves. This indicates an increasing maximum information transmission capacity, but does not yield insight into the content of the signal nor the amount of information being extracted by the receiver [45]. Figure 9.6 shows a similar increase in information transmission capacity of the communication signals of the top agent from the snapshots taken during run 2.

9.3 Evolved Communication Schemes

Let us now examine more closely the evolved communication schemes. Figures 9.7–9.10 show four different test runs with two copies of the top agent from run 1. The two agents are able to find each other from all four starting configurations. In fact, these agents can find each other from any starting configuration (see Figure 9.11). The equations of the agent's EMM

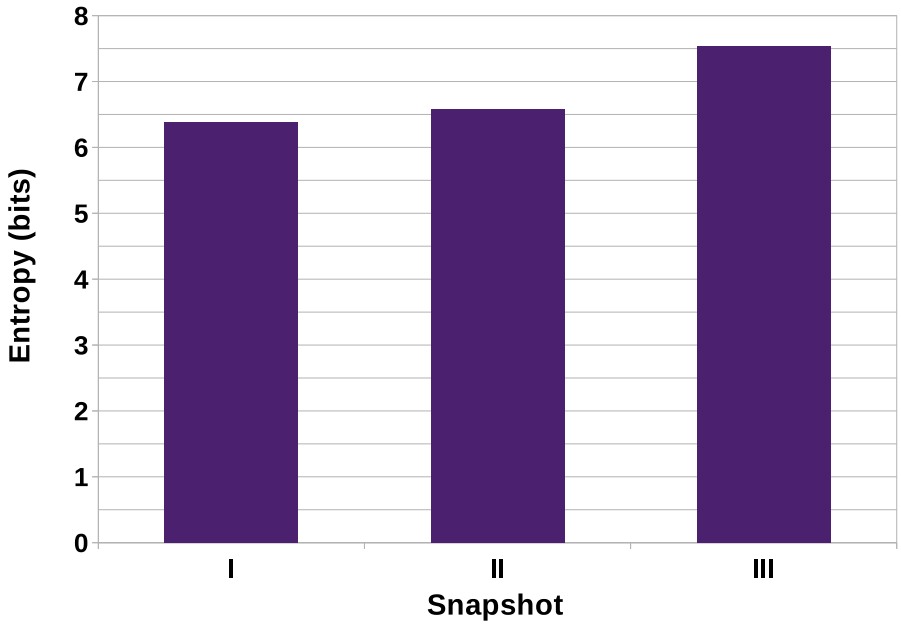


Figure 9.5: The Shannon information entropy of the communication output variable of the top reproducing agent from each snapshot of run 1 using data gathered as it interacts with other agents on its island.

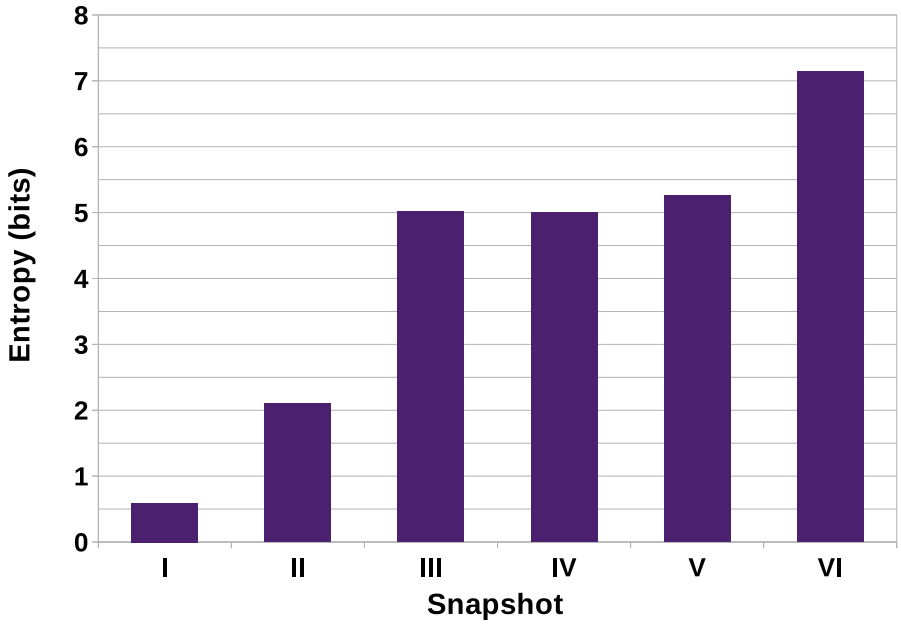


Figure 9.6: The Shannon information entropy of the communication output variable of the top reproducing agent from each snapshot of run 2 using data gathered as it interacts with other agents on its island.

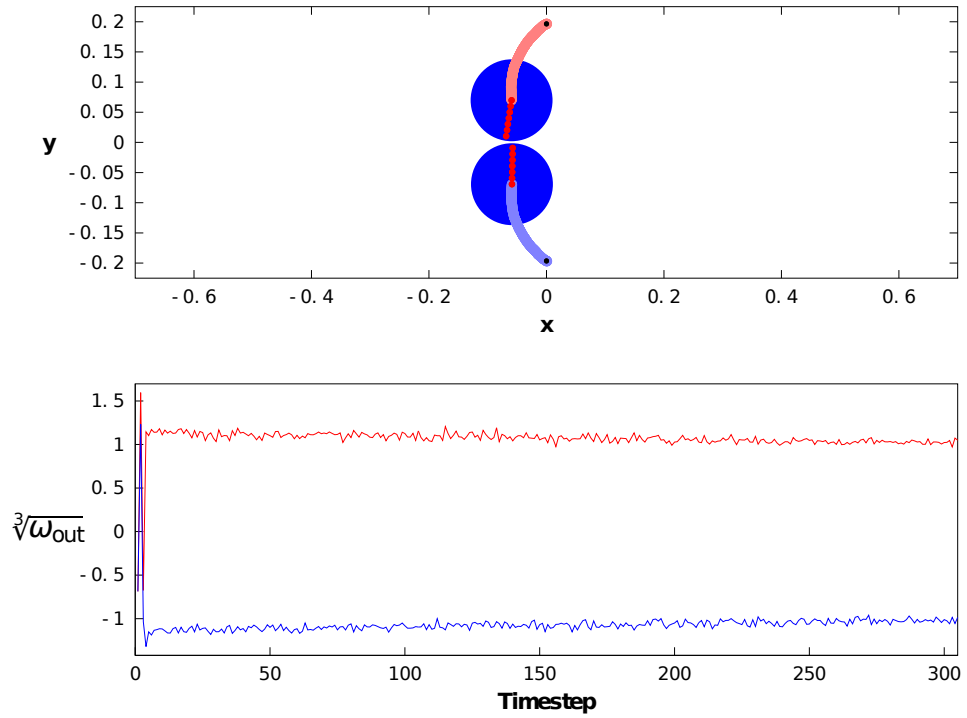


Figure 9.7: A test run using two clones of the top agent from run 1. The top figure shows the movements of the two agents over time, while the bottom figure shows $\sqrt[3]{\omega_{\text{out}}}$ for the two agents over time.

are

$$\theta^{t+\Delta t} = -5.50y^t + \omega_{\text{in}}^t \tag{9.2}$$

$$\omega_{\text{out}}^{t+\Delta t} = 5.50y^t - \frac{1.39}{(\omega_{\text{in}}^t - 5.50y^t)(1.53 + x^t)} \tag{9.3}$$

As agents are provided with no direct information about the whereabouts of their neighbour, any such information must be received via the one-dimensional communication channel. Figures 9.7 and 9.8 show successful test runs from vertically and horizontally aligned starting configurations, respectively. This clearly demonstrates that the agents are able to interpret both x and y position information about their neighbour via communication. The agent's evolved communication equation (9.3) corroborates this claim; it contains both the agent's x and y position inputs. Hence, agents encode information about their positions and transmit it while the neighbouring agents decode it and use it in determining their motions. Furthermore, an agent's output signal is affected by its input signal from its nearest neighbouring agent. Note that ω_{in} appears in the agent's communication output equation (ω_{out}). Thus communication between neighbours is a dialogue.

The evolved communication scheme from run 1 appears to have two modes, "lateral" and

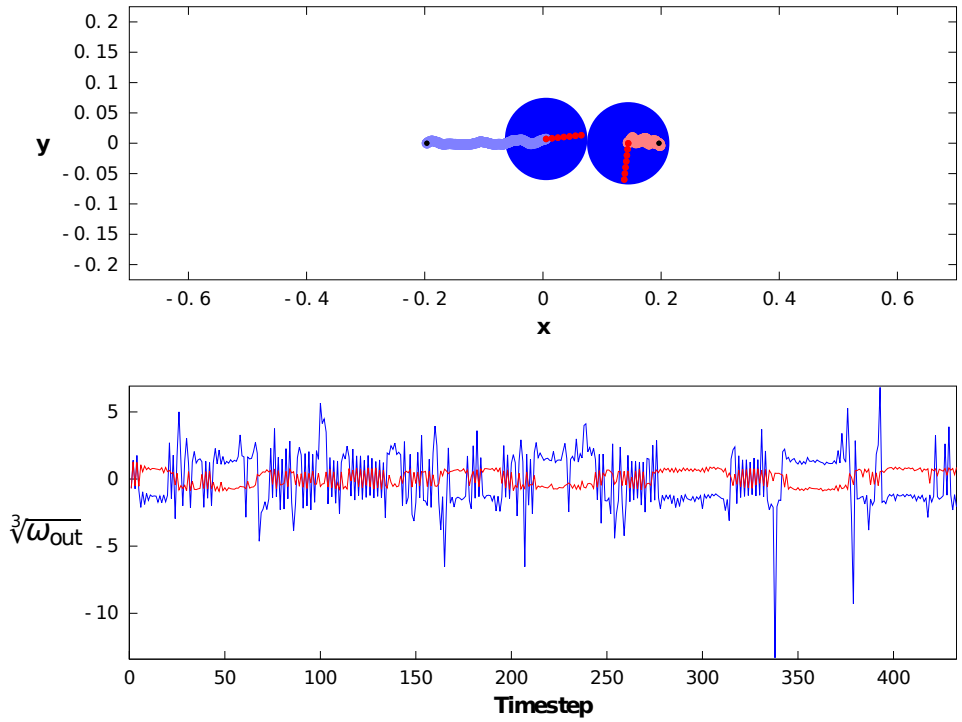


Figure 9.8: A test run using two clones of the top agent from run 1. The top figure shows the movements of the two agents over time, while the bottom figure shows $\sqrt[3]{\omega_{\text{out}}}$ for the two agents over time.

"longitudinal." This reflects how the agents see their world; they locate themselves by a rectangular coordinate system (x and y) and their orientation θ is determined relative to a specified direction (in this case, $\theta = 0$ is aligned with the positive x direction). The longitudinal mode (see Figure 9.7) operates at a low order of magnitude and contains primarily y position information. Furthermore, there is a high degree of symmetry in the communication and movement behaviours of the two agents. On the other hand, agent behaviours in the lateral mode are highly asymmetrical (see Figure 9.8), with one of the communication outputs operating at a different order of magnitude than the other. To better visualize these two orders of magnitude simultaneously, $\sqrt[3]{w_{\text{out}}}$ is plotted in all figures here, as it has the effect of squashing the original signals while preserving their sign.

The two test cases with diagonal starting configurations (Figures 9.9 and 9.10) demonstrate that agents start out in the longitudinal mode and transition to the lateral mode as they converge upon a common y position. It is interesting to note that this is also the order in which these behaviours evolved, as communication about y positions emerged first (see Section 9.4). We also see successful agents that have evolved with the behaviours of their longitudinal and lateral modes swapped (see Figures 9.14–9.17). This is clearly seen in the

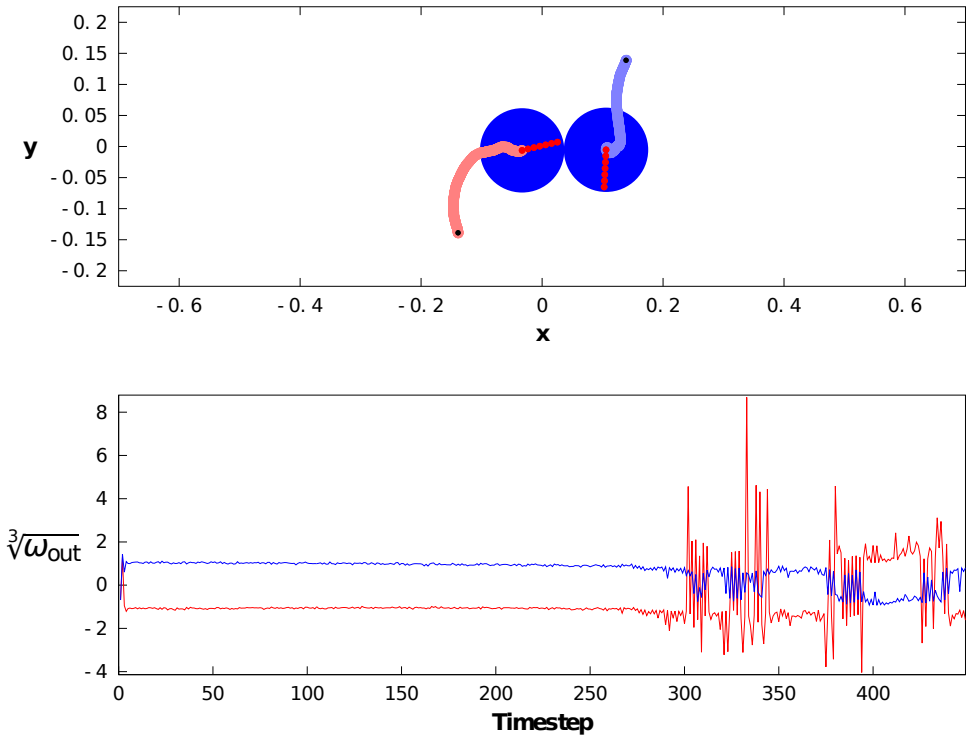


Figure 9.9: A test run using two clones of the top agent from run 1. The top figure shows the movements of the two agents over time, while the bottom figure shows $\sqrt[3]{\omega_{\text{out}}}$ for the two agents over time.

top agent EMM from run 2:

$$\theta^{t+\Delta t} = 3.06x^t - \omega_{\text{in}}^t \tag{9.4}$$

$$\omega_{\text{out}}^{t+\Delta t} = 3.06x^t + \frac{8.88}{(3.06x^t - \omega_{\text{in}}^t)\,(3.00y^t + 6.36)} \tag{9.5}$$

which has a very similar structure as the top EMM from run 1 ((9.2) and (9.3)) but with the positions of the x and y inputs swapped (there are differences in constants and signs as well). The evolutionary trajectory towards this agent is also opposite to the one leading up to the top agent of run 1, with communication about x positions emerging first (again, see Section 9.4).

The behaviour of the ω_{out} output of the top agent from run 1 is explored further in Figures 9.12 and 9.13. Figure 9.12(a) shows that when $\omega_{\text{in}} = 0$, ω_{out} outputs predominantly scaled y values, except when y is close to 0, in which case x values begin to influence ω_{out}. Notice that in this case, lower x values produce exponentially higher ω_{out} values. This is consistent with the behaviours seen above (see, e.g., Figure 9.9), namely that the agent with the lower x position outputs communication values on a higher order of magnitude than the other agent when in "lateral" mode. Figure 9.12(b) shows that when a top agents

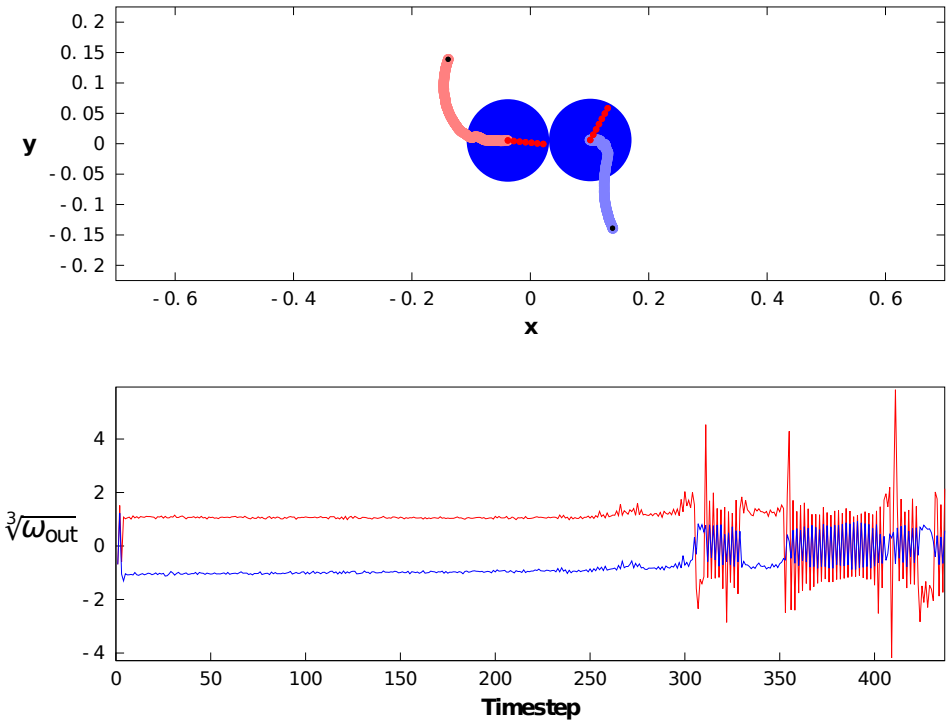


Figure 9.10: A test run using two clones of the top agent from run 1. The top figure shows the movements of the two agents over time, while the bottom figure shows $\sqrt[3]{\omega_{\text{out}}}$ for the two agents over time.

hears inputs at a higher order of magnitude, in this case $\omega_{\text{in}} = 100$, their ω_{out} values are scaled y values for all values of x. Why this occurs is obvious when examining (9.3), as the second term goes to zero as $\omega_{\text{in}} \to \infty$, leaving $\omega_{\text{out}}^{t+\Delta t} = 5.50y^t$. The figures in 9.13 show the behaviours of ω_{out} at four other ω_{in} values: -3, -1, 1, and 3. Notice that the y value at which ω_{out} begins to contain x information moves in the positive direction as the values of ω_{in} are increased. This is consistent with the behaviours described above, namely that agents operate in "longitudinal" mode until they converge on a common y position (as determined by comparing their y position to their ω_{in} value), at which point they switch to "lateral" mode and begin communicating x information.

Simplified top agent EMMs from three other *NoiseWorld* runs are:

$$\theta^{t+\Delta t} = \omega_{\text{out}}^t + 0.91y^t - \frac{v_3^t}{\omega_{\text{out}}^t - \omega_{\text{in}}^t - 0.03} \tag{9.6}$$

$$\omega_{\text{out}}^{t+\Delta t} = -0.06 - y^t - \frac{v_3^t}{-0.06 - y^t - (2.29/v_4^t v_5^t) - \omega_{\text{in}}^t} \tag{9.7}$$

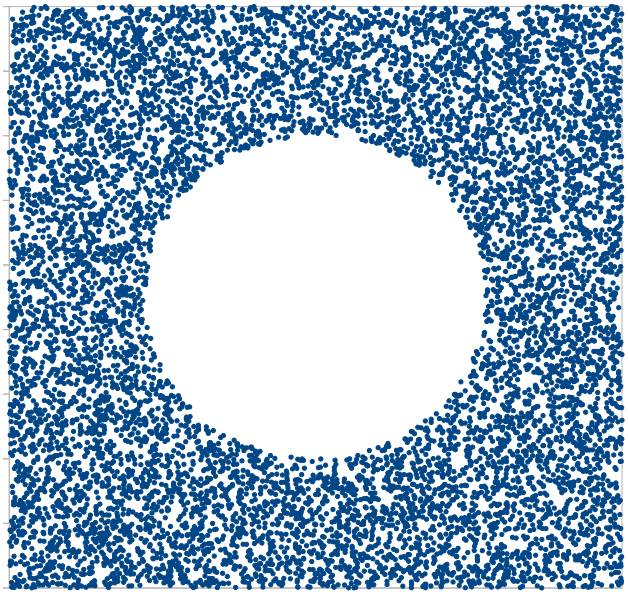
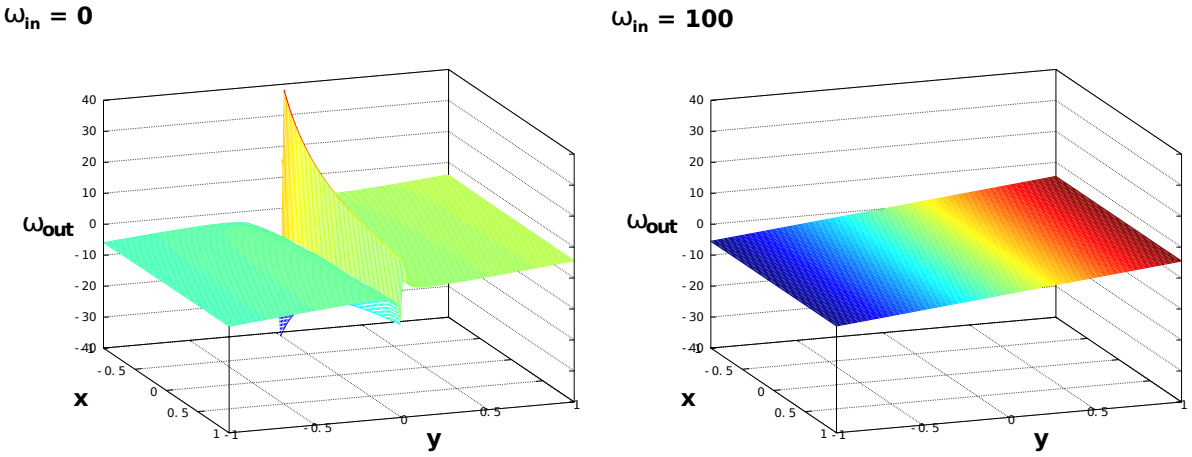


Figure 9.11: A visualization of 10,000 tests with one agent always starting in the middle and the second agent starting from randomly generated positions. If the two agents manage to reproduce within a 3,000 timestep test run, a blue dot is placed on the second agent's starting position, otherwise a red dot is placed. This figure shows such a test for two clones of the top agent from run 1. All 10,000 test runs were successful.



(a) (b)

Figure 9.12: ω_{out} values for the top agent from run 1 at various x and y input values and with either (a) $\omega_{\text{in}} = 0$ or (b) $\omega_{\text{in}} = 100$.

$$\theta^{t+\Delta t} = -1.59 + \frac{-0.76 + 1.58\left(y^t - 1.40\right) - v_3^t\left(\left(1.71 - x^t + v_4^t\right)/4.49 v_5^t v_6^t\right)}{8.57\left(1.15 + x^t\right) + \omega_{\text{in}}^t} \tag{9.8}$$

$$\omega_{\text{out}}^{t+\Delta t} = -8.42\left(1.15 + x^t\right) + \frac{2.51\left(y^t - 1.22\right)}{8.42\left(1.15 + x^t\right) + \omega_{\text{in}}^t} \tag{9.9}$$

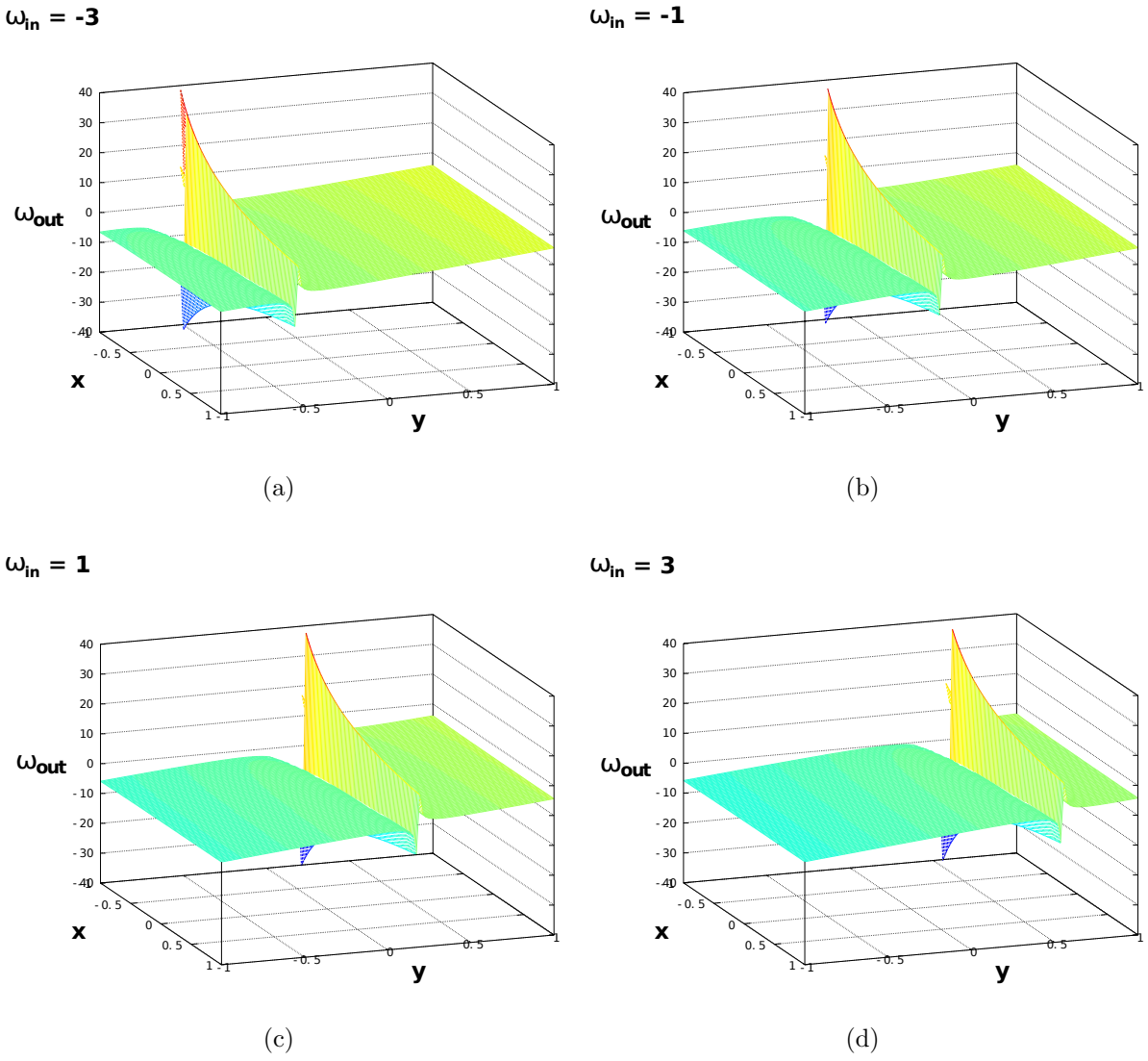


Figure 9.13: ω_{out} values for the top agent from run 1 for various x and y input values and with (a) $\omega_{\text{in}} = -3$, (b) $\omega_{\text{in}} = -1$, (c) $\omega_{\text{in}} = 1$, or (d) $\omega_{\text{in}} = 3$,.

$$\theta^{t+\Delta t} = -\frac{0.80 + 0.50x^t}{\omega_{\text{in}}^t + y^t} \tag{9.10}$$

$$\omega_{\text{out}}^{t+\Delta t} = -y^t + \frac{2.14\,(0.80 + 0.50x^t)}{(\omega_{\text{in}}^t + y^t)\,(-7.09 + 5.33x^t - x^t x^t)} \tag{9.11}$$

The equations for the extra variables v_i, $i = 3...N+N'$ are omitted. The reproduction rate on the island from which the top agent from run 1 was extracted was 23,394 reproduction events per era, while for run 2 it was 22,994. For the three EMMs shown above, the reproduction rates on their islands when they were extracted were 23,072, 25,281 and 25,295 reproduction events per era, respectively. These evolved EMMs illustrate the variability of solutions

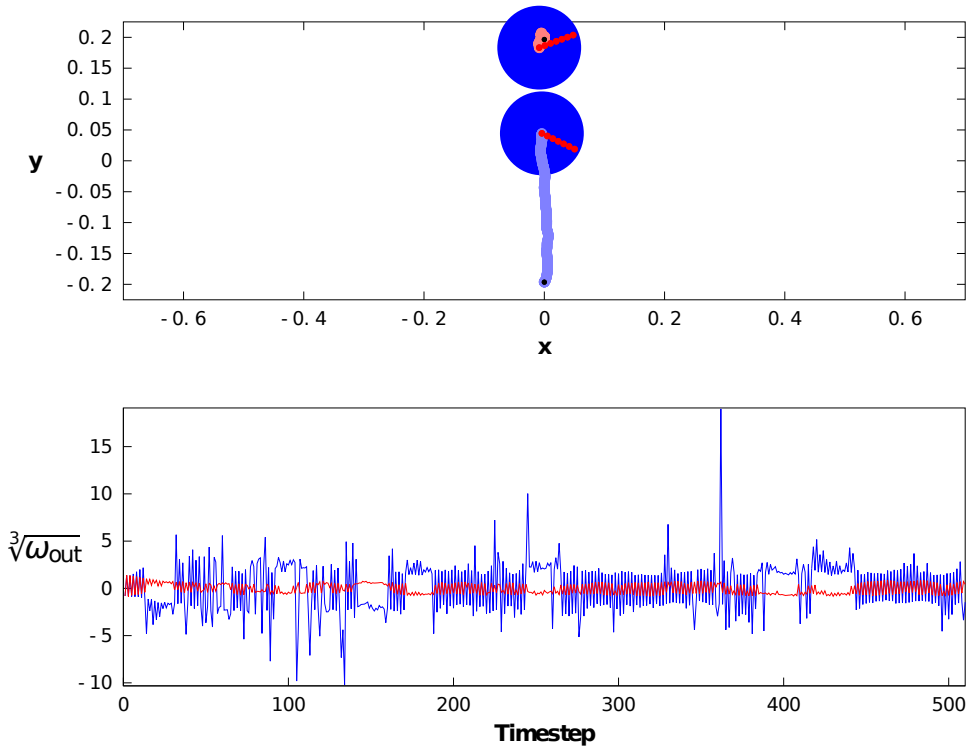


Figure 9.14: A test run using two clones of the top agent from run 2. The top figure shows the movements of the two agents over time, while the bottom figure shows $\sqrt[3]{\omega_{\text{out}}}$ for the two agents over time.

produced from different *NoiseWorld* runs. Furthermore, all five of these top solutions contain the communication input variable in their communication output equations, suggesting that the dialogue property of these communication schemes may be a crucial component in communicating 2D information over a 1D channel.

Figures 9.18–9.21 show four test runs using the EMM in (9.10) and (9.11), which is the top agent from the island with the highest reproduction rate across all simulation runs reported here. While the "lateral" and "longitudinal" modes are still obvious in Figures 9.18 and 9.19, there is no sign of the "lateral" mode when agents start in diagonal configurations (Figures 9.20 and 9.21). Even though the two agents are clearly moving diagonally to reach each other, the communication outputs are qualitatively very similar to those seen in typical "longitudinal" mode behaviours where agents movements are predominantly vertical. It would appear that the "longitudinal" mode has evolved to communicate more x position information than in other cases shown here. Furthermore, the information entropy for this top agent's communication scheme is *lower* than the values for the top agents from run 1 and run 2 (6.04 bits versus 7.54 bits and 7.14 bits, respectively). This indicates that this agent is using the information transmission capacity of its communication scheme more efficiently than those seen in run 1 and run 2.

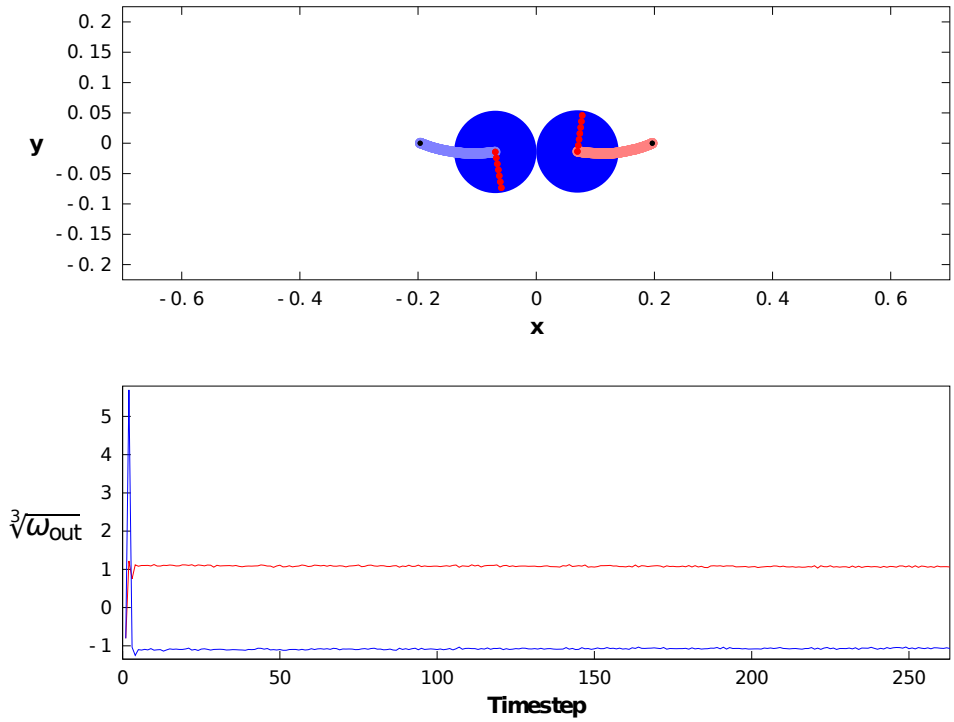


Figure 9.15: A test run using two clones of the top agent from run 2. The top figure shows the movements of the two agents over time, while the bottom figure shows $\sqrt[3]{\omega_{\text{out}}}$ for the two agents over time.

9.4 Analysis of Evolutionary Trajectories

These *NoiseWorld* simulations provide a unique opportunity by generating digital archaeological records of the emergence and evolution of the complex communication schemes described above. Figures 9.22–9.24 show three major milestones in the evolutionary trajectory of run 1. Early in the simulation at snapshot I (Figure 9.22), the top agent's EMM equations are

$$\theta^{t+\Delta t} = \theta^t + v_3^t \tag{9.12}$$

$$\omega_{\text{out}}^{t+\Delta t} = 3.74 y^t \tag{9.13}$$

$$v_3^{t+\Delta t} = \frac{-3.82}{(\theta^t - 2.46)(-0.67 - ((\theta^t + v_3^t)/(y^t - 4.81)))} \tag{9.14}$$

Agents have evolved to drive in circular patterns based on their y positions, yielding some reproductive benefits without the need for cooperative communication. Furthermore, while the agents are not "listening," their communication outputs contain information about their y positions. These are neutral communication outputs (or "purposeless sounds" as Darwin called them [34]) because they neither help nor hinder reproductive viability. Their complexity coupled with their similarity to later, reproductively beneficial communication schemes

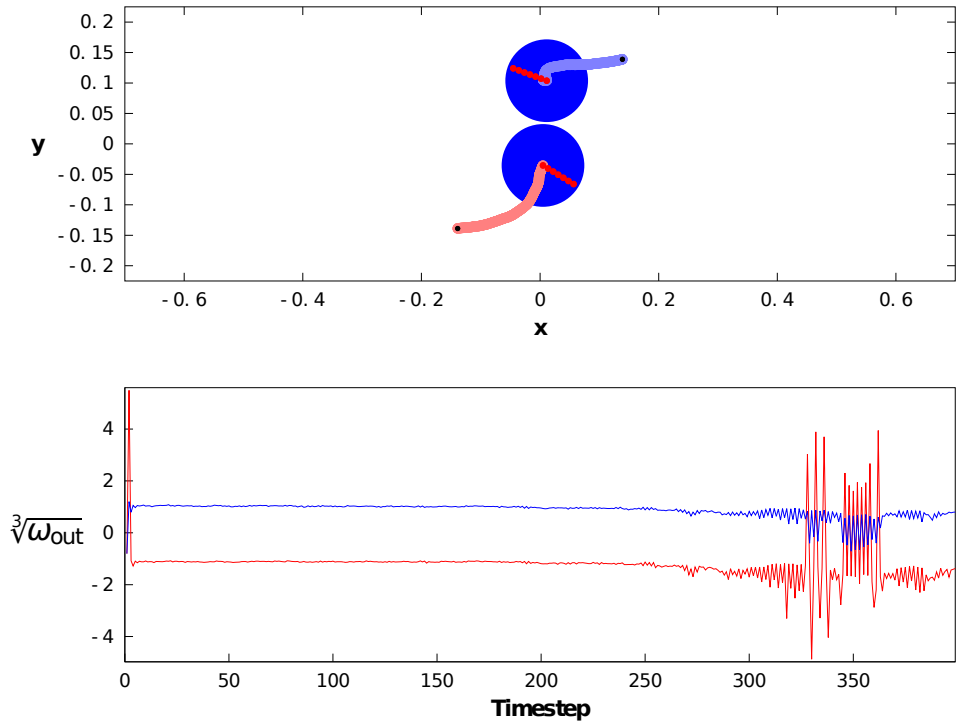


Figure 9.16: A test run using two clones of the top agent from run 2. The top figure shows the movements of the two agents over time, while the bottom figure shows $\sqrt[3]{\omega_{\text{out}}}$ for the two agents over time.

suggests shaping by selection, perhaps through one or more communication "false starts."

Figure 9.25 shows further evidence of "false starts," as before communication becomes fixed in the population, a significant percentage of the population seems to be modifying their behaviour based on what they are hearing (i.e., their $\theta^{t+\Delta t}$ equation contains ω_{in}^{t}) at several eras that are separated by periods where this "listening" behaviour is completely absent from the population.

This "false starts" hypothesis of how adaptive communication might emerge from initially noncommunicating agents differs from both the "receiver bias" hypothesis first put forth by Dawkins and Krebs [35] and the "producer bias" hypothesis put forth by Mirolli and Parisi [104] (see also [4]). The receiver bias hypothesis postulates that communication emerges to allow senders to exploit preexisting listening behaviours that evolved for some other adaptive purpose in the receivers. This is obviously not occurring here, as the EMM of the top agent at snapshot I does not contain the communication input variable. The producer bias hypothesis postulates that senders may first evolve to produce signals that are correlated with some environmental features and that provide some adaptive benefit to the sender, such as helping it to internally categorize experiences. Receivers can then evolve to exploit the information in these signals for their own benefit. Again, the EMM of the top agent at

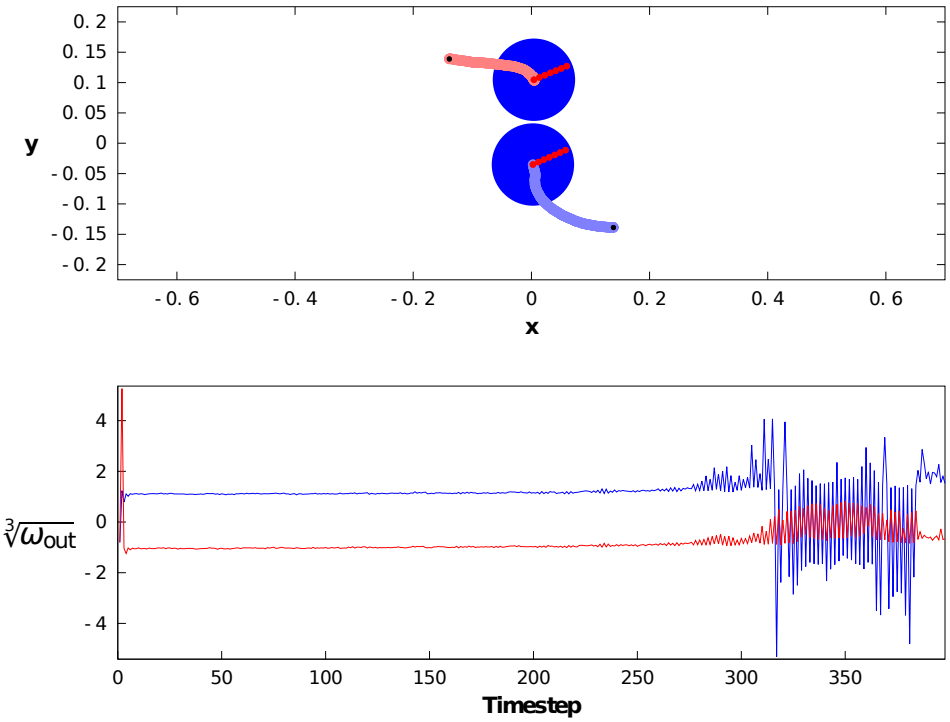


Figure 9.17: A test run using two clones of the top agent from run 2. The top figure shows the movements of the two agents over time, while the bottom figure shows $\sqrt[3]{\omega_{\text{out}}}$ for the two agents over time.

snapshot I allows us to rule this theory out, as its own communication output does not appear on the right hand side of any of its equations. The receiver bias hypothesis is used to explain how a manipulative communication scheme might emerge, while the producer bias hypothesis is used to explain how an altruistic communication scheme might emerge. In *NoiseWorld*, communication is neither manipulative nor altruistic; it is cooperative. Thus, perhaps a third theory is required, namely our "false start" theory, for which further evidence is uncovered in the analysis of the evolutionary trajectory of run 2, below. The idea of "false starts" can also be thought of as multiple iterations of evolution exploiting a receiver bias, which might then create a producer bias (or vice versa), which might then be exploited to further develop the receiver bias, and so on until communication becomes fixed in the population.

At snapshot II (Figure 9.23), the top agent's EMM equations are

$$\theta^{t+\Delta t} = 2.45(\omega_{\text{in}}^{t} - 4.36y^{t}) \tag{9.15}$$

$$\omega_{\text{out}}^{t+\Delta t} = 4.36y^{t} \tag{9.16}$$

Agents have evolved to utilize a communication signal similar to the previously neutral one to obtain significant reproductive benefits. Evidence of neutral communications being co-

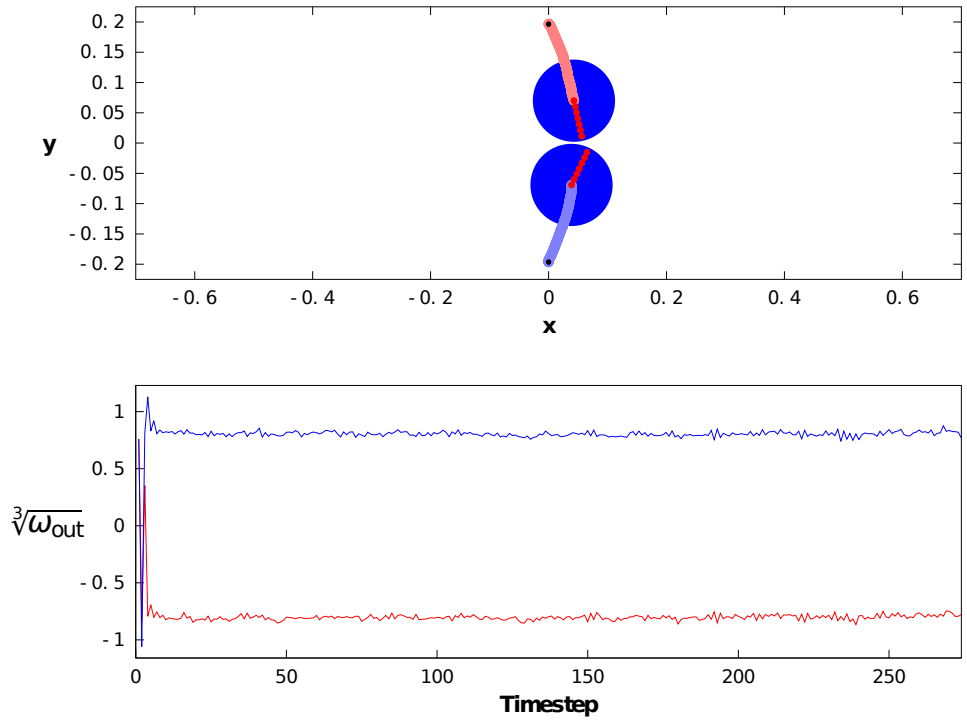


Figure 9.18: A test run using two clones of the EMM from (9.10) and (9.11). The top figure shows the movements of the two agents over time, while the bottom figure shows $\sqrt[3]{\omega_{\text{out}}}$ for the two agents over time.

opted for adaptive communication was also uncovered by de Greeff and Nolfi in [36]. Since the information being communicated is about an agent's y position, it is no surprise that the agents are reproductively successful from vertically aligned starting configurations (see Figure 9.28). At this point, no "plan B" has evolved for when agents are not vertically aligned. It is clear from the agent output equations that once agents converge to a common y position, they will move in a fixed direction indefinitely.

At snapshot III (Figure 9.24), the top agent's EMM equations are

$$\theta^{t+\Delta t} = \omega_{\text{in}}^{t} - 5.50y^{t} \tag{9.17}$$

$$\omega_{\text{out}}^{t+\Delta t} = 5.50y^{t} - \frac{1.39}{(1.53 + x^{t})(\omega_{\text{in}}^{t} - 5.50y^{t})} \tag{9.18}$$

We see that the communication output has undergone complexification. While the y position variable remains relatively unchanged from the previous milestone, a new term containing both x and y position inputs as well as ω_{in} has evolved. Thus, neighbouring agents are now explicitly in dialogue and are transmitting encoded x and y position information over one-dimensional channels. Agent orientation output equations have remained relatively unchanged, suggesting that the communication scheme evolved to further exploit preexisting

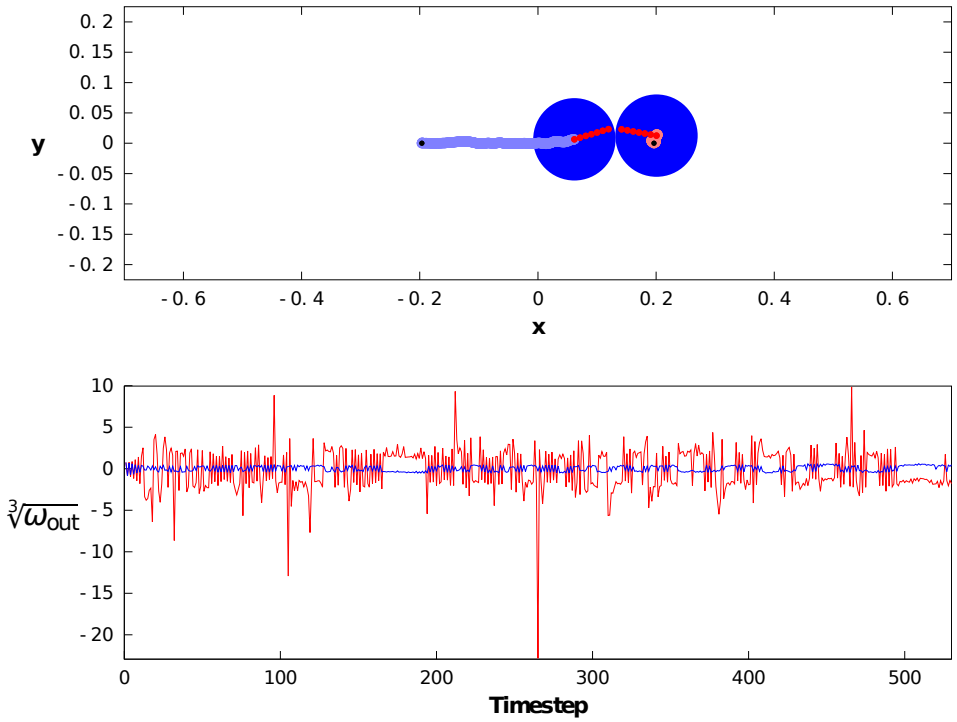


Figure 9.19: A test run using two clones of the EMM from (9.10) and (9.11). The top figure shows the movements of the two agents over time, while the bottom figure shows $\sqrt[3]{\omega_{\text{out}}}$ for the two agents over time.

listening behaviours (for other examples of evolution exploiting preexisting behaviours, see [98, 36]).

Figure 9.26 shows the distribution of various variables across agent equations over the eras where the complexification between snapshots II and III occurs. It is interesting to note that there are "false starts" here as well, as complexification seems to occur twice, separated by the population falling back to its previous communication scheme, before a new scheme becomes fixed in the population. Note that when the new scheme becomes fixed, both the x input variable and the ω_{in} variable seem to enter the population's communications simultaneously. This could be explained by a migration event introducing a new, highly beneficial subfunction containing both variables, or it could just be that the resolution of the snapshots is too coarse. Figure 9.27 shows the distribution of various variables across agent equations for the entire length of run 1. This demonstrates the stability of communication in *NoiseWorld*, as once it becomes fixed in the population, it does not disappear. However, the more complex communication scheme is not as stable, as late in the simulation the population returns back to the simple signalling strategy.

A second evolutionary trajectory was examined, this time from run 2, where $\theta = 0$ was aligned with the positive y direction. More emphasis was placed on the emergence of commu-

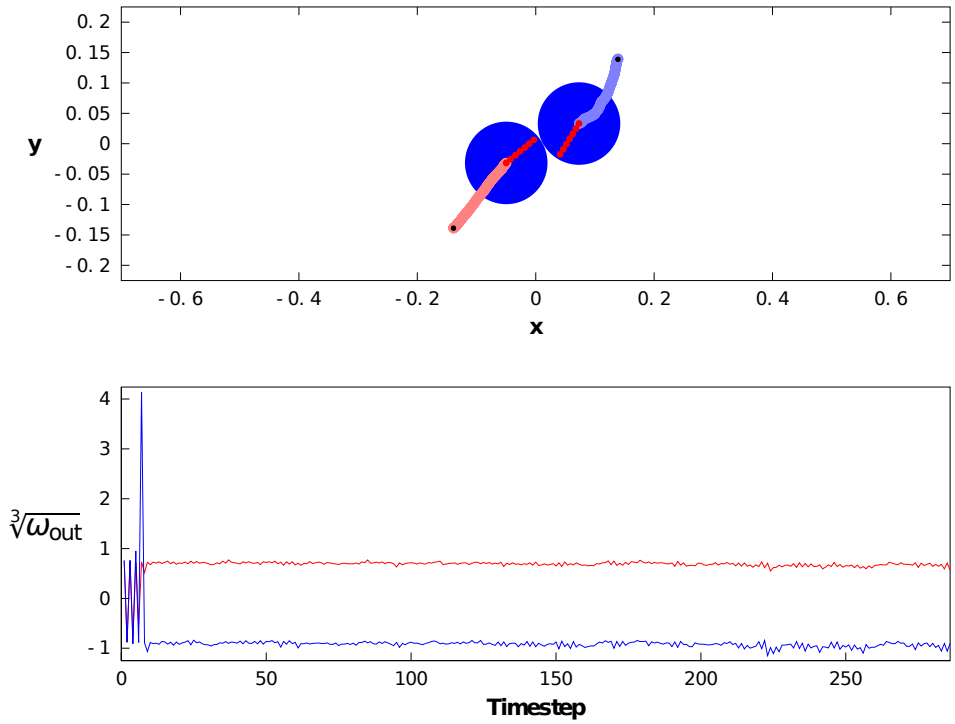


Figure 9.20: A test run using two clones of the EMM from (9.10) and (9.11). The top figure shows the movements of the two agents over time, while the bottom figure shows $\sqrt[3]{\omega_{\text{out}}}$ for the two agents over time.

nication during this analysis, therefore there are six snapshots instead of the previous three, with the first three of these coming before communication has become fixed in the population (see Figures 9.29–9.34). Figure 9.4 shows the reproductive success of these six snapshots. Snapshot II seems to have some communication occurring, although it has disappeared by snapshot III. This provides further evidence for the "false starts" theorized above. Figure 9.6 shows Shannon's information entropy of the communication output of the top agent from each of these snapshots. Again we see evidence of "false starts" shaping initial communication outputs, as the maximum potential information content seems to be rising steadily, even before communication has become fixed in the population.

The simplified genomes of the top agent from each snapshot are as follows. The EMM for snapshot I is

$$\theta^{t+\Delta t} = 1.57v_3^t - x^t - 2.97 \tag{9.19}$$

$$\omega_{\text{out}}^{t+\Delta t} = -1.42 \tag{9.20}$$

$$v_3^{t+\Delta t} = v_4^t - 4.00 \tag{9.21}$$

$$v_4^{t+\Delta t} = v_4^t - 4.00 \tag{9.22}$$

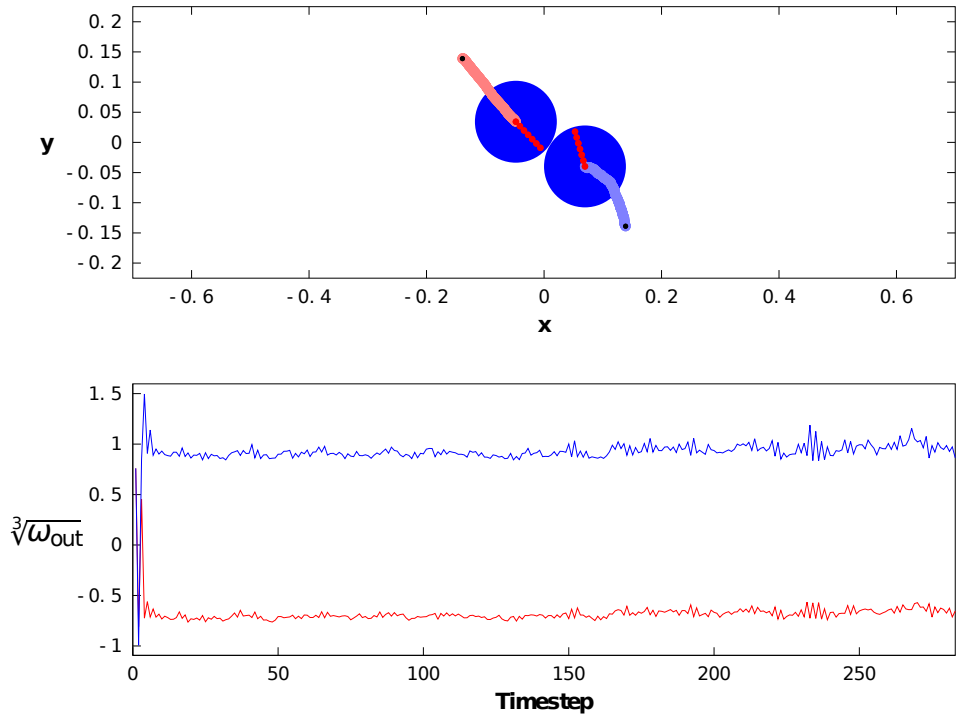


Figure 9.21: A test run using two clones of the EMM from (9.10) and (9.11). The top figure shows the movements of the two agents over time, while the bottom figure shows $\sqrt[3]{\omega_{\text{out}}}$ for the two agents over time.

The EMM for snapshot II is

$$\theta^{t+\Delta t} = 1.57v_3^t - x^t - 2.97 \tag{9.23}$$

$$\omega_{\text{out}}^{t+\Delta t} = -1.05 + \frac{x^t}{v_4^t} \tag{9.24}$$

$$v_3^{t+\Delta t} = v_4^t + v_5^t \tag{9.25}$$

$$v_4^{t+\Delta t} = -5.05 + \frac{x^t}{v_4^t} \tag{9.26}$$

$$v_5^{t+\Delta t} = \frac{\omega_{\text{in}}^t}{\omega_{\text{out}}^t} \tag{9.27}$$

The EMM for snapshot III is

$$\theta^{t+\Delta t} = \theta^t + \frac{9.20}{3.43 + 268.60\omega_{\text{out}}^t} \tag{9.28}$$

$$\omega_{\text{out}}^{t+\Delta t} = 1.43x^t - 6.39 \tag{9.29}$$

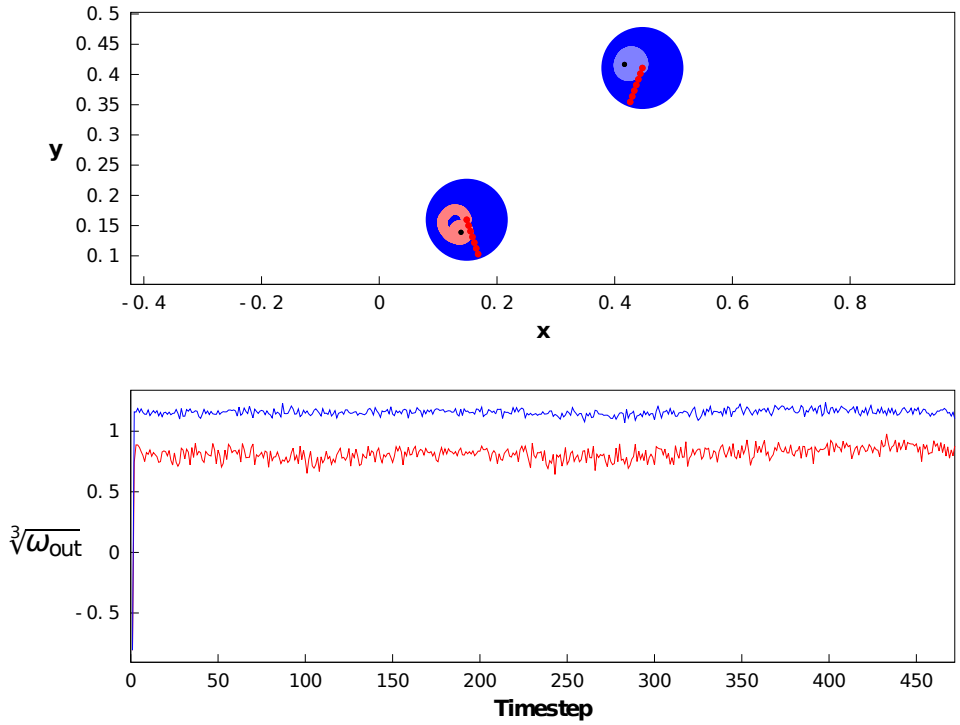


Figure 9.22: Test run for snapshot I taken during simulation run 1. The EMM used is shown in (9.12), (9.13) and (9.14).

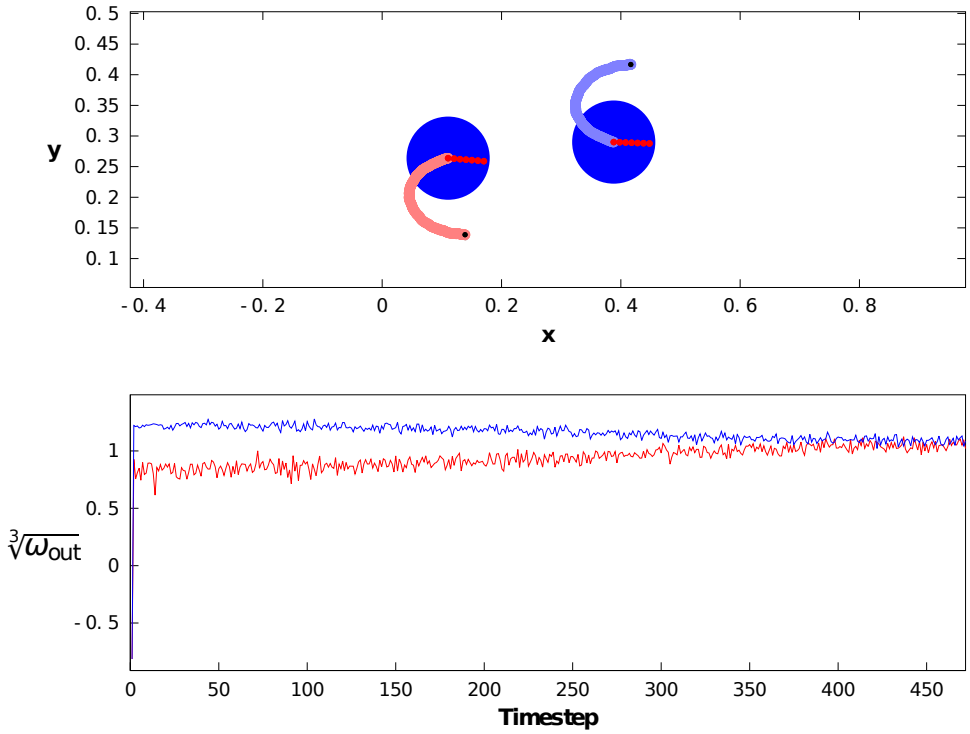


Figure 9.23: Test run for snapshot II taken during simulation run 1. The EMM used is shown in (9.15) and (9.16).

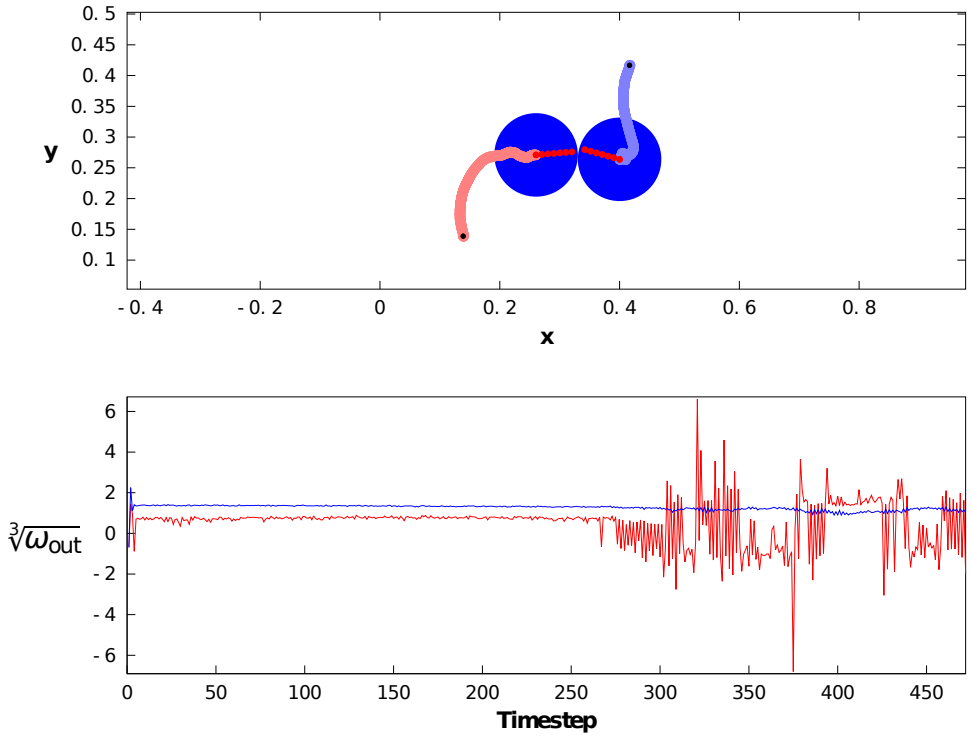


Figure 9.24: Test run for snapshot III taken during simulation run 1. The EMM used is shown in (9.17) and (9.18).

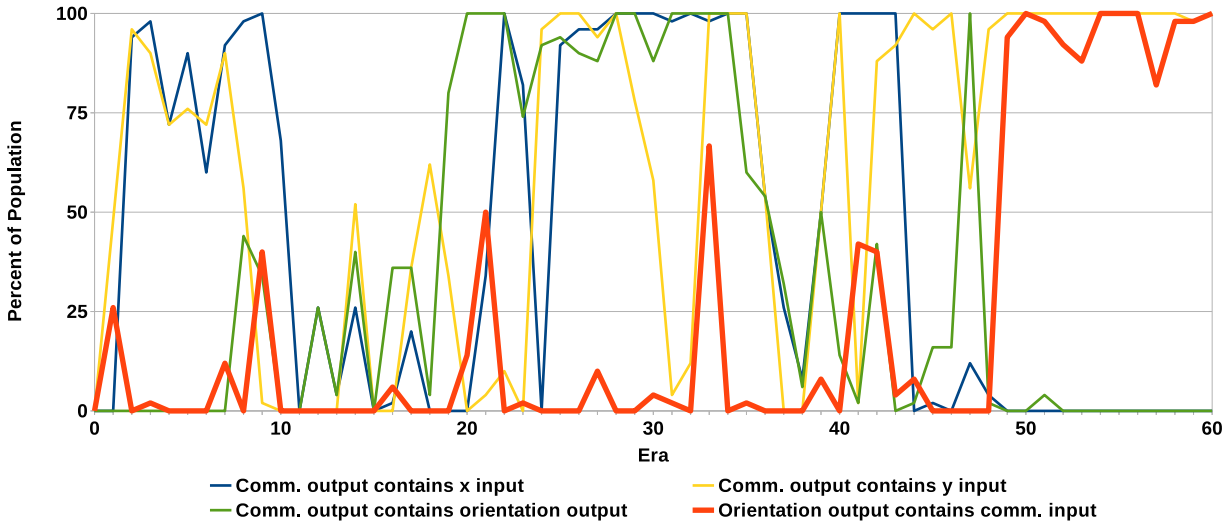


Figure 9.25: Percentage of the population of island 64 in run 1 containing various variables in their output equations, over the first 60 eras. If an output equation is modified by an extra state variable that is in turn modified by the variable in question, this variable is counted as modifying the output in question. Note that neutral variables (e.g., if the variable is divided by ∞) are still counted.

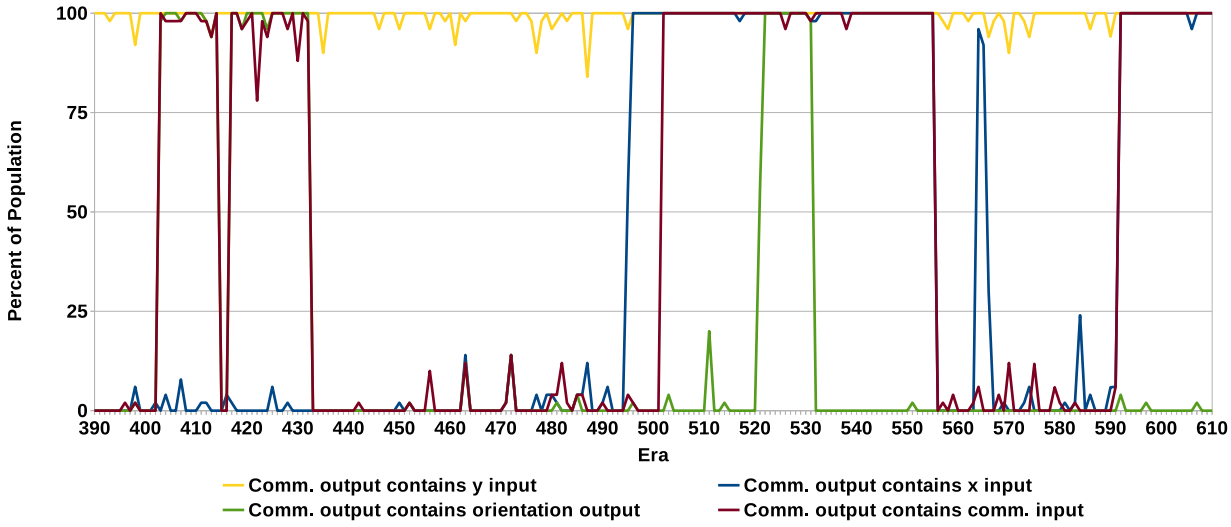


Figure 9.26: Percentage of the population of island 64 in run 1 containing various variables in their output equations, over eras 450 to 650. If an output equation is modified by an extra state variable that is in turn modified by the variable in question, this variable is counted as modifying the output in question. Note that neutral variables (e.g., if the variable is divided by ∞) are still counted.

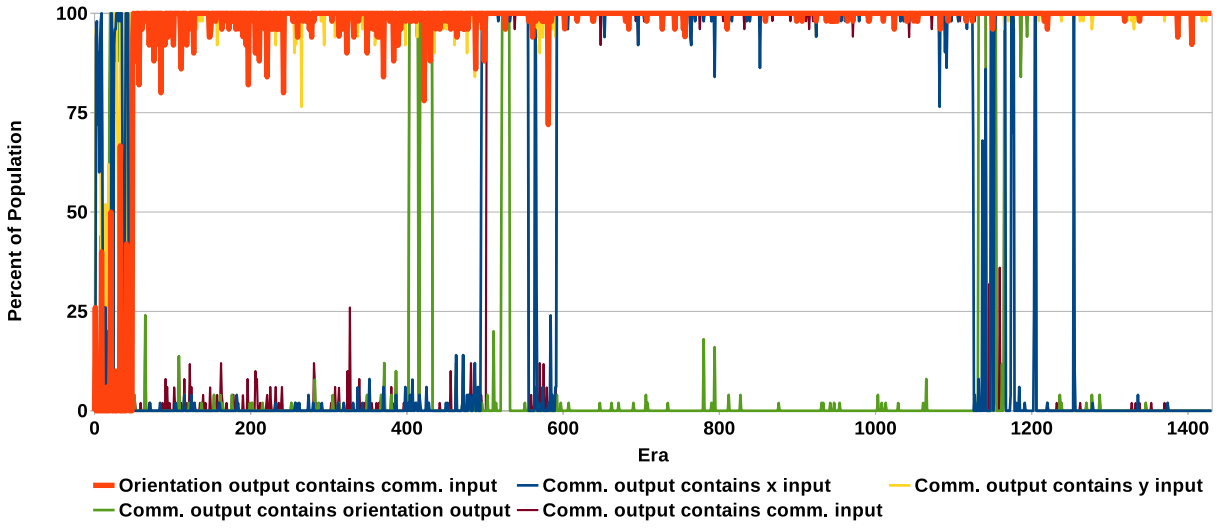


Figure 9.27: Percentage of the population of island 64 in run 1 containing various variables in their output equations, over the entire run. If an output equation is modified by an extra state variable that is in turn modified by the variable in question, this variable is counted as modifying the output in question. Note that neutral variables (e.g., if the variable is divided by ∞) are still counted.

The EMM for snapshot IV is

$$\theta^{t+\Delta t} = \frac{2.34}{1.31x^t - \omega_{\text{in}}^t + (2.34/\left(1.44x^t - \omega_{\text{in}}^t + \theta^t\right))} \tag{9.30}$$

$$\omega_{\text{out}}^{t+\Delta t} = 1.44x^t \tag{9.31}$$

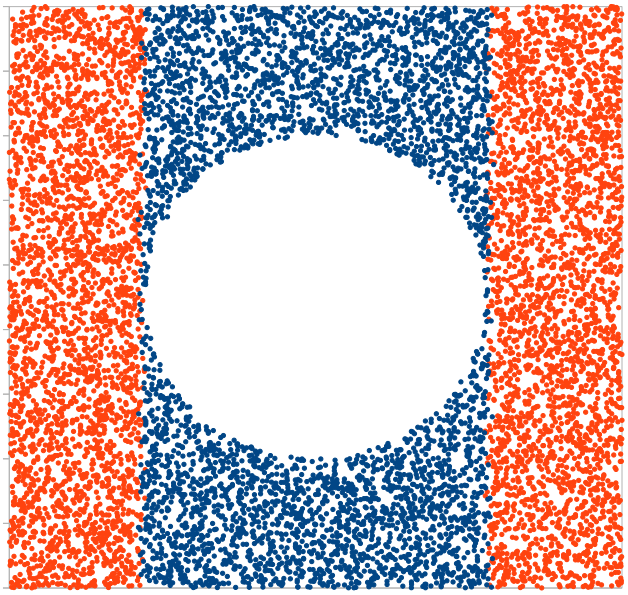


Figure 9.28: A visualization of 10,000 tests with one agent always starting in the middle and the second agent starting from randomly generated positions. If the two agents manage to reproduce within a 3,000 timestep test run, a blue dot is placed on the second agent's starting position, otherwise a red dot is placed. This figure shows such a test for two clones of the top agent from snapshot II of run 1. Of the 10,000 test runs, 4,253 were successful.

The EMM for snapshot V is

$$\theta^{t+\Delta t} = 1.93x^t - \omega_{\text{in}}^t + \frac{1.30}{\omega_{\text{out}}^t - \omega_{\text{in}}^t + (1.30/\left(1.77x^t - \omega_{\text{in}}^t + \theta^t\right))} \tag{9.32}$$

$$\omega_{\text{out}}^{t+\Delta t} = 1.77x^t \tag{9.33}$$

Finally, the EMM for snapshot VI is

$$\theta^{t+\Delta t} = 3.06x^t - \omega_{\text{in}}^t \tag{9.34}$$

$$\omega_{\text{out}}^{t+\Delta t} = 3.06x^t + \frac{8.88}{\left(3.06x^t - \omega_{\text{in}}^t\right)\left(3.00y^t + 6.36\right)} \tag{9.35}$$

Their corresponding behaviours are shown in Figures 9.29–9.34. Again, we can see that in snapshot II there is some communication occurring, which has evolved away by snapshot III. Furthermore, the communication output equations (ω_{out}) show a clear progression in structure as the population goes back and forth between communicating and noncommunicating over the first four snapshots.

Figure 9.35 again provides further evidence of "false starts," as again we see the repeated

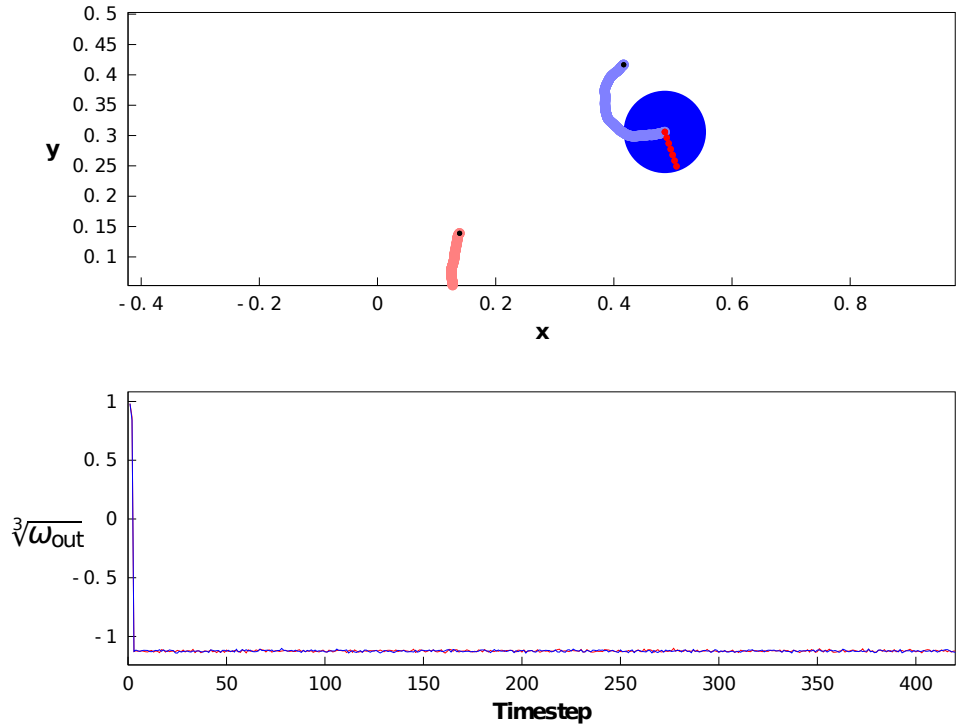


Figure 9.29: Test run for snapshot I taken during simulation run 2. The EMM used is shown in (9.19), (9.20), (9.21) and (9.22).

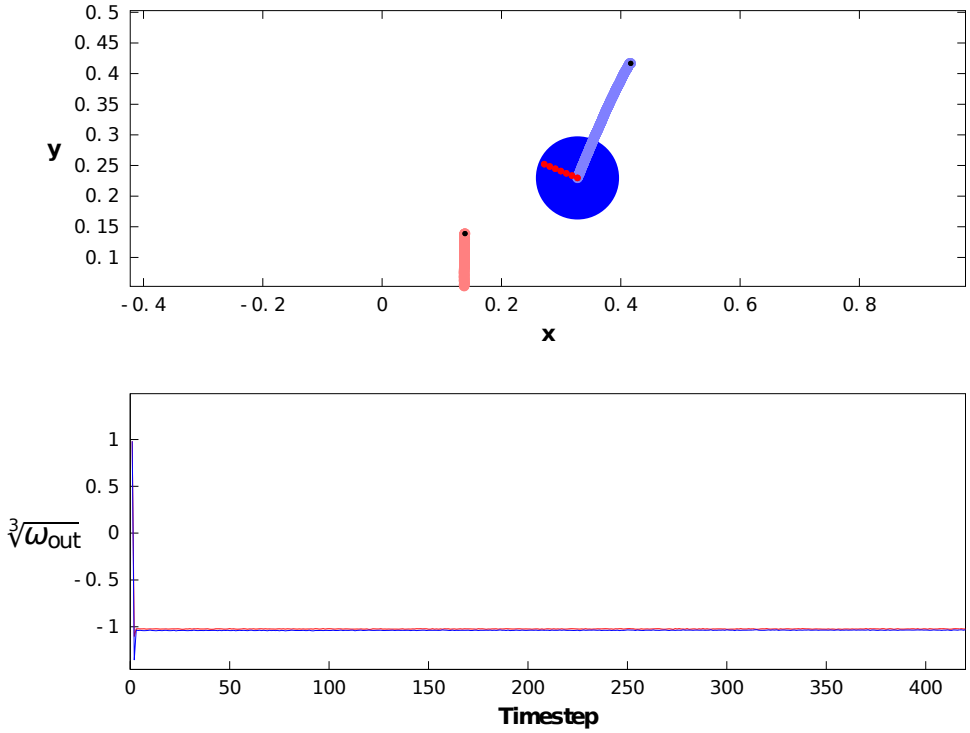


Figure 9.30: Test run for snapshot II taken during simulation run 2. The EMM used is shown in (9.23), (9.24), (9.25), (9.26) and (9.27).

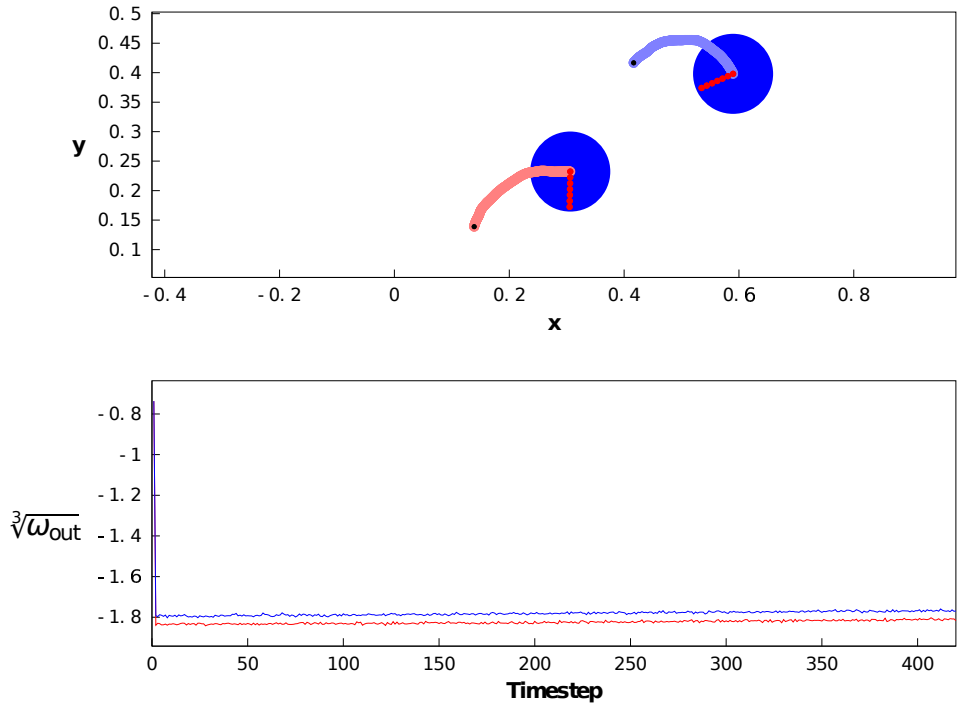


Figure 9.31: Test run for snapshot III taken during simulation run 2. The EMM used is shown in (9.28) and (9.29).

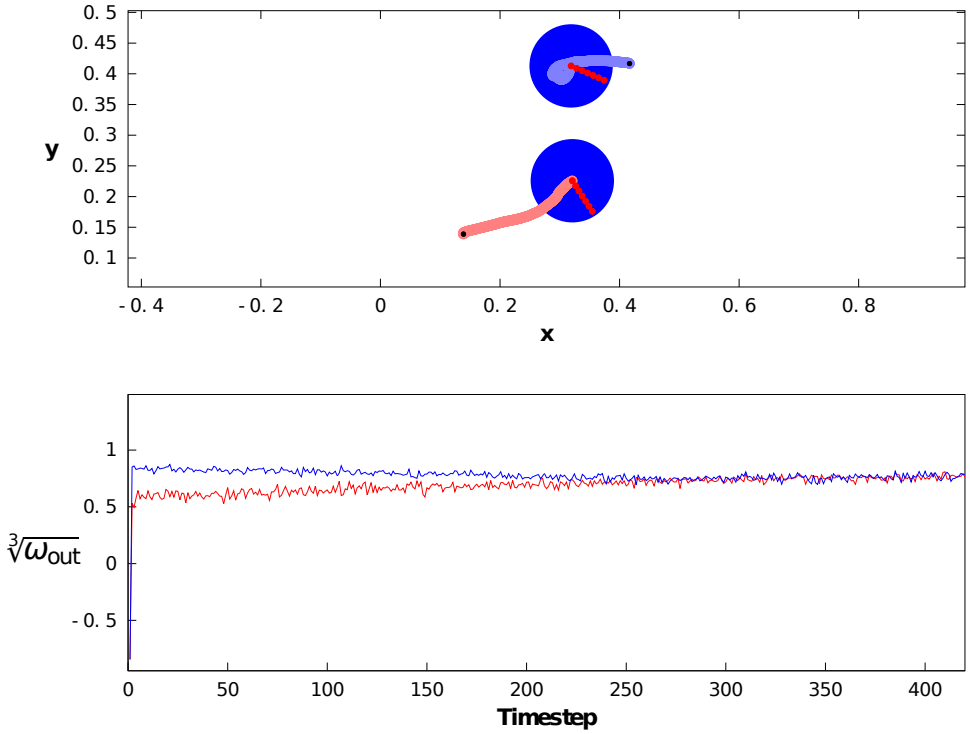


Figure 9.32: Test run for snapshot IV taken during simulation run 2. The EMM used is shown in (9.30) and (9.31).

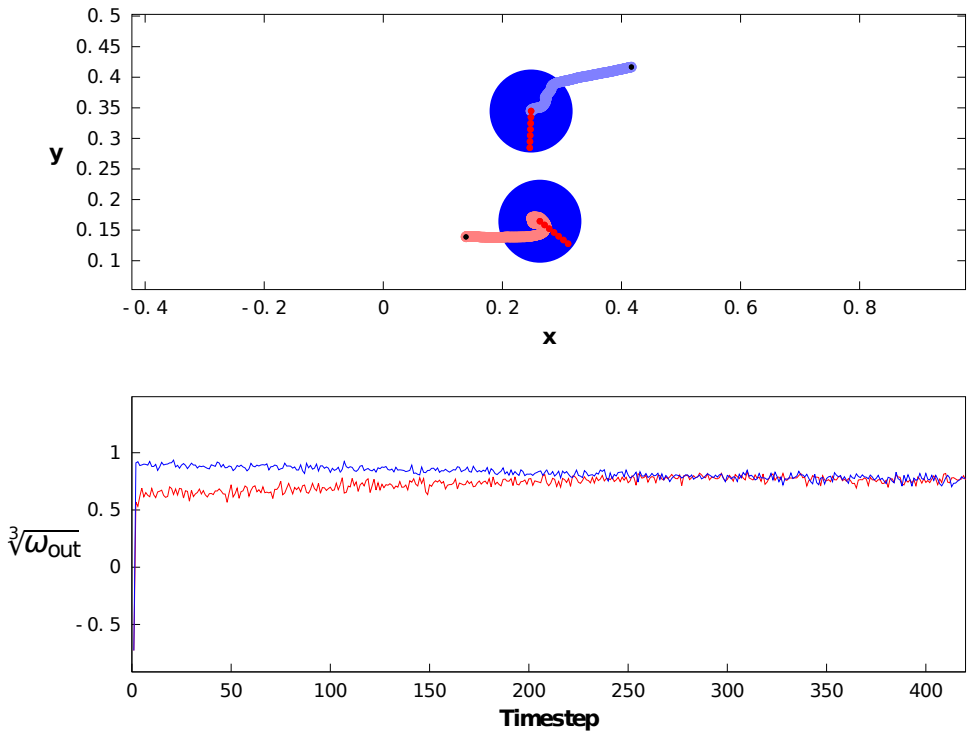


Figure 9.33: Test run for snapshot V taken during simulation run 2. The EMM used is shown in (9.32) and (9.33).

rise and fall of listening behaviour before communication becomes fixed in the population. Figure 9.36 shows the transition from simple signalling to encoded communication. Again, several different communication schemes seem to rise and fall before the new encoding scheme becomes fixed in the population. It is interesting to note that this figure differs significantly from Figure 9.26, as here the dialogue behaviour seems to fix in the population long before the second position variable (in this case, y) becomes fixed. In run 1, the dialogue behaviour and the second position variable become fixed in the population simultaneously. This indicates that there are at least two different evolutionary trajectories that lead to a similar encoded communication scheme (although we suspect that there are many more). Finally, Figure 9.37 shows the percentage of variables in various output equations across all eras of run 2, showing the relative stability of the communication schemes once they become fixed in the population.

9.5 Evaluation of Evolved Communication Schemes

We will now evaluate the evolved communication schemes of *NoiseWorld*, described above, using the evaluation criteria proposed by Mirolli and Nolfi in [102].

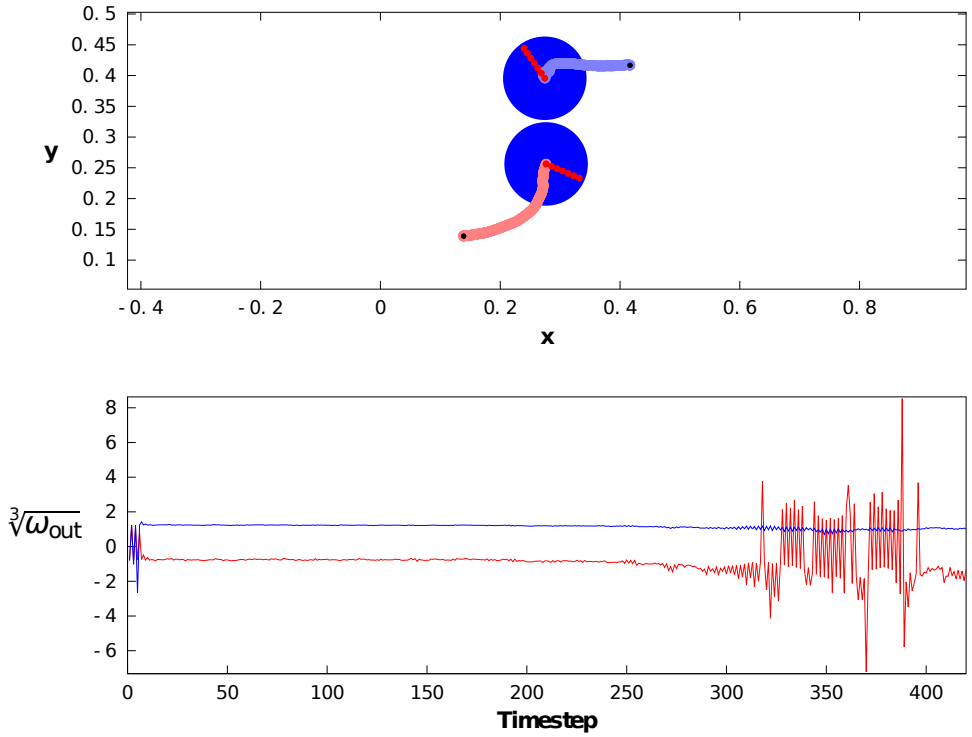


Figure 9.34: Test run for snapshot VI taken during simulation run 2. The EMM used is shown in (9.34) and (9.35).

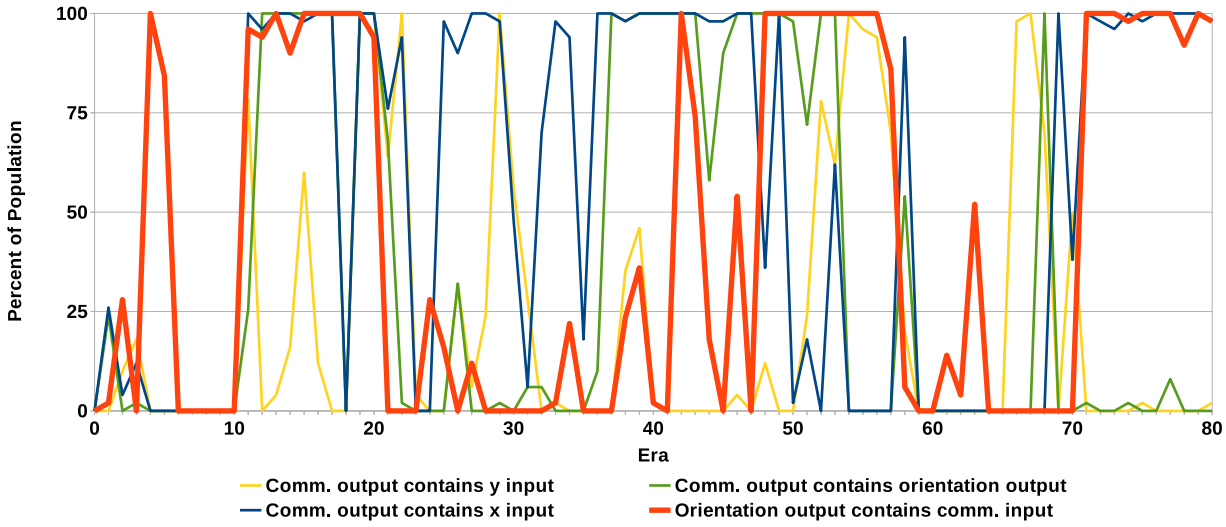


Figure 9.35: Percentage of the population of island 98 in run 2 containing various variables in their output equations, over the first 80 eras. If an output equation is modified by an extra state variable that is in turn modified by the variable in question, this variable is counted as modifying the output in question. Note that neutral variables (e.g., if the variable is divided by ∞) are still counted.

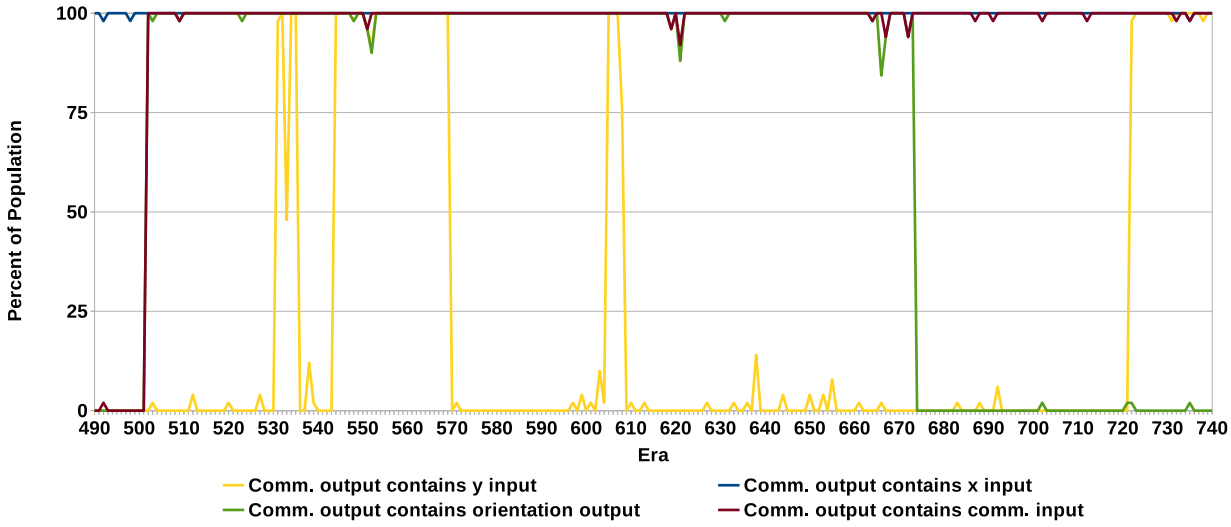


Figure 9.36: Percentage of the population of island 98 in run 2 containing various variables in their output equations, from eras 490 to 740. If an output equation is modified by an extra state variable that is in turn modified by the variable in question, this variable is counted as modifying the output in question. Note that neutral variables (e.g., if the variable is divided by ∞) are still counted.

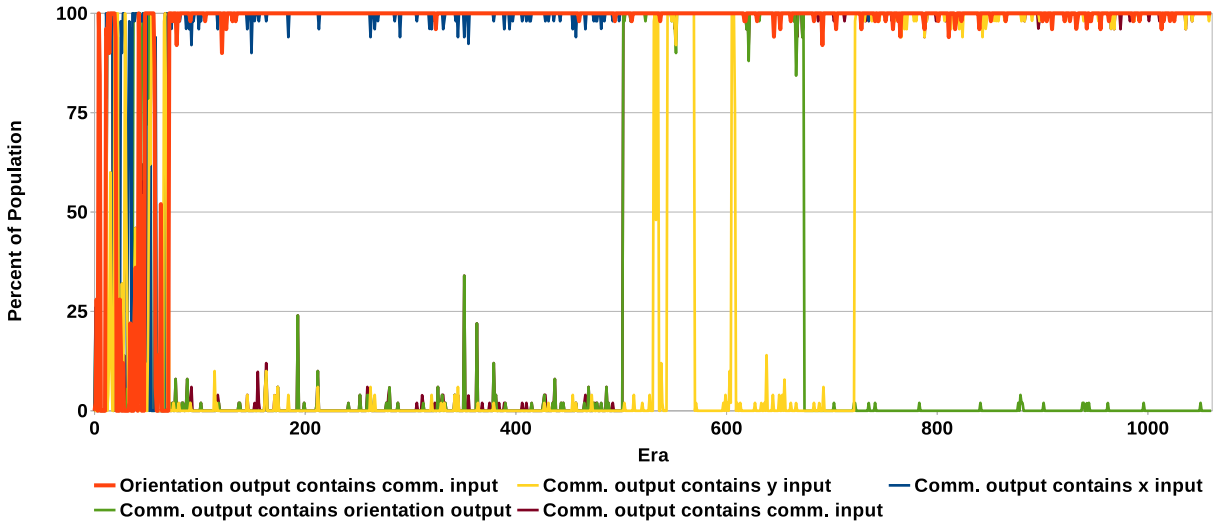


Figure 9.37: Percentage of the population of island 98 in run 2 containing various variables in their output equations, over the entire run. If an output equation is modified by an extra state variable that is in turn modified by the variable in question, this variable is counted as modifying the output in question. Note that neutral variables (e.g., if the variable is divided by ∞) are still counted.

9.5.1 Adaptive Role

From Figures 9.3 and 9.4 above, it is clear that the communication schemes in *NoiseWorld* are reproductively beneficial to the agents that employ them. Cooperation via the commu-

nication channel allows agents to find one another, and thus reproduce, more rapidly.

9.5.2 Expressive Power and Organizational Complexity

The type of signal is referential, i.e., it provides information about the external environment, namely the position of the agent. The signal is deictic, as it is based on the current context of the sender. Furthermore, the signals are relational, meaning that the roles of the individuals in communication cannot be distinguished. However, the evolved communication schemes have both symmetrical and asymmetrical modes. The communication scheme also possesses a rudimentary signal structure, as agents will first communicate about one dimension, and then once they reach the same position along that dimension, they will begin communicating about the second dimension. The most interesting trait of the evolved signalling strategy, however, is that it is not only abstract, i.e., it encodes information that has been generated by integrating sensory-motor information over time, it also evolved from a non-abstract communication scheme, where the signal provided information that was directly and currently available (see, e.g., Snapshot II, run 1, above). As far as the author knows, this is the first case where transitions from noncommunication to non-abstract signalling, and then from non-abstract signalling to abstract signalling, have been observed, either in nature or in simulation.

9.5.3 Stability, Robustness, and Evolvability

Figures 9.27 and 9.37 clearly indicate that once communication becomes fixed in a population, it remains indefinitely. Thus, *NoiseWorld* communication schemes are highly stable, even though selection is operating on the level of the individual. This is most likely owing to the fact that cooperation in *NoiseWorld* is mutually beneficial. It should be noted, however, that late in run 1 the population returned to the simple signalling scheme, indicating that perhaps the encoded communication scheme is less stable.

Evolved communication schemes are also highly robust, as they continue to provide reproductive benefits to agents even after transfer into physical hardware (see Chapter 10, below).

Finally, *NoiseWorld* has a high degree of evolvability, as cooperative communication consistently emerges from initially noncommunicating agents, potentially owing to the communication "false starts" hypothesized above. Furthermore, *NoiseWorld* allows for complexification beyond the initial signalling schemes that emerge, from 1D signalling to a communication scheme that encodes two dimensions of information. Future work will look to further increase the open-endedness of the simulation.

9.5.4 Knowledge Gain

NoiseWorld has provided some knowledge gain with the hypothesis of communication "false starts" shaping initial neutral signals. Furthermore, *NoiseWorld* provides a unique opportunity to observe the transition from simple signalling to complex encoded communication in a continuous simulation world with no discrete generations, no explicit group selection and no explicit fitness function.

Chapter 10

Hardware Experiments

10.1 Setup

The simulated agents (EMMs) are readily transferrable to e-puck robots. The e-puck robot (see [109]) has two wheels, each with their own motor (with a resolution of 1,000 steps per wheel revolution), and a large collection of sensors and actuators, including eight infrared proximity sensors, a 3D accelerometer, three microphones, a colour camera and a speaker. An example e-puck robot is shown in Figure 10.1. Only the motors are employed for the following experiments.

The e-puck robots have a diameter of 75 mm. This is the real-world "reproduction distance," which was 0.139 units in the simulations described above. Therefore, when converting simulation units into real-world values, we use the conversion

$$1 \text{ simulation unit} \quad \approx \quad 0.54 \text{ m} \tag{10.1}$$

For all hardware experiments described here, two agents were run in a synchronized fashion on a laptop, with motor speed adjustments being sent to the robots via Bluetooth. This was implemented using MATLAB, therefore the EMMs and the methods to evaluate them had to be ported from C++. This resulted in a loss of precision for the constants used in the EMMs, as well as several other simplifications. To control the e-pucks via Bluetooth using MATLAB, the free ePic2 framework was used. Robot orientation and x and y positions were determined using an overhead webcam, custom colour detection software and coloured markers on top of the robots themselves (see Figure 10.2). The colour detection software was written in C# and connected to MATLAB via TCP/IP socket (our MATLAB code would run a Java-based TCP/IP server and the C# colour detection program would connect as a client).

Figure 10.1: An e-puck robot; from [109].

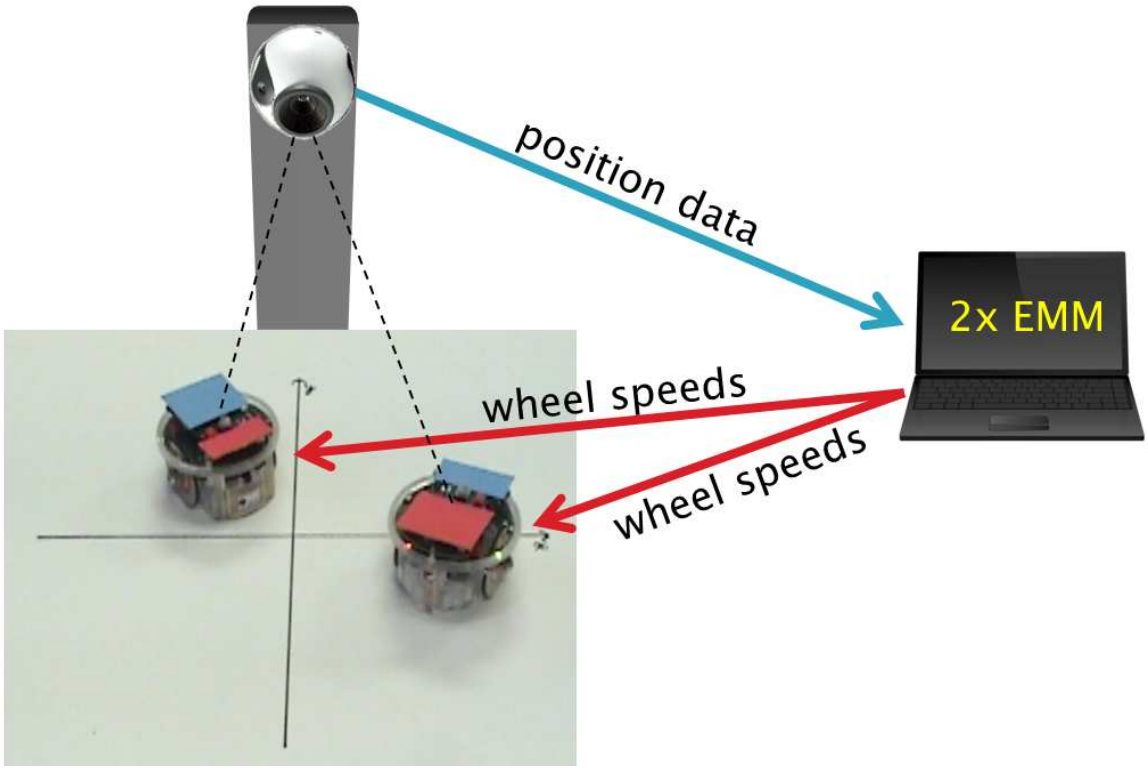


Figure 10.2: Setup for hardware experiments.

Robotic experiments proceed as follows:

1. Set all agent variables to their initial values $\mathbf{v}^{t=0}$.

2. Get agent x and y positions from overhead tracking system.

3. Evaluate both sets of agent equations for 10 timesteps, with an agent's ω_{in} being set to the other agent's ω_{out} from the previous timestep. The x and y input values are not

changed during these 10 timesteps, although new noise values are used at each step. θ is treated as an internal variable and is thus updated at each step.

4. Calculate the cumulative (i.e., over the past 10 timesteps) expected motion of each agent. This yields a new expected position. Each robot is turned to face its expected position ($\pm\pi/16$) and then set to drive forward. If a robot is already within $\pm\pi/16$ of this expected orientation, it is not turned. If the robots are in motion and at least one agent needs to turn, both robots are stopped. Otherwise they are left to continue forward in their current direction.

5. Go to step 2.

The EMM equations used for the hardware experiments are

$$\theta^{t+\Delta t} = \frac{3.89 + x^t}{(-8.80y^t + \omega_{\text{in}}^t)} \tag{10.2}$$

$$\omega_{\text{out}}^{t+\Delta t} = 8.81y^t - 0.76\theta^t(1.80 + x^t) \tag{10.3}$$

This is a simplified top agent from an unreported canonical *NoiseWorld* simulation run. This EMM originates from a slightly older version of *NoiseWorld* than the other EMMs reported in this document. The only difference between the older version and the current implementation is that the latest version of *NoiseWorld* contains a slightly improved method for introducing incoming migrants into an island's population. All other implementation details are identical.

10.2 Experiments and Results

Video snapshots and communication data from four hardware experiments with different initial robot positions are shown in Figures 10.3, 10.6, 10.9 and 10.12. Note that this hardware experiment setup does not seem to allow for agents to remain sufficiently close to each other's y position to remain in "lateral" mode continuously. This is especially obvious in the run shown in Figure 10.12, where the agents were initialize with similar y positions. The agents instead repeatedly switch between their "lateral" and "longitudinal" modes, which is still a successful behaviour. These EMM agents are demonstrating a remarkable level of adaptability and hardware transferability, as they readily adapt to the lack of position and movement precision that they had in the simulation world.

The e-puck position data as recorded by the overhead camera and calculated by the custom colour detection software for these four experiments are shown in Figures 10.4, 10.7,

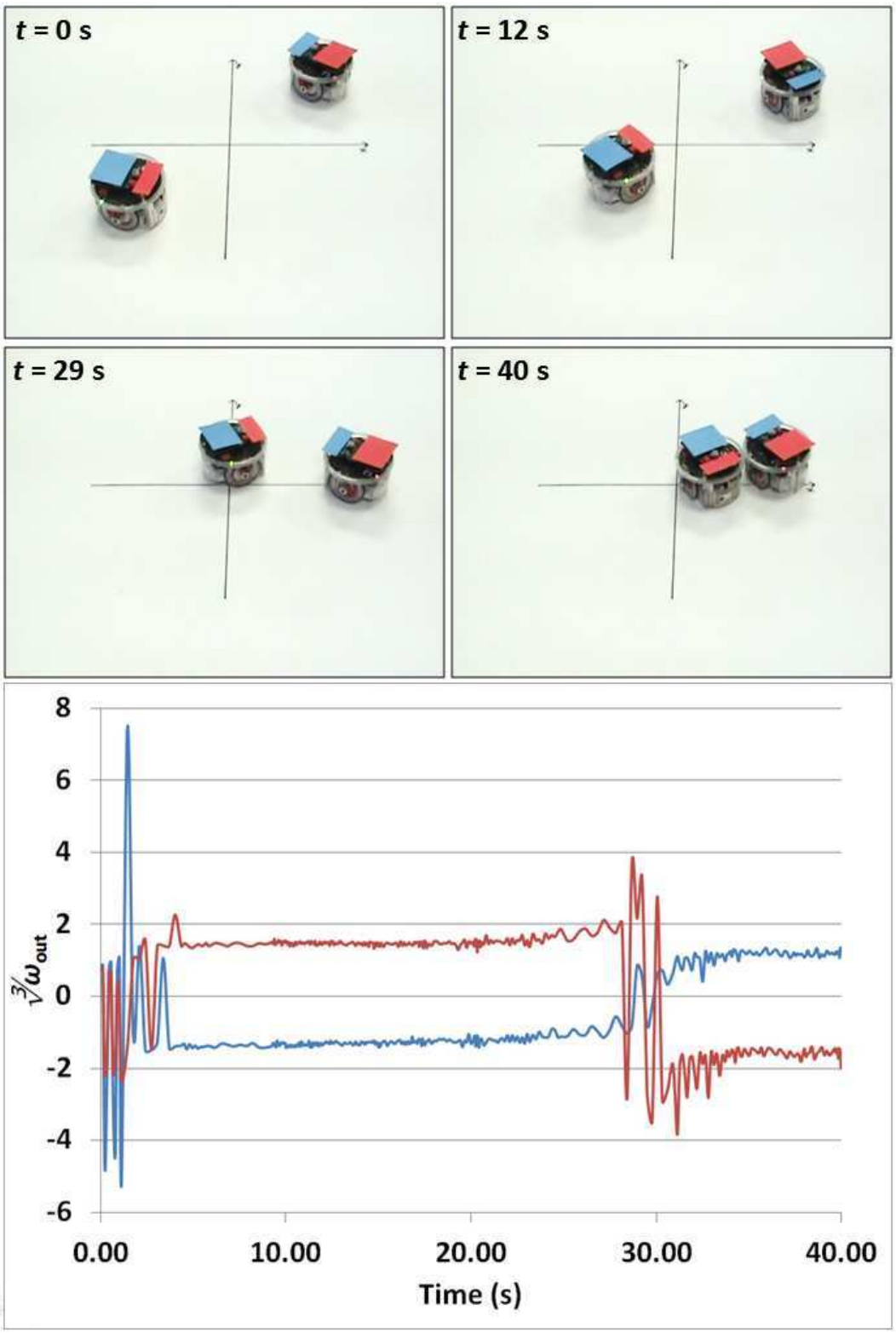


Figure 10.3: Video snapshots and communication data from a hardware experiment.

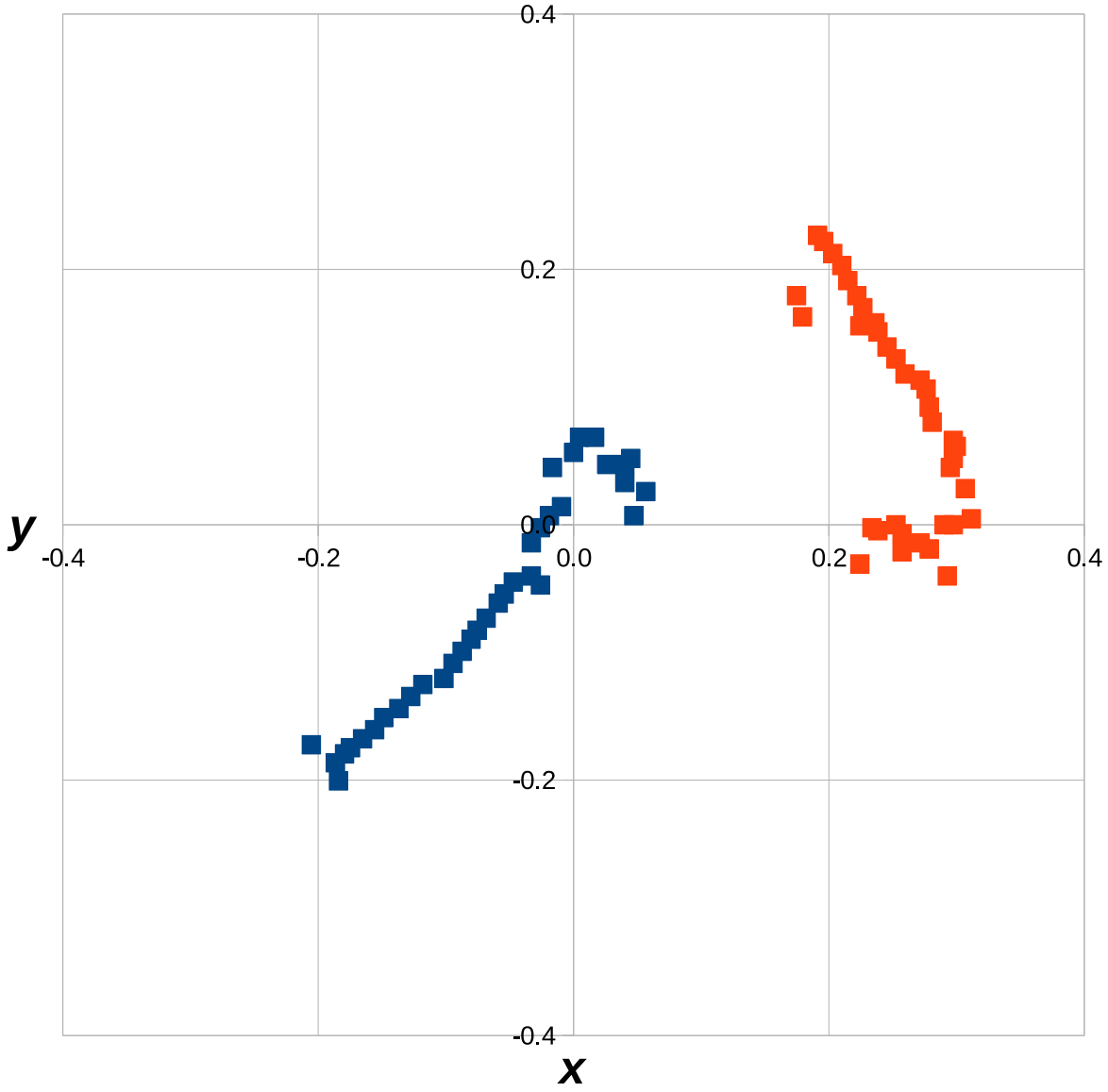


Figure 10.4: Position data from the overhead system as seen by the agents during the experiment shown in Figure 10.3.

10.10 and 10.13. These are the data that were used for each 10-timestep interval over the course of an experiment. For each timestep within a given interval, the agent would see its current position coordinates plus random values drawn from a Gaussian distribution with mean 0 and standard deviation 0.025. The data in Figure 10.13 further demonstrate that the embodied agents cannot maintain sufficiently close y positions to remain in "lateral" mode. Instead, the agents seem to overshoot each other's y positions twice.

While the e-puck position detection system operates at a high level of accuracy in general,

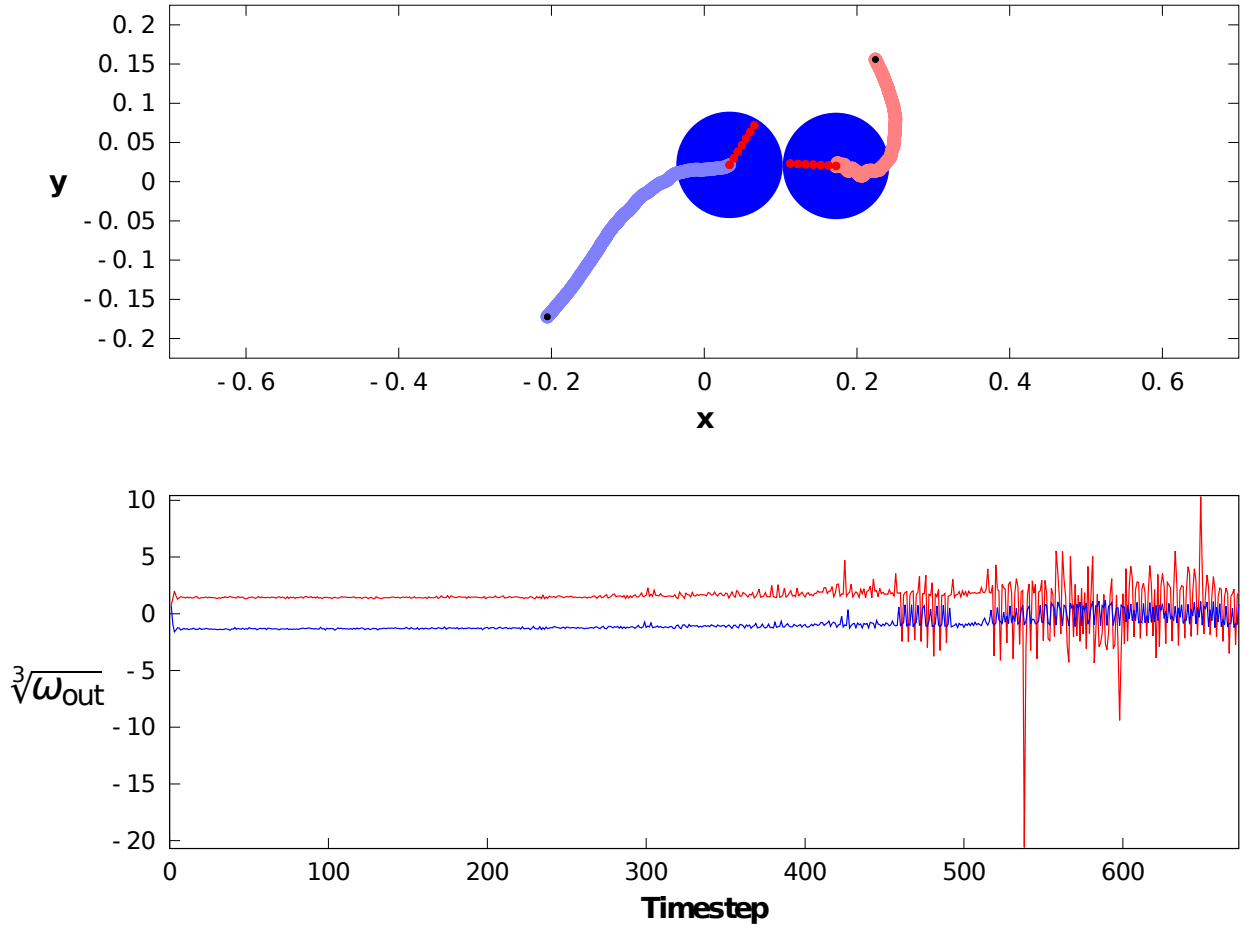


Figure 10.5: A simulation run using the same EMMs and initial conditions as the experiment shown in Figure 10.3.

there were several inaccurate readings, most commonly near the beginning of an experiment. These noisy points could have been problematic for the agents, especially considering that they were used as position data for 10 timesteps; however, the agents seemed to be relatively unfazed by these inaccuracies. This demonstrates the robustness of these EMM controllers.

This robustness is further exemplified by examining the average speeds of the embodied agents. In simulation, agents had an average speed of 0.0005 units per timestep, with a standard deviation of 1.25×10^{-5}. In the hardware experiments shown in Figure 10.3, however, the blue agent experienced an average speed of 0.0017 units per timestep, with a standard deviation of 0.0011. The red agent experienced an average speed of 0.0023 units per timestep, with a standard deviation of 0.0039. Because of these differences in speed, the agents find each other faster (in terms of number of timesteps) in the hardware experiments than in simulation (taking less than 370 timesteps in the hardware experiment shown in Figure 10.3 while taking 673 timesteps in the corresponding test simulation shown in Figure 10.5). These robot speeds were calculated from the position data (recorded every 10 timesteps) with the first

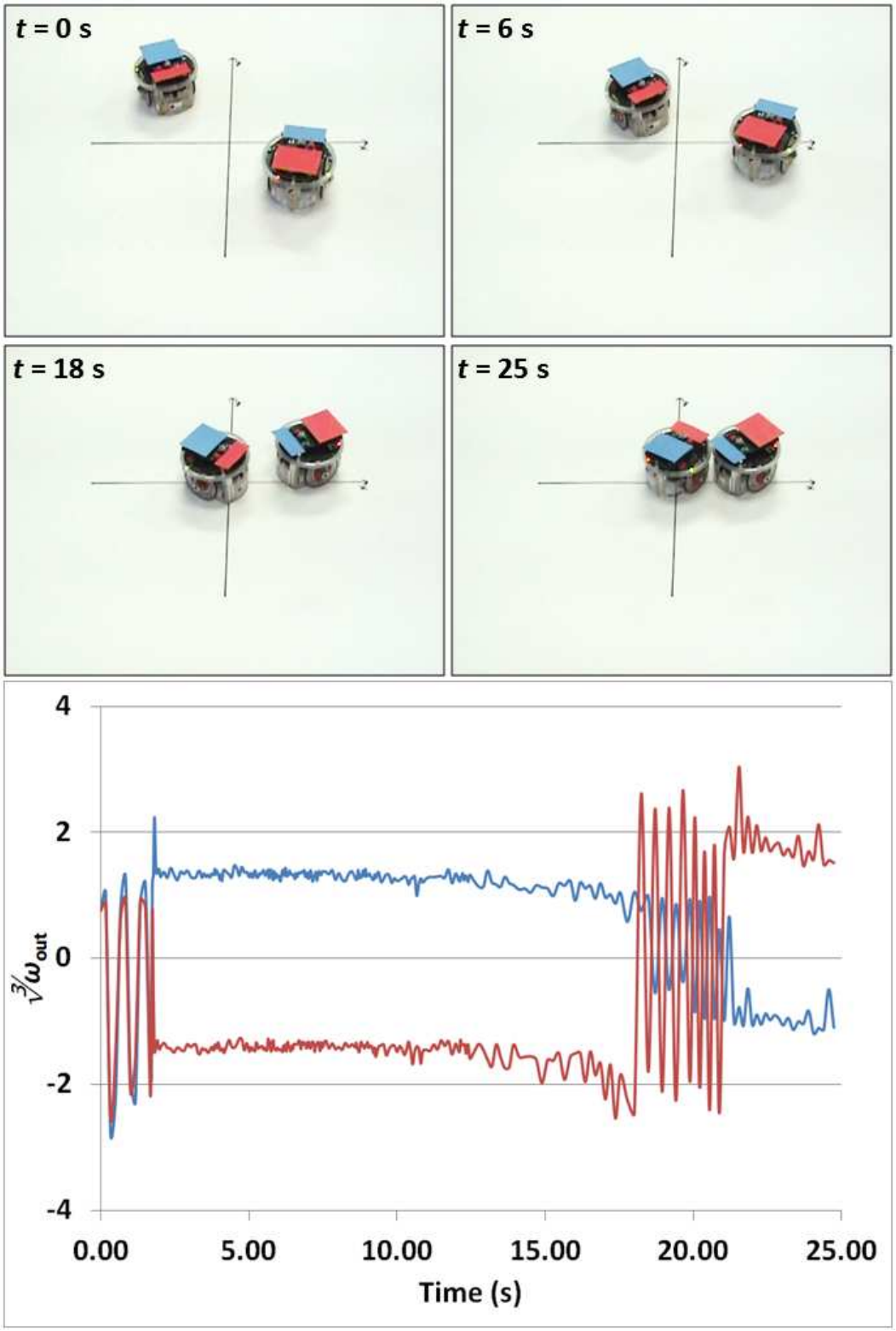


Figure 10.6: Video snapshots and communication data from a hardware experiment.

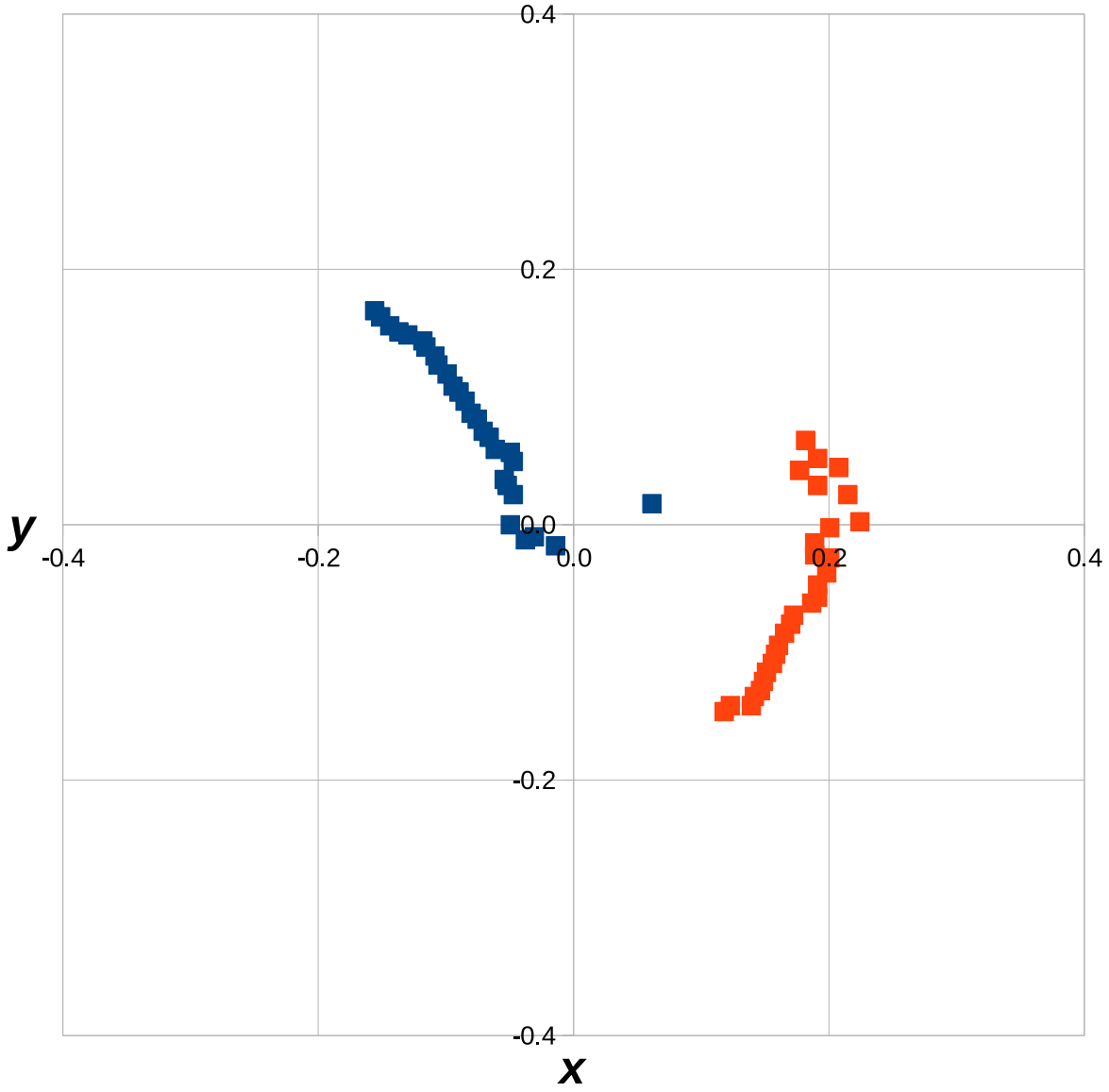


Figure 10.7: Position data from the overhead system as seen by the agents during the experiment shown in Figure 10.6.

point excluded for both agents, as it was obviously inaccurate. Data from all four hardware experiments and their simulation counterparts are presented in Table 10.1. Thus, not only are the embodied agents experiencing a much higher average speed than in simulation, these speeds are also more variable and differ between interacting agents. Nonetheless, these agents are able to adapt to these novel conditions and remain reproductively viable.

To evaluate further the performance of the embodied agents, the EMM used for the hardware experiments was ported back to the test simulations (i.e., the original EMM was

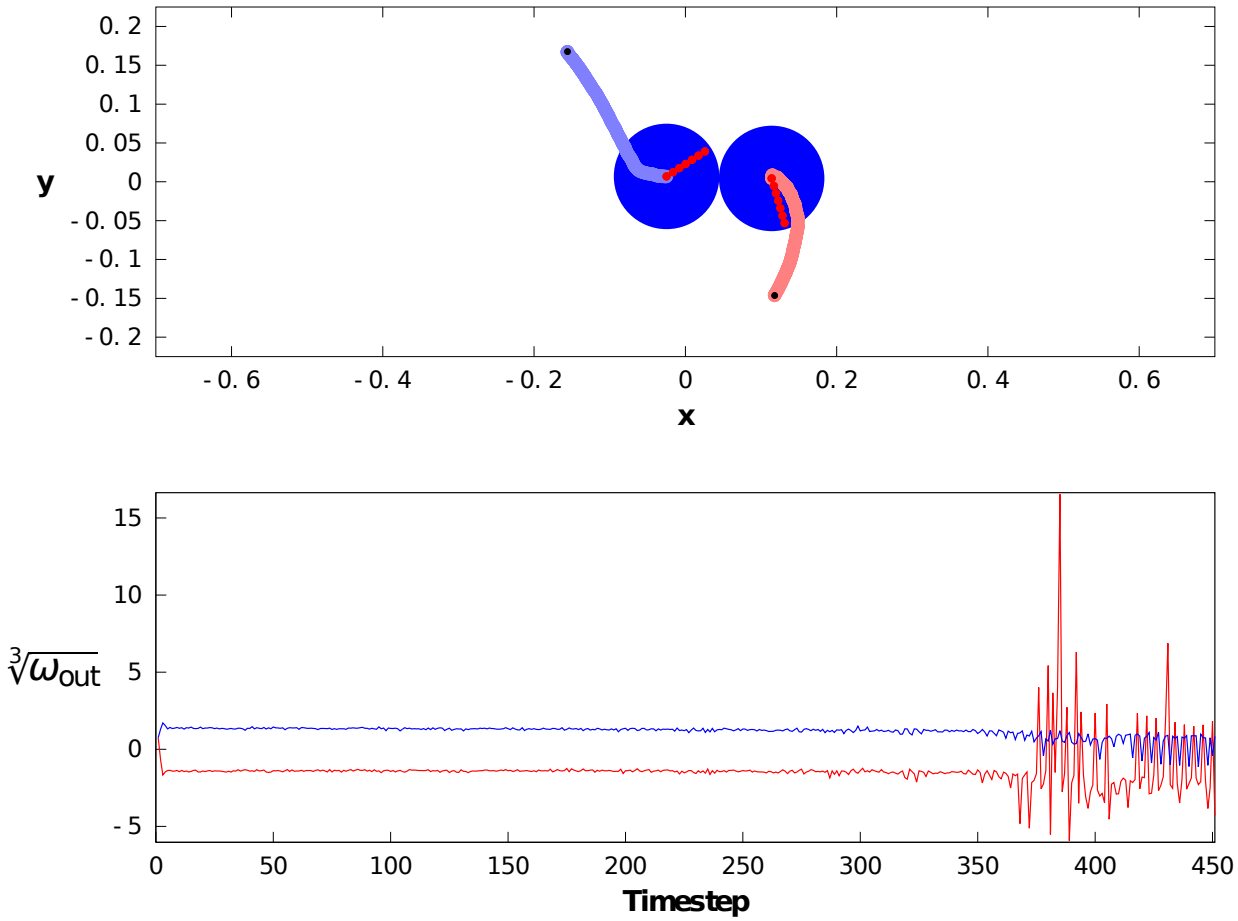


Figure 10.8: A simulation run using the same EMMs and initial conditions as the experiment shown in Figure 10.6.

Table 10.1: Robot speed data from hardware experiments. Initial inaccurate data points are omitted from the calculations (although subsequent inaccurate data points are included). The number of timesteps these EMM agents took to find each other in simulation from the same initial conditions as in the hardware experiments are also shown. Agents had an average speed of 0.0005 units per timestep, with a standard deviation of 1.25×10^{-5} in all simulation runs. We use μ to denote mean and σ to denote standard deviation.

Experiment	Blue Robot's Speed		Red Robot's Speed		# of Timesteps to Success	
	μ	σ	μ	σ	Hardware	Simulation
1	0.0017	0.0011	0.0023	0.0039	< 370	673
2	0.0011	0.0005	0.0019	0.0033	< 290	451
3	0.0016	0.0025	0.0016	0.0027	< 190	327
4	0.0015	0.0010	0.0053	0.0092	< 250	471

not used, instead the version used in the MATLAB implementation was copied into the test simulation code). Simulation experiments were then run using the same starting positions

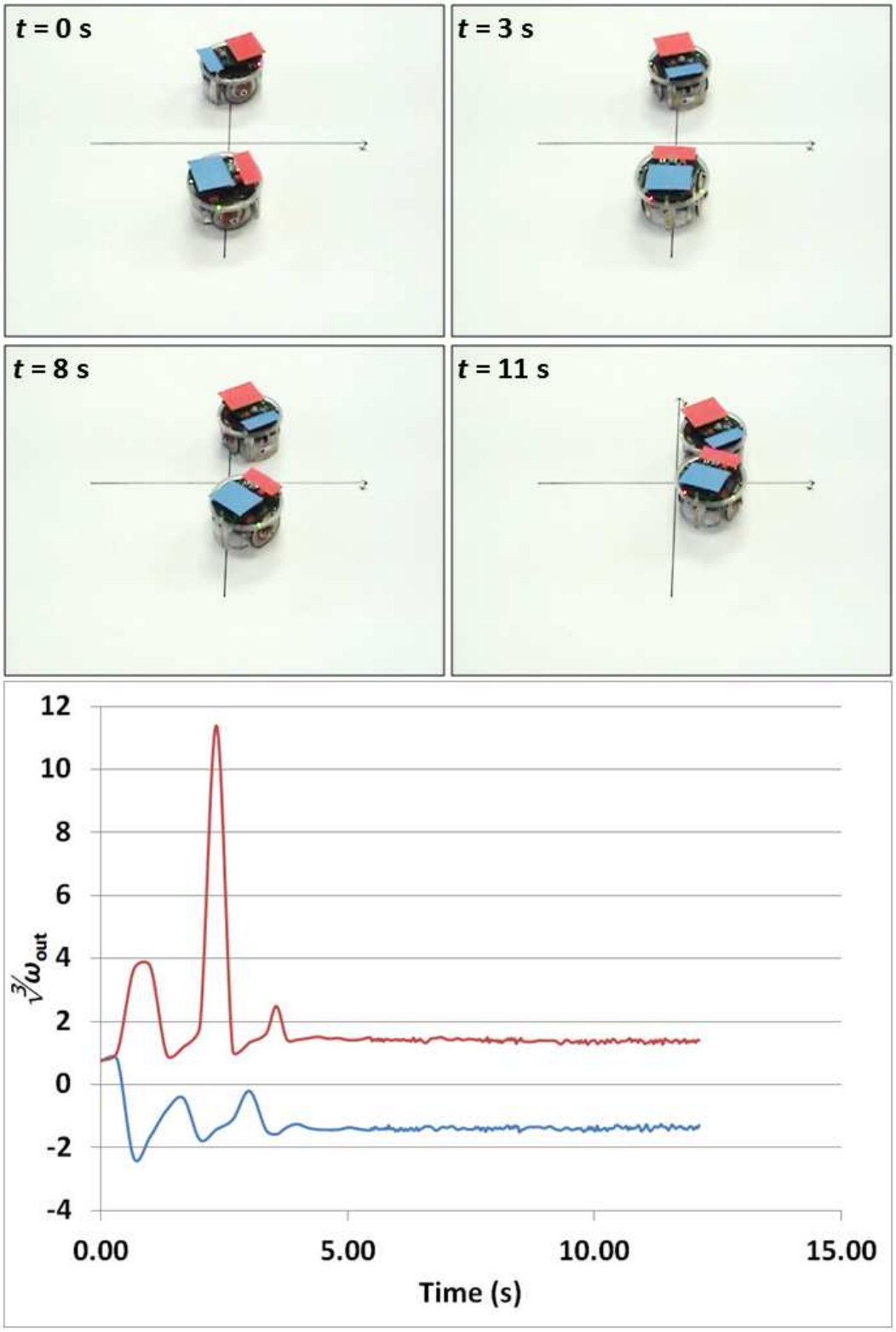


Figure 10.9: Video snapshots and communication data from a hardware experiment.

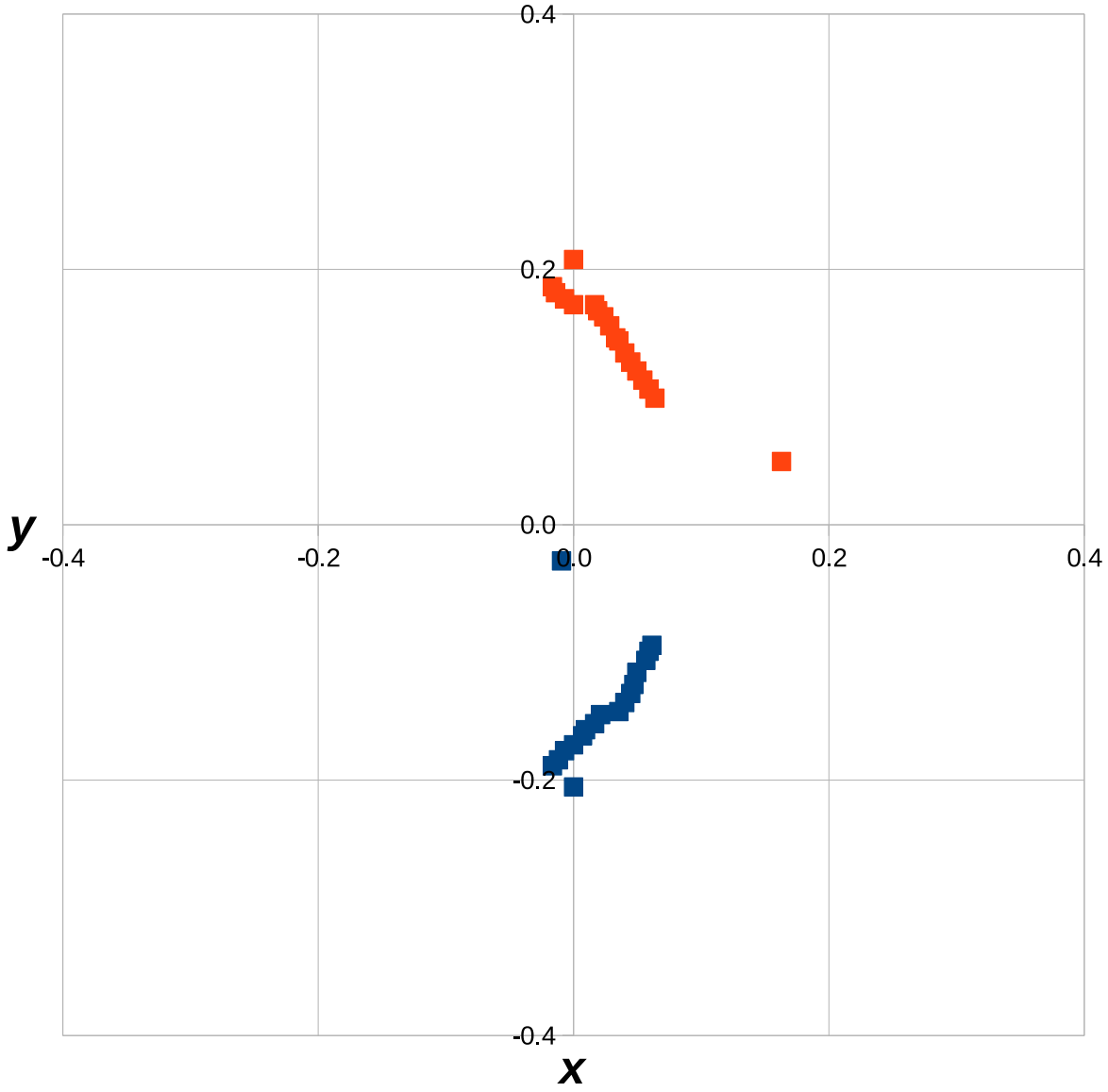


Figure 10.10: Position data from the overhead system as seen by the agents during the experiment shown in Figure 10.9.

as recorded by the overhead system during the hardware experiments. The results of these simulation runs are shown in Figures 10.5, 10.8, 10.11 and 10.14. Aside from the behavioural differences arising from the lack of position precision in the hardware experiments (which are most noticeable when comparing Figures 10.13 and 10.14), the embodied agents' movement patterns are very similar to those produced by their EMMs in simulation. This demonstrates that the EMMs were successfully ported to the e-puck hardware.

Other experiments were conducted where the experimenter repeatedly interfered with

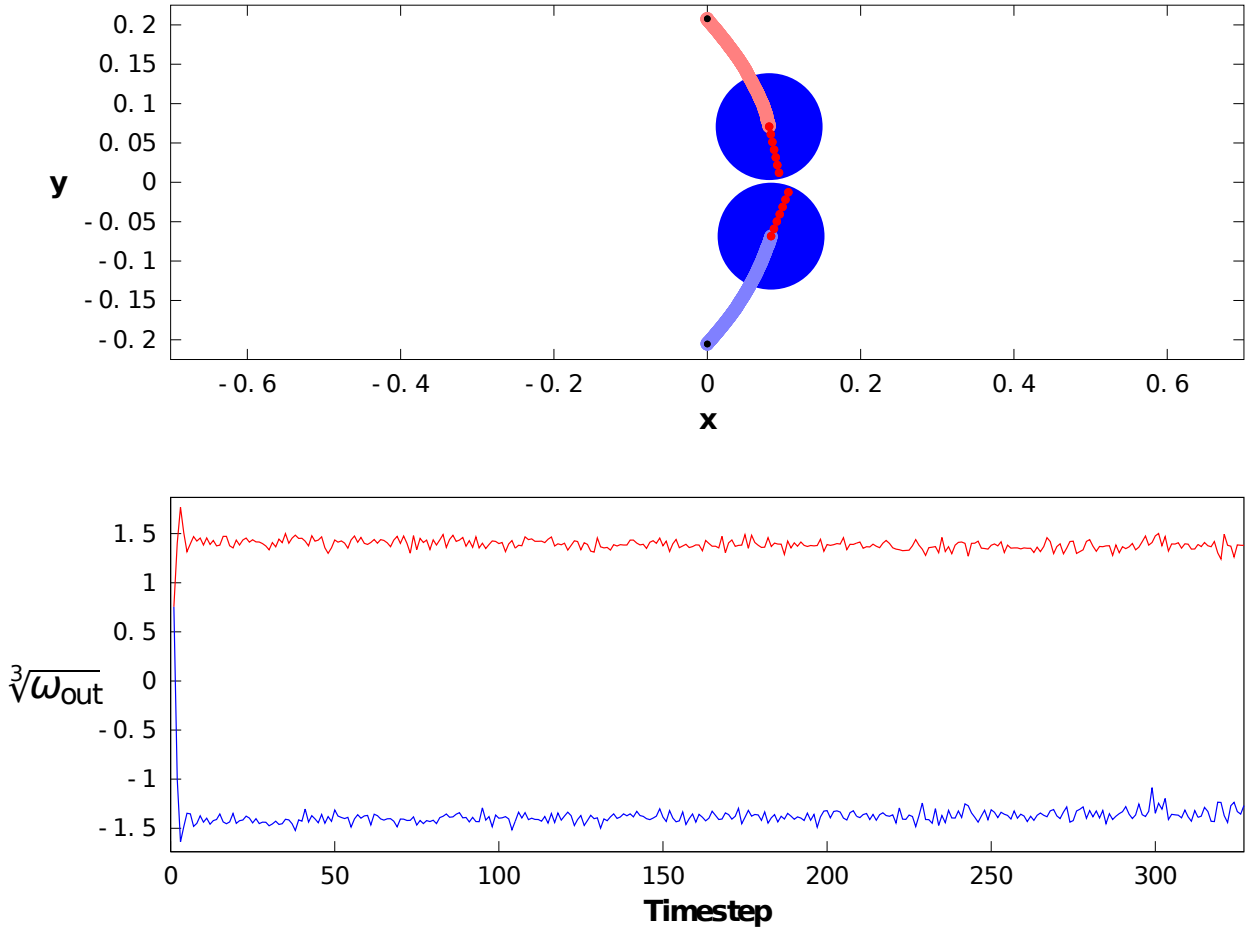


Figure 10.11: A simulation run using the same EMMs and initial conditions as the experiment shown in Figure 10.9.

the robots' movements, moving them to new random locations and temporarily obfuscating position tracking. Agents continually adjusted to these unforeseen circumstances (which would never have been experienced during the simulation run), managing to locate one another once the experimenter relented. This demonstrates the robustness of these EMM-based agents, as they are able to successfully adapt to novel, never-before-seen situations.

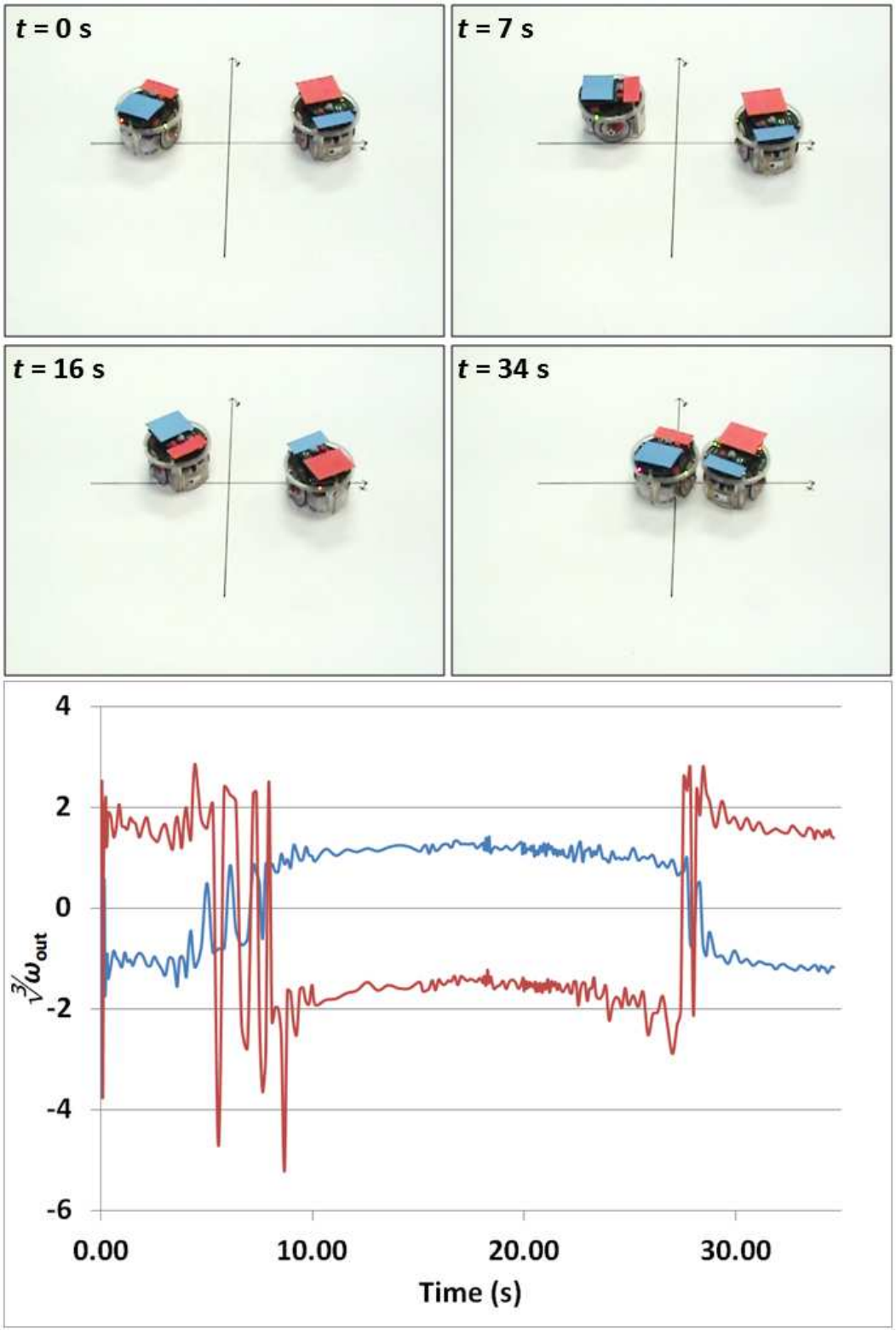


Figure 10.12: Video snapshots and communication data from a hardware experiment.

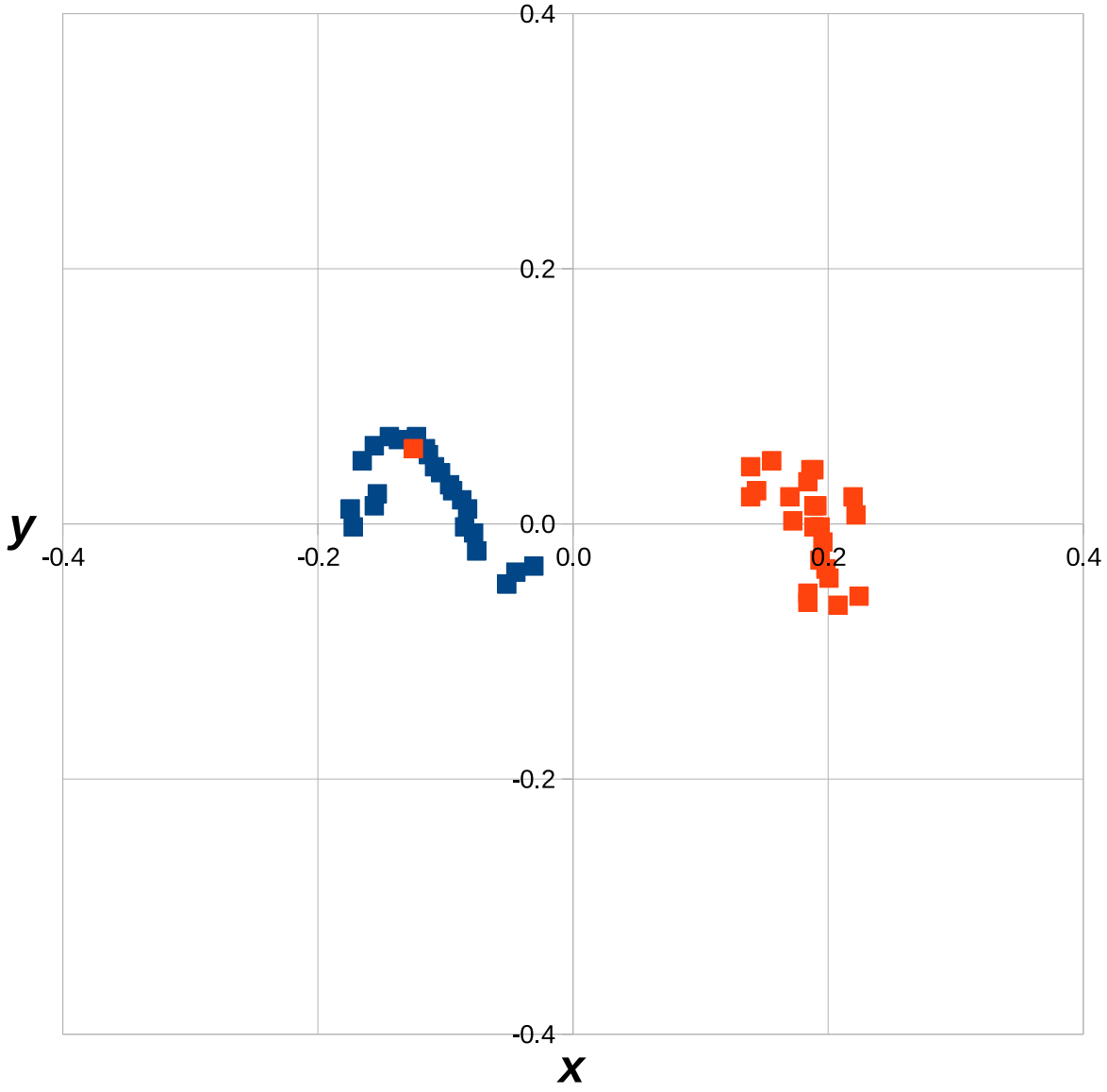


Figure 10.13: Position data from the overhead system as seen by the agents during the experiment shown in Figure 10.12.

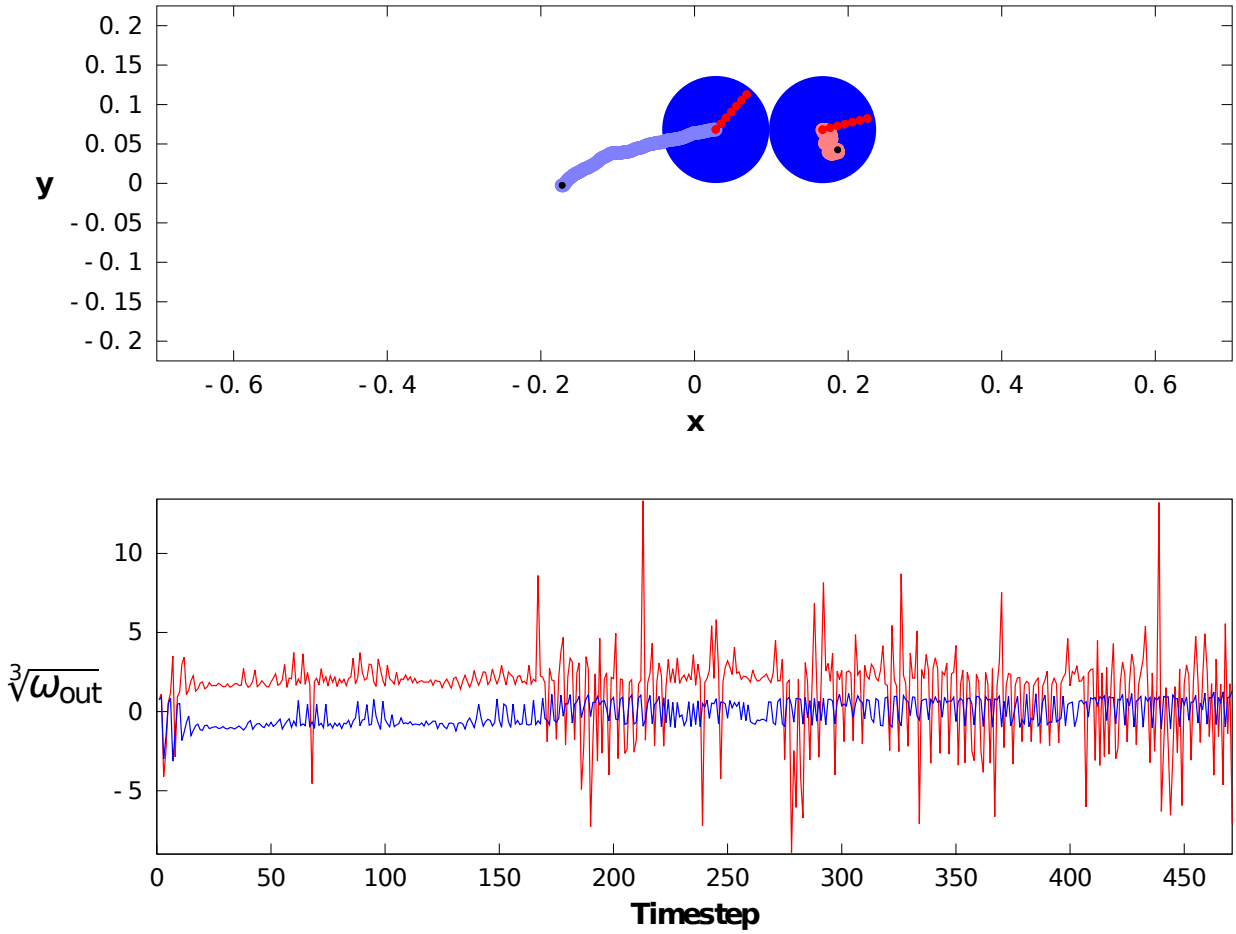


Figure 10.14: A simulation run using the same EMMs and initial conditions as the experiment shown in Figure 10.12.

Chapter 11

Conclusion of Part II

We have successfully constructed a simulation world where initially noncommunicating EMM-based agents repeatedly evolve complex dialogue-based cooperative communication, thus proving the hypothesis of Part II correct. This was accomplished through the creation of a novel Artificial Life simulation that we call *NoiseWorld*. Populations of simulated, finite-lifespan agents, controlled via EMMs, can move freely within a 2D world and automatically reproduce upon encountering other agents. The only means to detect other agents is through a one-dimensional, nondirectional real-valued communication channel. Cooperative communication emerges within this simulation from initially random EMMs that have no communication scheme specified *a priori*. Several (but not all) of these communication schemes evolve to a point where agents can locate each other from any initial configuration, demonstrating that both x and y position information is being communicated. As the communication channel is one- dimensional, this implies that 2D position information must be encoded in some fashion by the "sender" and then decoded into meaningful information (in terms of reproductive benefits) by the "receiver." Furthermore, all top communication schemes are explicitly dialogue-based, as agent communication outputs are directly modified by what they hear via their communication input. Finally, different simulation runs starting from different initial random populations produce similar but distinct communication schemes.

Evolutionary trajectories leading to top agents were examined via digital snapshots taken during simulation runs. It was found that agents first evolved to broadcast simple information about a single dimension, and then evolved to incorporate information about the second dimension into the communication scheme by using a different order of magnitude and dialogue-based communication. Evidence of the shaping of communication outputs by selection before communication has become fixed in the population was also uncovered.

A top agent from a simulation run was transferred to e-puck robotic hardware, where reproductively successful communication and movement behaviours were demonstrated from

a variety of starting configurations. The communications and movements were not identical to those seen in simulation, indicating that the EMMs (whose structure was fixed during hardware testing) were adapting their behaviours to the lack of precision in these real-world scenarios.

While previous works have successfully simulated the emergence and evolution of inter-agent communication, the majority of them were Evolutionary Robotics experiments with experimenter-defined fitness functions, explicit group selection, and discrete generations. Furthermore, in many of these experiments, communication signals contained inherent information, such as the direction and/or distance of the sender relative to the receiver. This makes it difficult to determine how much information, if any, is being transmitted via an evolved communication scheme. While the ALife implementation of Werner and Dyer [157] did not employ the aforementioned techniques, their results were relatively simple, with agents that live in a discrete world evolving a mapping between three-bit strings and three discretized movements (although we suspect that similar results could have been achieved using only two bits). The results presented here have no discrete generations, no enforced group selection and no explicit fitness functions. Furthermore, the world is continuous and there is no information inherent in the communication mechanisms. The top evolved communication schemes are explicitly dialogue-based and capable of transmitting both x and y position information over a 1D channel. Thus, these simulation results provide a unique opportunity to study the emergence and evolution of complex cooperative communication.

NoiseWorld opens up a large number of future research directions. The "language" of these evolved communication schemes has yet to be fully understood, as the dialogue property complicates analysis. However, the underlying behaviours are governed by explicit mathematical equations (EMMs), therefore rigorous analysis is possible. Further study on the emergence of communication in *NoiseWorld* would be beneficial to the understanding of the emergence of communication in nature. More fine-grained snapshots and analyses would shed more light on the "false starts" hypothesis described above. Also, further information entropy analysis may lead to a better understanding of information content in animal communication.

This simulation was developed with a focus on simplicity. Now that cooperative communication has emerged in the simple case, we can examine how additional elements affect the emergence and evolution of communication. Additions to the simulation might include predators and prey, an explicit concept of energy, and dividing agents into male and female. Furthermore, simulating the robotic agents with higher fidelity, e.g., giving agents direct control over their two wheels, would also add interesting complexity to the simulation. The ultimate goal is to develop a completely open-ended simulation. As it stands now, the evolution of the agents always reaches a plateau after which no new reproduction rate "high

scores" appear, even if the simulation run is extended for another 48 wall clock hours. A detailed, open-ended simulation world with predator/prey dynamics (i.e., coevolution) might be one way to avoid these evolutionary ceilings.

One of our goals in developing this simulation was to demonstrate the capabilities of EMMs by achieving novel results. We have achieved this with *NoiseWorld*, however, EMMs within the simulation should be replaced with ANN-based algorithms to see if they are capable of evolving comparable results. The question is not if ANNs can represent the results presented here, as we already know that CTRNNs can approximate any dynamics and are thus theoretically capable of approximating the behaviours of any EMM. The question is whether or not these results can be reproduced within *NoiseWorld* from randomly initialized ANNs. To put it another way, is there an evolutionary trajectory from randomly initialized ANNs to an ANN that approximates the top EMMs from these experiments?

A big question that arises from *NoiseWorld* is what can these artificially evolved communication schemes tell us about natural communication, and, more interestingly, natural language? What, if anything, can the parallels between the evolved communication schemes and their evolutionary trajectories (i.e., when communicating 2D information, agents communicate about one dimension and then the other, and this occurs in the order in which these variables first became fixed in the communication outputs of the evolving population) tell us? Can specific "meaning" be attributed to certain communication output values or combinations of values? These are highly interdisciplinary questions and it is hoped that scientists from various disciplines will take an interest and download *NoiseWorld* for themselves.

Chapter 12

Thesis Conclusion

A Novel Artificial Intelligence Paradigm

A novel artificial intelligence paradigm was developed and tested, both in simulation and in hardware. Our Evolvable Mathematical Models (EMMs) are composed of a series of equations represented in an artificial agent's genome as equation trees. Each equation can be composed of the four basic mathematical operators, addition, subtraction, multiplication and division, as well as agent inputs and outputs and numerical constants. These equations can approximate any analytic function. More sophisticated operations can also be used, however, exploring the benefits and drawbacks of additional operators is left to future studies. Agents have one equation modifying each of their experimenter defined outputs, and additional equations modifying extra state variables can be added through mutation. Other types of mutations can modify operators, constants, variables and entire subtrees. Two levels of sexual recombination, where an offspring genome is generated from two parent genomes, were also implemented. One occurs at the tree level, where subtrees from the two parents are grafted together, and the other one is an operation that occurs at the equation level, where an offspring genome is generated with equations from both parents. The EMM paradigm is based on Symbolic Regression (SR) in Genetic Programming (GP), where equations are evolved to fit data. The main difference between SR and EMMs is that while SR looks to model fixed data sets, EMMs look to artificially evolve lifelike behavioural patterns. Thus, the input data that an EMM sees varies depending on its own outputs, the environment in which it finds itself, and, in some cases, other EMMs.

Benchmarking Experiments

EMMs were first tested on the most common benchmark in Evolutionary Robotics, the double-pole balancing without velocity information benchmark. EMMs were able to solve this task, successfully balancing two poles of different lengths on a cart in simulation, using only pole angle and cart position information. It took EMMs roughly three times more fitness function evaluations to find a solution than the current state of the art in Evolutionary Robotics. The next benchmark was one that involved agent learning, the Double-T Maze with homing. This task has only previously been solved using Artificial Neural Networks (ANNs) with connection weight plasticity. Agents are required to search four different arms of a maze to find the one high reward. Once the arm containing this reward is found, successful agents should repeatedly revisit this arm in subsequent trials. If the high reward is then moved to a different arm, agents should return to their foraging behaviours to relocate the reward. This task requires that agents demonstrate learning behaviours without genetic modification, as their genomes remain unchanged throughout a given series of trials. EMMs were able to solve this task, demonstrating learning capabilities using fixed mathematical equations. Furthermore, neuromodulation-like behaviour (where learning is enabled and disabled via a specific signal) was seen in some of the top results, similar to results from previous work where special neuromodulation neurons were specified *a priori*. Thus we have shown that the EMM paradigm can match the results from ANN-based algorithms in two benchmarking tasks.

NoiseWorld

The next step was to achieve novel results using EMMs. The chosen task was the simulation of the emergence and evolution of interagent communication, as it is a notoriously hard problem spanning the fields of artificial intelligence, artificial life, and the natural evolution of communication and language, the last of which is a high controversial topic and the subject of ongoing debates among some of the world's top scientists. We developed a novel, open-source ALife simulation that we call *NoiseWorld*. Within this world, EMM-based agents can move freely in two dimensions and can receive real-valued communication outputs from their nearest neighbour. These communication values are nondirectional and are not attenuated by distance, therefore they contain no inherent information about the location of the sender. When two agents encounter one another in this world, they automatically produce one offspring containing genetic material from both parents. For every offspring born, one agent is selected at random to die, maintaining constant population sizes and producing a selection pressure that favours reproducing agents without the need for an explicit, experimenter-defined fitness function. As agents have no information about the whereabouts of their neighbours, this selection pressure engenders the emergence of interagent communication schemes to help these "blind" agents find one another. Simple communication schemes appear in all simulation runs, however in some simulations additional complexification occurs, allowing agents to communicate both their x and y position information via the provided one dimensional communication channel. Agents can use this information to locate one another from *any* starting configuration. These communication schemes are explicitly dialogue-based, a fact revealed by a cursory examination of the agent EMMs. The evolutionary trajectories leading up to these complex communications schemes were examined, yielding significant insight into the emergence and subsequent evolution of cooperative communication.

Hardware Experiments

Finally, by taking a top EMM agent from the *NoiseWorld* simulation and transferring it to e-puck robotic hardware, the transferability of EMMs from simulation to hardware was demonstrated. The embodied agents were successfully able to adapt their behaviours to the lower precision and higher noise of the hardware experiments, achieving reproductive success from a variety of initial configurations. The EMM paradigm seems to be readily transferrable to robotic hardware, an important trait for an AI implementation that hopes to produce real-world applications.

Final Thoughts

The overarching hypothesis behind Evolvable Mathematical Models is that it is possible to model the behaviour of biological agents with mathematics. The hypothesis of this thesis is less broad, however, postulating that it is possible to evolve mathematical equation-based controllers artificially that can demonstrate learning, adaptability and generalization capabilities *similar* to those seen in biological agents. The work described here has proven this thesis correct, demonstrating learning capabilities through the Double-T Maze experiments and adaptability and generalization capabilities via transfer of agents from simulation to hardware, as well as through various tests across all experiments described here. Furthermore, these mathematical equations evolved to produce complex, dialogue-based cooperative communication schemes, demonstrating their potential for producing lifelike behaviour.

This thesis is intended as a first step towards producing artificial intelligences as mathematical models of behaviour. While the capabilities of this paradigm have been elucidated via several challenging experiments, some big questions remain unanswered. Can we mathematically model animal behaviour? Can we mathematically model communication schemes at the level of complexity of human language? Can we model, and thus reproduce, consciousness? If the answer to any of these questions is "no," then this engenders a different line of questioning, namely, are there limitations to the modelling capabilities of mathematics? We are optimistic that the novel paradigm presented here can help answer some of these questions, and that it will lead to new and exciting results and applications in autonomous robotics, artificial intelligence, and artificial life.

Bibliography

[1] Patrick Abbot, Jun Abe, John Alcock, Samuel Alizon, Joao AC Alpedrinha, Malte Andersson, Jean-Baptiste Andre, Minus van Baalen, Francois Balloux, Sigal Balshine, et al. Inclusive fitness theory and eusociality. *Nature*, 471(7339):E1–E4, 2011.

[2] David H Ackley and Michael L Littman. Altruism in the evolution of communication. In *Artificial life IV*, pages 40–48, 1994.

[3] Giovanni Adorni, Stefano Cagnoni, and Monica Mordonini. Genetic programming of a goal-keeper control strategy for the robocup middle size competition. In *Genetic Programming*, pages 109–119. Springer, 1999.

[4] Christos Ampatzis, Elio Tuci, Vito Trianni, and Marco Dorigo. Evolution of signaling in a multi-robot system: Categorization and communication. *Adaptive Behavior*, 16(1):5–26, 2008.

[5] David Andre and Astro Teller. Evolving team darwin united. In *RoboCup-98: Robot soccer world cup II*, pages 346–351. Springer, 1999.

[6] Ion Androutsopoulos, Georgios Paliouras, Vangelis Karkaletsis, Georgios Sakkis, Constantine D Spyropoulos, and Panagiotis Stamatopoulos. Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach. *arXiv preprint cs/0009009*, 2000.

[7] Ramnik Arora, Rupesh Tulshyan, and Kalyanmoy Deb. Parallelization of binary and real-coded genetic algorithms on CUDA. *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.

[8] Sebastian Bader, Steffen Hölldobler, and Valentin Mayer-Eichberger. Extracting propositional rules from feed-forward neural networks-a new decompositional approach. In *NeSy*, 2007.

[9] Craig H Bailey, Maurizio Giustetto, Yan-You Huang, Robert D Hawkins, and Eric R Kandel. Is heterosynaptic modulation essential for stabilizing hebbian plasiticity and memory. *Nature Reviews Neuroscience*, 1(1):11–20, 2000.

[10] Wolfgang Banzhaf and Barry McMullin. Artificial life. In *Handbook of Natural Computing*, pages 1805–1834. Springer, 2012.

[11] Nils Aall Barricelli. Symbiogenetic evolution processes realized by artificial methods. *Methodos*, 9:143–182, 1957.

[12] Randall D Beer and John C Gallagher. Evolving dynamical neural networks for adaptive behavior. *Adaptive behavior*, 1(1):91–122, 1992.

[13] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[14] Jesper Blynel and Dario Floreano. Exploring the T-maze: Evolving learning-like robot behaviors using CTRNNs. In *Applications of evolutionary computing*, pages 593–604. Springer, 2003.

[15] Zdeněk Buk, Jan Koutník, and Miroslav Šnorek. NEAT in HyperNEAT substituted with genetic programming. In *Adaptive and Natural Computing Algorithms*, pages 243–252. Springer, 2009.

[16] Robert Burbidge, Joanne H Walker, and Myra S Wilson. Grammatical evolution of a robot controller. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 357–362. IEEE, 2009.

[17] Robert Burbidge and Myra S Wilson. Vector-valued function estimation by grammatical evolution for autonomous robot control. *Information Sciences*, 258:182–199, 2014.

[18] Angelo Cangelosi. Evolution of communication and language using signals, symbols, and words. *Evolutionary Computation, IEEE Transactions on*, 5(2):93–101, 2001.

[19] Angelo Cangelosi and Domenico Parisi. The emergence of a language in an evolving population of neural networks. *Connection Science*, 10(2):83–97, 1998.

[20] Angelo Cangelosi and Domenico Parisi. *Simulating the evolution of language*. Springer London, 2002.

[21] Erick Cantu-Paz. *Efficient and accurate parallel genetic algorithms*, volume 1. Springer, 2000.

[22] Erick Cantu-Paz and David E Goldberg. Efficient parallel genetic algorithms: theory and practice. *Computer methods in applied mechanics and engineering*, 186(2):221–238, 2000.

[23] Genci Capi and Kenji Doya. Evolution of neural architecture fitting environmental dynamics. *Adaptive Behavior*, 13(1):53–66, 2005.

[24] Shu-Heng Chen and Chung-Chih Liao. Agent-based computational modeling of the stock price-volume relation. *Information Sciences*, 170(1):75–100, 2005.

[25] Morten H Christiansen, Rick AC Dale, Michelle R Ellefson, and Christopher M Conway. The role of sequential learning in language evolution: Computational and experimental studies. In *Simulating the evolution of language*, pages 165–187. Springer, 2002.

[26] Morten H Christiansen and Simon Kirby. Language evolution: The hardest problem in science? 2003.

[27] Paul C Chu and John E Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics*, 4(1):63–86, 1998.

[28] Jeff Clune, Benjamin E Beckmann, Charles Ofria, and Robert T Pennock. Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 2764–2771. IEEE, 2009.

[29] Jeff Clune, Charles Ofria, and Robert T Pennock. The sensitivity of HyperNEAT to different geometric representations of a problem. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 675–682. ACM, 2009.

[30] James P Cohoon, Shailesh U Hegde, Worthy N Martin, and D Richards. Punctuated equilibria: a parallel genetic algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 148–154. L. Erlbaum Associates Inc., 1987.

[31] David B D'Ambrosio, Skyler Goodell, Joel Lehman, Sebastian Risi, and Kenneth O Stanley. Multirobot behavior synchronization through direct neural network communication. In *Intelligent Robotics and Applications*, pages 603–614. Springer, 2012.

[32] David B D'Ambrosio, Joel Lehman, Sebastian Risi, and Kenneth O Stanley. Evolving policy geometry for scalable multiagent learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages

731–738. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[33] David B D'Ambrosio and Kenneth O Stanley. Generative encoding for multiagent learning. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 819–826. ACM, 2008.

[34] Charles Darwin. *The descent of man*. D. Appleton and Company, 1871.

[35] Richard Dawkins and John R Krebs. Animal signals: information or manipulation? *Behavioural ecology: An evolutionary approach*, 2:282–309, 1978.

[36] Joachim de Greeff and Stefano Nolfi. Evolution of implicit and explicit communication in mobile robots. In *Evolution of communication and language in embodied agents*, pages 179–214. Springer, 2010.

[37] Kenneth A De Jong. *Evolutionary computation: a unified approach*. MIT press Cambridge, 2006.

[38] Ezequiel A Di Paolo. A little more than kind and less than kin: The unwarranted use of kin selection in spatial models of communication. In *Advances in artificial life*, pages 504–513. Springer, 1999.

[39] Ezequiel A Di Paolo. Behavioral coordination, structural congruence and entrainment in a simulation of acoustically coupled agents. *Adaptive Behavior*, 8(1):27–48, 2000.

[40] Ezequiel A Di Paolo. Evolving spike-timing-dependent plasticity for single-trial learning in robots. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 361(1811):2299–2319, 2003.

[41] Bogdan Draganski, Christian Gaser, Volker Busch, Gerhard Schuierer, Ulrich Bogdahn, and Arne May. Neuroplasticity: changes in grey matter induced by training. *Nature*, 427(6972):311–312, 2004.

[42] Jan Drchal, Jan Koutník, and Miroslav Snorek. HyperNEAT controlled robots learn how to drive on roads in simulated environment. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 1087–1092. IEEE, 2009.

[43] P Durr, Claudio Mattiussi, Andrea Soltoggio, and Dario Floreano. Evolvability of neuromodulated learning for robots. In *Learning and Adaptive Behaviors for Robotic Systems, 2008. LAB-RS'08. ECSIS Symposium on*, pages 41–46. IEEE, 2008.

[44] Peter Dürr, Claudio Mattiussi, and Dario Floreano. Neuroevolution with analog genetic encoding. In *Parallel Problem Solving from Nature-PPSN iX*, pages 671–680. Springer, 2006.

[45] David B Dusenbery. *Sensory ecology: how organisms acquire and respond to information*. WH Freeman New York, 1992.

[46] Niles Eldredge and Stephen Jay Gould. Punctuated equilibria: an alternative to phyletic gradualism. *Models in paleobiology*, 82:115, 1972.

[47] Francisco Fernández, Marco Tomassini, and Leonardo Vanneschi. An empirical study of multipopulation genetic programming. *Genetic Programming and Evolvable Machines*, 4(1):21–51, 2003.

[48] D. Floreano and F. Mondada. Evolution of plastic neurocontrollers for situated agents. In *4th International Conference on Simulation of Adaptive Behavior (SAB'1996)*. MA: MIT Press, 1996.

[49] Dario Floreano, Peter Durr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1:47–62, 2008.

[50] Dario Floreano, Phil Husbands, and Stefano Nolfi. Evolutionary robotics. *Springer handbook of robotics*, pages 1423–1451, 2008.

[51] Dario Floreano and Claudio Mattiussi. Evolution of spiking neural controllers for autonomous vision-based robots. In *Evolutionary Robotics. From Intelligent Robotics to Artificial Life*, pages 38–61. Springer, 2001.

[52] Dario Floreano, Sara Mitri, Stéphane Magnenat, and Laurent Keller. Evolutionary conditions for the emergence of communication in robots. *Current biology*, 17(6):514–519, 2007.

[53] Dario Floreano and Joseba Urzelai. Evolutionary robots with on-line self-organization and behavioral fitness. *Neural Networks*, 13(4):431–443, 2000.

[54] Ken-ichi Funahashi and Yuichi Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks*, 6(6):801–806, 1993.

[55] Jason Gauci and Kenneth O Stanley. Indirect encoding of neural networks for scalable Go. In *Parallel Problem Solving from Nature, PPSN XI*, pages 354–363. Springer, 2010.

[56] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. *Urbana*, 51:61801–2996, 1991.

[57] Faustino Gomez and Risto Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5(3-4):317–342, 1997.

[58] Faustino J Gomez and Risto Miikkulainen. Solving non-markovian control tasks with neuroevolution. In *IJCAI*, volume 99, pages 1356–1361, 1999.

[59] Stephen Jay Gould and Richard C Lewontin. The spandrels of San Marco and the Panglossian paradigm: a critique of the adaptationist programme. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 205(1161):581–598, 1979.

[60] Paul Grouchy and Gabriele MT D'Eleuterio. Supplanting neural networks with ODEs in evolutionary robotics. In *Simulated Evolution and Learning*, pages 299–308. Springer, 2010.

[61] Paul Grouchy and Hod Lipson. Evolution of self-replicating cube conglomerations in a simulated 3D environment. In *Artificial Life*, volume 13, pages 59–66, 2012.

[62] Paul Grouchy, Jekanthan Thangavelautham, and Gabriele MT D'Eleuterio. An island model for high-dimensional genomes using phylogenetic speciation and species barcoding. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1355–1362. ACM, 2009.

[63] Frederic Gruau, Darrell Whitley, and Larry Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Proceedings of the First Annual Conference on Genetic Programming*, pages 81–89. MIT Press, 1996.

[64] Steven Gustafson and Edmund K Burke. The speciating island model: An alternative parallel evolutionary algorithm. *Journal of Parallel and Distributed Computing*, 66(8):1025–1036, 2006.

[65] Simon Harding and Wolfgang Banzhaf. Fast genetic programming and artificial developmental systems on GPUs. In *High Performance Computing Systems and Applications, 2007. HPCS 2007. 21st International Symposium on*, pages 2–2. IEEE, 2007.

[66] Marc D Hauser. *The evolution of communication*. The MIT Press, 1996.

[67] Marc D Hauser, Noam Chomsky, and W Tecumseh Fitch. The faculty of language: What is it, who has it, and how did it evolve? *science*, 298(5598):1569–1579, 2002.

[68] Donald O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Wiley, New York, 1949.

[69] Geoffrey E Hinton and Terrence Joseph Sejnowski. *Unsupervised learning: foundations of neural computation*. 1999.

[70] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952.

[71] John H Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press, 1975.

[72] Gregory S Hornby and Jordan B Pollack. Evolving L-systems to generate virtual creatures. *Computers & Graphics*, 25(6):1041–1048, 2001.

[73] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[74] Daniel Howard, Simon C. Roberts, and Conor Ryan. Pragmatic genetic programming strategy for the problem of vehicle detection in airborne reconnaissance. *Pattern Recognition Letters*, 27(11):1275–1288, 2006.

[75] Gerard Howard, Ella Gale, Larry Bull, Ben de Lacy Costello, and Andy Adamatzky. Evolution of plastic learning in spiking networks via memristive connections. *Evolutionary Computation, IEEE Transactions on*, 16(5):711–729, 2012.

[76] Christian Igel. Neuroevolution for reinforcement learning using evolution strategies. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 4, pages 2588–2595. IEEE, 2003.

[77] Eduardo Izquierdo, Inman Harvey, and Randall D Beer. Associative learning on a continuum in evolved dynamical neural networks. *Adaptive Behavior*, 16(6):361–384, 2008.

[78] Blynel Jesper and Dario Floreano. Levels of dynamics and adaptive behavior in evolutionary neural controllers. In *From Animals to Animats 7: Proceedings of the Seventh International Conference on Simulation of Adaptive Behavior*, volume 7, page 272. MIT Press, 2002.

[79] Fei Jiang, Hugues Berry, and Marc Schoenauer. Supervised and evolutionary learning of echo state networks. In *Parallel Problem Solving from Nature–PPSN X*, pages 215–224. Springer, 2008.

[80] George H John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345. Morgan Kaufmann Publishers Inc., 1995.

[81] Mak Kaboudan. Extended daily exchange rates forecasts using wavelet temporal resolutions. *New Mathematics and Natural Computing*, 1(1):79–107, 2005.

[82] Yohannes Kassahun, Jose de Gea, Mark Edgington, Jan Hendrik Metzen, and Frank Kirchner. Accelerating neuroevolutionary methods using a Kalman filter. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1397–1404. ACM, 2008.

[83] Simon Kirby. Spontaneous evolution of linguistic structure-an iterated learning model of the emergence of regularity and irregularity. *Evolutionary Computation, IEEE Transactions on*, 5(2):102–110, 2001.

[84] David B. Knoester, Philip K. McKinley, Benjamin Beckmann, and Charles Ofria. Directed evolution of communication and cooperation in digital organisms. In Fernando Almeida e Costa, Luis Mateus Rocha, Ernesto Costa, Inman Harvey, and Antnio Coutinho, editors, *ECAL*, pages 384–394. Springer, 2007.

[85] John R Koza. Genetic programming: on the programming of computers by means of natural selection (complex adaptive systems), 1992.

[86] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, may 1994.

[87] John R Koza, Martin A Keane, Jessen Yu, Forrest H Bennett III, and William Mydlowec. Automatic creation of human-competitive programs and controllers by means of genetic programming. *Genetic Programming and Evolvable Machines*, 1(1-2):121–164, 2000.

[88] John R Koza and James P Rice. Automatic programming of robots using genetic programming. In *AAAI*, volume 92, pages 194–207, 1992.

[89] Chris G Langton. Computation at the edge of chaos: Phase transitions and emergent computation. *Physica D: Nonlinear Phenomena*, 42(1):12–37, 1990.

[90] Christopher G Langton. Studying artificial life with cellular automata. *Physica D: Nonlinear Phenomena*, 22(1):120–149, 1986.

[91] Pedro Larranaga, Cindy M. H. Kuijpers, Roberto H. Murga, Inaki Inza, and Sejla Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13(2):129–170, 1999.

[92] Richard E Lenski, Charles Ofria, Robert T Pennock, and Christoph Adami. The evolutionary origin of complex features. *Nature*, 423(6936):139–144, 2003.

[93] Jason Lohn, Gregory Hornby, and Derek Linden. An evolved antenna for deployment on NASA's space technology 5 mission. In Una-May O'Reilly, Tina Yu, Rick L. Riolo, and Bill Worzel, editors, *Genetic Programming Theory and Practice II*, chapter 18, pages 301–315. Springer, Ann Arbor, 2004.

[94] Chris Loken, Daniel Gruner, Leslie Groer, Richard Peltier, Neil Bunn, Michael Craig, Teresa Henriques, Jillian Dempsey, Ching-Hsing Yu, Joseph Chen, et al. Scinet: lessons learned from building a power-efficient top-20 system and data centre. In *Journal of Physics: Conference Series*, volume 256, page 012026. IOP Publishing, 2010.

[95] Santosh Manicka. Analysis of evolved agents performing referential communication. In *Artificial Life*, volume 13, pages 393–400, 2012.

[96] Hubert Markl. Manipulation, modulation, information, cognition: some of the riddles of communication. 1985.

[97] Davide Marocco, Angelo Cangelosi, and Stefano Nolfi. The emergence of communication in evolutionary robots. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 361(1811):2397–2421, 2003.

[98] Davide Marocco and Stefano Nolfi. Origins of communication in evolving robots. In *From Animals to Animats 9*, pages 789–803. Springer, 2006.

[99] Davide Marocco and Stefano Nolfi. Self-organization of communication in evolving robots. In *Proceedings of the Conference on Artificial Life (ALIFE)*, pages 178–184, 2006.

[100] Davide Marocco and Stefano Nolfi. Emergence of communication in embodied agents evolved for the ability to solve a collective navigation problem. *Connection Science*, 19(1):53–74, 2007.

[101] Claudio Mattiussi and Dario Floreano. Evolution of analog networks using local string alignment on highly reorganizable genomes. In *Evolvable Hardware, 2004. Proceedings. 2004 NASA/DoD Conference on*, pages 30–37. IEEE, 2004.

[102] Marco Mirolli and Stefano Nolfi. Evolving communication in embodied agents: Theory, methods, and evaluation. In *Evolution of Communication and Language in Embodied Agents*, pages 105–121. Springer, 2010.

[103] Marco Mirolli and Domenico Parisi. How can we explain the emergence of a language that benefits the hearer but not the speaker? *Connection Science*, 17(3-4):307–324, 2005.

[104] Marco Mirolli and Domenico Parisi. How producer biases can favor the evolution of communication: an analysis of evolutionary dynamics. *Adaptive Behavior*, 16(1):27–52, 2008.

[105] Melanie Mitchell, James P Crutchfield, and Peter T Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D: Nonlinear Phenomena*, 75(1):361–391, 1994.

[106] Graeme J Mitchison and Nicholas V Swindale. Can Hebbian volume learning explain discontinuities in cortical maps? *Neural computation*, 11(7):1519–1526, 1999.

[107] Sara Mitri, Dario Floreano, and Laurent Keller. The evolution of information suppression in communicating robots with conflicting interests. *Proceedings of the National Academy of Sciences*, 106(37):15786–15790, 2009.

[108] Sara Mitri, Dario Floreano, and Laurent Keller. Relatedness influences signal reliability in evolving robots. *Proceedings of the Royal Society B: Biological Sciences*, 278(1704):378–383, 2011.

[109] Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptocz, Stéphane Magnenat, Jean-Christophe Zufferey, Dario Floreano, and Alcherio Martinoli. The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th conference on autonomous robot systems and competitions*, volume 1, pages 59–65, 2009.

[110] Hirotaka Moriguchi and Hod Lipson. Learning symbolic forward models for robotic motion planning and control. In *Proceedings of European Conference of Artificial Life (ECAL 2011)*, pages 558–564, 2011.

[111] J-B Mouret and Stéphane Doncieux. Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 1161–1168. IEEE, 2009.

[112] John von Neumann and Arthur W Burks. Theory of self-reproducing automata. 1966.

[113] Yael Niv, Daphna Joel, Isaac Meilijson, and Eytan Ruppin. Evolution of reinforcement learning in uncertain environments: A simple explanation for complex foraging behaviors. *Adaptive Behavior*, 10(1):5–24, 2002.

[114] S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. Cambridge, MA: MIT Press/Bradford Books, 2000.

[115] Stefano Nolfi and Marco Mirolli. *Evolution of Communication and Language in Embodied Agents*. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2010.

[116] Stefano Nolfi and Domenico Parisi. Learning to adapt to changing environments in evolving neural networks. *Adaptive Behavior*, 5:75–98, 1997.

[117] Peter Nordin and Wolfgang Banzhaf. Genetic programming controlling a miniature robot. In *Working Notes for the AAAI Symposium on Genetic Programming*, pages 61–67, 1995.

[118] Peter Nordin and Wolfgang Banzhaf. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behavior*, 5(2):107–140, 1997.

[119] Martin A Nowak, Natalia L Komarova, and Partha Niyogi. Evolution of universal grammar. *Science*, 291(5501):114–118, 2001.

[120] Martin A Nowak, Corina E Tarnita, and Edward O Wilson. The evolution of eusociality. *Nature*, 466(7310):1057–1062, 2010.

[121] Charles Ofria and Claus O. Wilke. Avida: A software platform for research in computational evolutionary biology. *Artificial Life*, 10(2):191–229, 2004.

[122] Michael Oliphant. The dilemma of Saussurean communication. *BioSystems*, 37(1):31–38, 1996.

[123] Mihai Oltean and Crina Grosan. A comparison of several linear genetic programming techniques. *Complex Systems*, 14(4):285–314, 2003.

[124] Alvaro Pascual-Leone, Amir Amedi, Felipe Fregni, and Lotfi B Merabet. The plastic human brain cortex. *Annu. Rev. Neurosci.*, 28:377–401, 2005.

[125] Steven Pinker and Paul Bloom. Natural language and natural selection. *The adapted mind*, pages 451–493, 1992.

[126] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A Field Guide to Genetic Programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[127] Justin K Pugh, Skyler Goodell, and Kenneth O Stanley. Directional communication in evolved multiagent teams. 2013.

[128] Matt Quinn. Evolving communication without dedicated communication channels. In *Advances in Artificial Life*, pages 357–366. Springer, 2001.

[129] Sebastian Risi, Charles E Hughes, and Kenneth O Stanley. Evolving plastic neural networks with novelty search. *Adaptive Behavior*, 18(6):470–491, 2010.

[130] Sebastian Risi, Joel Lehman, and Kenneth O Stanley. Evolving the placement and density of neurons in the HyperNEAT substrate. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 563–570. ACM, 2010.

[131] Sebastian Risi and Kenneth O Stanley. Indirectly encoding neural plasticity as a pattern of local rules. In *From Animals to Animats 11*, pages 533–543. Springer, 2010.

[132] Sebastian Risi and Kenneth O Stanley. A unified approach to evolving plasticity and neural geometry. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8. IEEE, 2012.

[133] Sebastian Risi, Sandy D Vanderbleek, Charles E Hughes, and Kenneth O Stanley. How novelty search escapes the deceptive trap of learning to learn. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 153–160. ACM, 2009.

[134] Marek Ruciński, Dario Izzo, and Francesco Biscani. On the impact of the migration topology on the island model. *Parallel Computing*, 36(10):555–571, 2010.

[135] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, April 2009.

[136] Michael D Schmidt, Ravishankar R Vallabhajosyula, Jerry W Jenkins, Jonathan E Hood, Abhishek S Soni, John P Wikswo, and Hod Lipson. Automated refinement and inference of analytical models for metabolic networks. *Physical biology*, 8(5):055011, 2011.

[137] Jimmy Secretan, Nicholas Beato, David B D Ambrosio, Adelein Rodriguez, Adam Campbell, and Kenneth O Stanley. Picbreeder: evolving pictures collaboratively online. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1759–1768. ACM, 2008.

[138] Shital C. Shah and Andrew Kusiak. Data mining and genetic algorithm based gene/SNP selection. *Artificial Intelligence in Medicine*, 31(3):183–196, 2004.

[139] Zbigniew Skolicki and Kenneth De Jong. The influence of migration sizes and intervals on island models. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1295–1302. ACM, 2005.

[140] John Maynard Smith. *The theory of evolution*. Cambridge University Press, 1993.

[141] Andrea Soltoggio. Neuromodulation increases decision speed in dynamic environments. In *Schlesinger, M., Berthouze, L., and Balkenius, C.(2008): Proceedings of the Eighth International Conference on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems. Lund University Cognitive Studies*, volume 139, 2008.

[142] Andrea Soltoggio, John A Bullinaria, Claudio Mattiussi, Peter Dürr, and Dario Floreano. Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios. In *Artificial Life*, volume 11, pages 569–576, 2008.

[143] Andrea Soltoggio, Peter Durr, Claudio Mattiussi, and Dario Floreano. Evolving neuromodulatory topologies for reinforcement learning-like problems. In *IEEE Congress on Evolutionary Computation (CEC 2007) 25-28 Sept. 2007*, pages 2471–2478. IEEE Press, 2007.

[144] Andrea Soltoggio and Ben Jones. Novelty of behaviour as a basis for the neuro-evolution of operant reward learning. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 169–176. ACM, 2009.

[145] Kenneth O Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007.

[146] Kenneth O Stanley, Bobby D Bryant, and Risto Miikkulainen. Evolving adaptive neural networks with and without adaptive synapses. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 4, pages 2557–2564. IEEE, 2003.

[147] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009.

[148] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.

[149] Luc Steels and Paul Vogt. Grounding adaptive language games in robotic agents. In *European Conference on Artificial Life, Cambridge, MA*, 1997.

[150] Vito Trianni and Marco Dorigo. Self-organisation and communication in groups of simulated and physical robots. *Biological cybernetics*, 95(3):213–231, 2006.

[151] Elio Tuci, Christos Ampatzis, Federico Vicentini, and Marco Dorigo. Evolving homogeneous neurocontrollers for a group of heterogeneous robots: Coordinated motion, cooperation, and acoustic communication. *Artificial Life*, 14(2):157–178, 2008.

[152] Elio Tuci and Matt Quinn. Behavioural plasticity in autonomous agents: a comparison between two types of controller. In *Applications of Evolutionary Computing*, pages 661–672. Springer, 2003.

[153] Alan M Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[154] Ryoko Uno, Davide Marocco, Stefano Nolfi, and Takashi Ikegami. Emergence of protosentences in artificial communicating systems. *Autonomous Mental Development, IEEE Transactions on*, 3(2):146–153, 2011.

[155] Karl Von Frisch. The dance language and orientation of bees. 1967.

[156] Kyle Wagner, James A Reggia, Juan Uriagereka, and Gerald S Wilkinson. Progress in the simulation of emergent communication and language. *Adaptive Behavior*, 11(1):37–69, 2003.

[157] Gregory M. Werner and Michael G. Dyer. Evolution of communication in artificial organisms. In Christopher G. Langton, Charles Taylor, Doyne J. Farmer, and Steen Rasmussen, editors, *Proceedings of the Workshop on Artificial Life '90*, pages 659–687. Reading, MA: Addison-Wesley, 1991.

[158] Paul L Williams, Randall D Beer, and Michael Gasser. Evolving referential communication in embodied dynamical agents. In *ALIFE*, pages 702–709, 2008.

[159] Steffen Wischmann, Dario Floreano, and Laurent Keller. Historical contingency affects signaling strategies and competitive abilities in evolving populations of simulated robots. *Proceedings of the National Academy of Sciences*, 109(3):864–868, 2012.

[160] Steffen Wischmann and Frank Pasemann. The emergence of communication by evolving dynamical systems. In *From Animals to Animats 9*, pages 777–788. Springer, 2006.

[161] Brian M Yamauchi and Randall D Beer. Sequential behavior and learning in evolved dynamical neural networks. *Adaptive Behavior*, 2(3):219–246, 1994.

[162] Qizhi Yu, Chongcheng Chen, and Zhigeng Pan. Parallel genetic algorithms on programmable graphics hardware. In *Advances in Natural Computation*, pages 1051–1059. Springer, 2005.

[163] Victor Zykov, Efstathios Mytilinaios, Bryant Adams, and Hod Lipson. Self-reproducing machines. *Nature*, 435(7038):163–164, 2005.

[164] Viktor Zykov, Efstathios Mytilinaios, Mark Desnoyer, and Hod Lipson. Evolved and designed self-reproducing modular robotics. *Robotics, IEEE Transactions on*, 23(2):308–319, 2007.