Abstract

Nelson, Andrew Lincoln. Competitive Relative Performance and Fitness Selection for Evolutionary Robotics. (Under the direction of Dr. Edward Grant.)

Evolutionary Robotics (ER) is a field of research that applies evolutionary computing methods to the automated design and synthesis of behavioral robotics controllers. In the general case, reinforcement learning (RL) using high-level task performance feedback is applied to the evolution of controllers for autonomous mobile robots. This form of RL learning is required for the evolution of complex and non-trivial behaviors because a direct error-feedback signal is generally not available. Only the high-level behavior or task is known, not the complex sensor-motor signal mappings that will generate that behavior. Most work in the field has used evolutionary neural computing methods. Over the course of the preceding decade, ER research has been largely focused on proof-of-concept experiments. Such work has demonstrated both the evolvablility of neural network controllers and the feasibility of implementation of those evolved controllers on real robots. However, these proof-of-concept results leave important questions unanswered. In particular, no ER work to date has shown that it is possible to evolve complex controllers in the general case. The research described in this work addresses issues relevant to the extension of ER to generalized automated behavioral robotics controller synthesis. In particular, we focus on fitness selection function specification. The case is made that current methods of fitness selection represent the primary factor limiting the further development of ER. We formulate a fitness function that accommodates the Bootstrap Problem during early evolution, but that limits human bias in selection later in evolution. In addition, we

apply ER methods to evolve networks that have far more inputs, and are of a much greater complexity than those used in other ER work. We focus on the evolution of robot controllers for the competitive team game *Capture the Flag*. Games are played in a variety of maze environments. The robots use processed video data requiring 150 or more neural network inputs for sensing of their environment. The evolvable artificial neural network (ANN) controllers are of a general variable-size architecture that allows for arbitrary connectivity. Resulting evolved ANN controllers contain on the order of 5000 weights. The evolved controllers are tested in competitions of 240 games against hand-coded knowledge-based controllers. Results show that evolved controllers are competitive with the knowledge-based controllers and can win a modest majority of games in a large tournament in a challenging world configuration.

COMPETITIVE RELATIVE PERFORMANCE AND FITNESS SELECTION FOR EVOLUTIONARY ROBOTICS

By

ANDREW L. NELSON

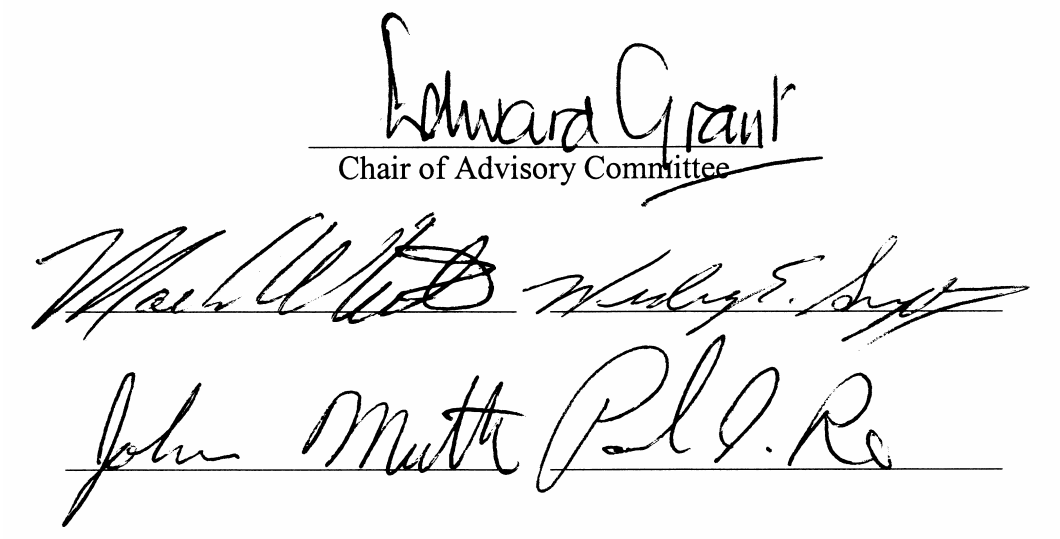
A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the degree of
Doctor of Philosophy

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Raleigh

May 2003

APPROVED BY:



Dedication

To my parents, Dr. David Alan Nelson and Dr. Joan Blackadar Nelson.

Biography

Andrew Lincoln Nelson was born February 28, 1967 in Laramie, Wyoming to Joan Blackadar Nelson and David Alan Nelson. He received his Bachelor of Science from the Evergreen State College, Olympia, Washington in 1991. He received his Master of Science in Electrical and Computer Engineering from North Carolina State University, Raleigh, North Carolina in 2000. He received his Doctor of Philosophy in Electrical and Computer Engineering from North Carolina State University, Raleigh, North Carolina in 2003.

Acknowledgements

I would like to thank my advisor Dr. Edward Grant for giving me the wonderful opportunity to pursue this research and this degree. Also, I would like to thank my committee members, Dr. John Muth, Dr. Wesley Snyder, Dr. Paul Ro, and especially Dr. Mark White for his insightful input and support.

I would like to thank the students at the Center for Robotics and Intelligent Machines (CRIM), especially Greg Barlow, John Galeotti, Leonardo Mattos, Tim Slusser and Kyle Luthy.

In addition, I would like to acknowledge my family members including my parents, brother Randal, and sisters Catherine, Julia, and especially Susan for all her help and encouragement.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1. INTRODUCTION TO DISSERTATION

1.1 Research Goals And Contributions To Knowledge

The goal of this research is to advance the sophistication of evolved embodied machine intelligences. The particular focus of the work is in the field of evolutionary robotics (ER). Evolutionary robotics applies evolutionary computing (EC) methods to evolve populations of controllers for use in robots. In this research, artificial neural networks (ANN) were evolved using reinforcement learning (RL) methods to control autonomous mobile robots. The work documented here represents advancements in the field of ER in several ways.

First, the selection of controllers during the evolutionary process was driven by a relative competitive performance metric. Populations of robot controllers were evolved to play a competitive team game. During the process of performance fitness evaluation, controllers within the population competed against one another to achieve relative fitness scores. The performance metric (fitness selection function, fitness function, or objective function) made use of two mutually exclusive selection modes. The first mode produced fitness values in randomly initialized populations of controllers. This was necessary because initial controller populations generally had no detectable ability to complete the game-playing task as a whole. The second mode of the performance function superceded the first and relied only on aggregate

competitive win/lose information. This mode of the function allowed constraints from the first mode, which incorporated human bias, to be discarded in cases where a controller completed the overall task. Interaction of these modes was automatically self-regulated and pure aggregate win/lose selection approached 100% as populations evolved. In short, the selection metric addresses the Bootstrap Problem in a way that doesn't permanently restrict the course of evolution with human bias.

Second, robot controllers relied on processed video images for sensing of their environment. These required many more network inputs than had been used in previous work. The evolved controllers are presented and analyzed in chapters 5 and 6 used 150 separate sensor inputs. In order to accommodate these, the size and complexity of the evolved neural network controllers were considerably greater than those used in other ER research. A simulation environment was integrated with a real environment with the main coupling point being at the level of the processed video sensors. Controllers were evolved in simulation and transferred to real robots. Both simulated and real controllers received sensor information of exactly the same format. Controllers evolved in this dual environment could be transferred directly from simulated to real robots, and even from real robot to real robot without any further modification.

Third, after the evolutionary process was halted, fully evolved controllers were evaluated in extensive competitive tournaments against hand-coded knowledge-base controllers designed to play the same team game as the evolved controllers. Results

obtained from the tournaments allowed the evolved controller performances to be measured with an absolute metric, even though they had been evolved using a relative competitive metric. An evaluation of the results shows that the best-evolved controllers could out-play the hand-coded controller over a series of many games. Fully evolved controllers were also evaluated in competition with other evolved controllers and less evolved versions of them selves using similar methods.

1.2 Overview Of Dissertation Chapters

This section contains an overview on the dissertation.

Chapter 2 presents a brief review of the origins of evolutionary robotics (ER). Research in the field is summarized in an extensive literature review. Methods of measuring fitness for selection in the artificial evolution of behavioral robotics controllers are reviewed. The definition and application of these fitness metrics is identified as a significant limiting factor in the field of ER.

In Chapter 3 preliminary research related to the later main focus of this dissertation is summarized. First an early simulation based imitative learning experiment is discussed. Second, work is discussed in which artificial neural network based robot controllers were evolved in simulation to perform a locomotion and object avoidance behavior using simple binary tactile sensors. The evolved controllers were then

transferred to real robots for experimentation. This represented the first work related to this dissertation that was verified using physical robots.

Chapter 4 describes an evolutionary robotics research test-bed that was developed in conjunction with this research. The test-bed consists of a colony of mobile robots, a physical reconfigurable maze environment, a closely coupled simulation environment, an evolutionary neural computing application, and a vision-based sensor system.

Chapter 5 describes the use of artificial evolution to synthesize neural network based controllers for mobile robots engaged in team behaviors. For selection of the fittest controllers during the artificial evolution process, a competitive relative fitness metric is defined. The metric is used to evolve controllers to play a competitive game with teams of autonomous mobile robots. In this research, populations of robot controllers were evolved to play the game *Capture the Flag* in maze environments. Controllers were evolved under different sets of environmental and selection conditions. Results from these evolutions are presented and evolved robot controller behaviors are analyzed qualitatively over the course of evolution. The controllers were further tested on real robots and results from these tests are discussed.

In Chapter 6 an additional metric is derived. This metric deals with the post evolution evaluation of robot controller performance. Evolved game-playing robot controllers were played in competition against a knowledge-based controller. Several extensive

tournaments of 240 games were conducted and results from the tournaments are presented. These results serve to rank the evolved controllers with respect to a controller of well-defined abilities (the knowledge-based controller).

Chapter 7 presents concluding remarks including an overview of results presented in this dissertation and a discussion of current issues related to the field of ER. Several potential future lines of research related to this work are also discussed.

CHAPTER 2. EVOLUTIONARY ROBOTICS AND ARTIFICIAL EVOLUTION

The fundamental goal of evolutionary robotics (ER) is to develop automatic methods of autonomous mobile robot controller synthesis that do not require hand coding or in depth human knowledge of the robot task for which the controller is intended.

As the state of evolutionary robotics technology stands now, human-designed robot controllers can significantly out-perform automatically generated or evolved robot controllers. Nonetheless, human designed controllers are brittle and can fail in situations not considered in the initial design. Also, humans may be incapable of directly designing controllers beyond a certain level of complexity.

In order to produce less brittle robot behavioral controllers, ones that can function in environments not understood by humans, or ones that produce behaviors beyond the complexity that a human can design, automated methods are likely to be required. We must develop technologies that allow us to synthesize controllers that produce desired complex behaviors. One possible way to do this is to automate a search process in the search space of *all possible controllers* for one that behaves in a desired way. This is the focus of evolutionary robotics (ER).

The concepts motivating ER are not new, but it has only been in the last 10 years that any serious attempts have been made to evolve intelligent autonomous robot controllers. This chapter will review relevant research from the literature and discuss some of the most pertinent points raised therein. In particular, current methods for determining controller fitness for the purposes of driving an artificial evolutionary process are identified as limiting factors in the field.

2.1 The Origins and History of Evolutionary Robotics

2.1.1 Roots and Definitions

ER has its roots in behavioral robotics [1], artificial life [2], evolutionary computing [3], and machine learning.

Perhaps the earliest behavioral robotics work was done by Gray Walter in the early 1950's [4]. Walter constructed a three-wheeled autonomous robot, called the tortoise, which displayed simple behavior including homing on light sources and returning to a recharging station when its battery was low. The tortoise was a purely analog device with very simple reactions, yet it gave the appearance of displaying complex behavior. Although, the tortoise's behaviors were in fact limited, they demonstrated by example that very simple rules can lead to complexity when expressed in a system that can act and react within a physical environment.

In the 1980's Braitenberg developed the concept of an autonomous robot controller in terms of direct connections of varying complexity from sensor inputs to actuator (drive motor) outputs. Braitenberg produced a classification of autonomous mobile robots based on the degree of complexity related to the mapping of primary sensor inputs to final actuator outputs [5]. This classification, though, was mainly focused on simple reactive controllers.

Modern behavioral robotics employs a wide variety of conceptual and structural robot control architectures. These include nested hierarchical controller structures [1] and quasi-parallel controller paradigms such as subsumption architecture [6]. Autonomous robot controllers developed using Evolutionary Computing methods are distinct from most other modern control structures in that they are consider to be "model-free".

Controllers that produce actuator outputs as a direct function of sensor inputs, without using an explicit internal environment model are referred to as being "model-free". Almost all robot controllers developed with ER methods fall into this category. It is possible to make the case that no system that interacts with its environment in an intelligent way can be truly model free. Even so, controllers that are constituted entirely of sensor input to actuator output mappings do not need to contain anything approaching an explicit environmental model. If a controller were hand-designed, as was the case in Grey Walter's tortoise, it is fair to say that the human designer made use of his/her own internal complex world model to configure the controller in the

first place. If a similar controller were evolved in an environment based on its own interactions, however, it is fair to say that the resulting controller is indeed model-free. It is possible that a controller could evolve an environmental model, if it were very beneficial to the performance of a particular task. However, this is very unlikely using current ER methods.

2.1.2 The History of Evolutionary Robotics Research

Starting in the early 1990's, advances in computing speeds allowed initial ER experiments to be conducted. Throughout most of the 1990's feasibility studies and proof-of-concept research were performed. In this section, ER research is chronicled. There is a fair amount of experimental research reported in the ER literature (on the order of 400-600 reports total). Much of the literature is made up of reports detailing the same or almost identical experiments presented by different members of related research groups. Here, the earliest examples of particular concepts and experiments are reported on. Later similar work is only briefly noted. In the entire literature, there are only 10 to 15 distinct experimental results of real interest. The entire remainder of the literature is comprised of either repeated reporting of the same results and/or slight variations on procedural methods. In this section only, dates are included in the text along with the reference citations. This was done to give a sense of the relative timing of important ER developments.

Early experimental research into ER (1992) is attributed to Koza [7]. In that work, genetic programming (GP) was used to evolve robot control programs for a wall-following task. This work might actually fall better under the heading of artificial life

(AL) since only simulated robot agents were used and no attempt was made to apply the work to real robots. Early works by Brooks [6][8] are often sighted in the ER literature. However, Brooks' ER work was almost entirely speculative and philosophical and included very little simulated or real-world evolutionally robotics research. In [9] Brooks does report on a GP based robot controller evolution application similar to that used earlier by Koza [7].

Navigation with obstacle avoidance behaviors using wheeled robots were studied in early ER work and continue to this day to be used as benchmark tests of ER methods. In [10] (1994) Nolfi et. al. report on the fully embodied evolution of navigation and obstacle avoidance in a real two-wheeled robot using 8 infra-red (IR) sensors and small recurrent neural networks. Their evolutionary process required 100 generation and about 60 hours to produce successful behaviors. In [11] (1995) Jakobi et. al. report on the evolution of obstacle avoidance and of phototaxis behaviors in a simulated two-wheeled robot using real sensor data. Similar later work can be found in [12][13][11][14].

Locomotion and obstacle avoidance behaviors in legged robots have been the subject of several ER studies [15][16][17]. An early embodied ER experiment (1994) is reported on in [18]. There, neural networks were evolved to produce oscillating leg gaits for a physical salamander-like animate. That work, however, did not include any sensor stimulus into the evolved system. In [15] (1994) Gruau reports on a cellular encoding scheme for evolvable modular neural networks for legged robot

control. Filliat et. al. [16] (1999) were able to evolve efficient locomotion and object avoidance behaviors. In their research, networks of threshold neurons were evolved to produce appropriate output signals to the leg actuators of a hexapod robot in response to IR sensors mounted on the robot. In this case evolution was carried out on populations of ANN's encoded with a descriptive encoding syntax similar to Gruau's cellular encoding [15]. Controllers were evolved in simulation and transferred to real robots for testing. Jakobi et. al. [17] (1998) described the use of minimal simulation to evolve behaviors in an eight legged robot with sixteen actuator motors. An ANN controller structure was used. Similarly to Filliat et. al. [16], walking and obstacle avoidance behaviors were evolved in simulation and then tested on a real robot.

Homing behaviors such as phototaxis and chemotaxis constitute an additional class of commonly used benchmark behaviors in ER. In [20] (1993) Cliff described the evolution of neural network controllers for a simulated two wheeled robot that used simulated photo detectors and tactile sensors. This simulated robot was evolved to travel to, and remain at the center of its world. In [21] (1994) Nolfi et. al. describe a simulation environment in which very simple cellular agents learn to find food objects. However, the cellular simulated word was quite simplified and this work might also fall more under the heading of artificial life (AL) than ER. Floreano et. al. report on a somewhat more complex behavior in [22] (1995) in which robots evolved a repetitive homing behavior that simulated returning to a battery recharging station in response to low energy levels. That work was carried out completely using

embodied evolution and required approximately ten days to complete 100 generations. Similar work was reported in [23]. In [24][25] Kodjabachian et. al. (1998) describe the incremental evolution of walking, object avoidance and chemotaxis in a simulated six-legged insectoid robot. In [26] Hornby et. al. (2000) describe the evolution of ball chasing using an 18-DOF quadruped robot.

Competitive evolution in which the fitness of one individual may affect the fitness evaluation of another is a central theme of the research described in this dissertation. In the literature, several examples of competitive evolution exist in the form of co-competitive evolution. Cliff and Miller (1995) investigated the co-evolution of competing populations in the form of predator-pray behaviors [27][28]. Those works mark the first use of co-competitive selection in ER in which the fitnesses of the agents were affected by the fitnesses of other evolving agents. Other similar later works have been reported on in [29][30][31][32]. Direct competitive evolution of controllers with in a single population is investigated in research related to this dissertation [33][34] and will be described in detail in chapters 5 an 6.

There has been a small amount of research into the co-evolution of robot bodies and controllers [35][36][86]. For example, in [35] (1999), both sensor configuration and neural controller structures were co-evolved for navigation and obstacle avoidance in two-wheeled robots. Jordan et. al. describe experiments involving evolution in simulation followed by the physical construction of real robots bodies made from triangular trusses [36] (1999). Such work is quite limited because it is extremely

difficult using current technologies (even in simulation only) to set up a system that can readily benefit from the evolution of physical and controller structures simultaneously.

In the last few years, somewhat more complex tasks have been investigated. It should be noted that many of these behaviors are only marginally more complicated than behaviors achieved in the earliest days of ER.

Peg pushing behaviors were evolved in [37] (1999) [38] (2000). In each of those works, the task required two-wheeled robots to push a peg (small cylinder) toward a light source. Earlier in, [39] Lee et. al. (1997) investigated a similar box-pushing behavior using GP.

The most difficult behaviors addressed in the literature involve some form of sequential action. Nolfi reports on the evolution of a garbage collection behavior in which a robot must pick up pegs in an arena and deposit them outside the arena [40](1997). In [41] Ziemke (1999) studied the evolution of robot controllers for a task in which a robot must collide with objects (collect) in one zone and avoid them in another. In [42] (2000), Floreano et. al. report on the evolution of a robot behavior in which robots move to a light and then back to a home zone. Another example of evolution more complex behaviors is reported in Tuci et. al. in [43](2002). There, robot controllers evolve to produce life-time learning in order to predict the location of a goal object based on the position of a light source.

Recently, flocking or group movement behaviors have been investigated. Ashiru describe a simple robot flocking behavior in [44] (1998). A two-robot coordination task in which two robots evolve to move while maintaining mutual proximity is reported by Quinn in [45] (2000). Baldassarre et. al. [46] (2002) evolved homogeneous controllers for a task in which 4 robots must move together in a small group toward a light or sound source.

Table 2.1. Summery of Landmark ER Research Results.

Date	Author and reference	Evolved Behavior
1992	Koza [7]	Simulated wall following
1993	Cliff et. al. [20]	Simulated zone homing
1994	Nolfi et. al. [10]	embodied evolution of navigation and object avoidance
1996	Cliff and Miller [27]	Co-evolution of predator and prey
1997	Nolfi [40]	Peg collection and deposition
1998	Jakobi [17]	Octopod locomotion
2000	Quinn [45]	Multi-robot Coordinated movement task
2000	Floreano [42]	Sequential zone homing
2001	Smith et. al. [47]	Differentiate and home in on shapes
2002	Tuci et. al. [43]	Life-time learning task involving using light source to locate goal

Table 2.1 contains a list of important results in the development of ER. Conceptually speaking, these results are incremental improvements. None of them is really representative of advances due to the discovery of fundamentally new methods, or the development of new basic theory. As will be discussed in the next section, the field of ER has become somewhat stagnant.

2.2 The State of the Art of ER

This and the next section discuss current trends and issues in ER. The field of ER has been reviewed in several publications [19][48][29][49]. Pertinent issues were raised in those works and include 1) embodied evolution in real robots vs. evolution in simulation, 2) the coupling of simulation to reality, 3) controller architecture, 4) the application of ER methods to more sophisticated and general behaviors, and 5) sensor complexity and configuration.

In addition to these issues, in Section 2.3 we will focus on methods of performance evaluation. We make the case that fitness evaluation during evolution is one of the major factors affecting the issue of generalization of ER methods to produce more sophisticated robot behavioral controllers.

2.2.1 Simulation or Embodiment

The question of whether simulation, embodied evolution or a combination of both is the best approach to advance ER has received attention in the ER literature [11][50][51]. The question has not yet been fully resolved. In [50] the authors made the case for the continued use of embodied evolution to address issues related to the extensibility of ER methods. In the research reported in this dissertation a somewhat different view is taken: the argument presented here is that the fundamental difficulty facing ER is the formulation of training conditions (whether simulated or real) that are capable of eventually evolving complex behavior in a truly automated fashion. The issues of simulation fidelity can be dealt with to a large degree by careful

implementation and by paying close attention to the interface between simulation and reality. Several issues related to difficulties of transference from simulation to reality were investigated in [11] [87] and found to be amenable to various remedies. Also, using robot interaction with the real world as the only form of performance evaluation is too slow and expensive: if simulation is competitive with reality now, in a few years increasing computer speeds will obviate the need for fully embodied evolution, at least for the types of behaviors within the grasp of current ER methods. It is not argued that embodied evolution will never yield results of value: massively parallel embodied evolutions involving hundreds or thousands of robots with advanced computing capabilities may extend the domain of ER. These types of experiments are currently out of reach. On the other hand, embodied tuning of evolved controllers may always be required for some types of behavioral evolutions.

Some ER work has been focused on making simulation methods simpler and faster. For example, minimal simulations have been investigated in [17]. Minimal simulation methods involve the corruption of all simulation environment elements with stochastic noise except those elements deemed to be important to the development of a particular behavior. One criticism of minimal simulation is that it requires humans to have sufficient knowledge of the characteristics of a desired behavior to select environmental which features are important to it.

2.2.2 Transference from Simulation to Reality

As noted above, there has been considerable research devoted to transference of controllers evolved in simulation to real robots. Sampling the physical system to

obtain a set of realistic sensor and robot-environment interactions has been used to generate high quality simulators [11]. Injection of appropriate levels of noise into simulation environments has also been shown to aid in transference [11]. In [52] a gantry apparatus that supplied real sensor data but allowed rapid repositioning of the sensor view is described. That method was intended to limit sensor transference issues while still allowing rapid evaluation of controllers. These methods have been shown to improve transference. There are however, a fair number of examples of controllers evolved in simulation that were effectively directly transferred to real robots [40][29][16][23].

2.2.3 Controller Architecture

Selection of controller structure has also been an issue. The majority of current ER work uses neural network based controllers. Much of the work, though, uses very simple network topologies and restricted weight values [53][22][40]. Such restrictions limit the scalability of the methods studied. Although neural networks are the dominant form of ER controller structure, the utility of other controller types has not been fully explored. Genetic Programming (GP) has been used to develop controllers [54][55] but it is argued that GP syntactic constructs restrict the controller search space to such a degree that the evolution of ideal controllers for complex tasks becomes intractable. As is the case with neural networks, successful evolution of GP based controllers that perform simple tasks yields little information about the scalability of such methods. Evolvable hardware has also been considered for use in ER controllers [29]. However, it is not clear whether the constraints of evolvable

hardware systems limit their utility for ER methods. Only a small proportion of ER research has used such systems.

2.2.4 Extension of ER to Complex General Behaviors

Possibly the most important unanswered question looming over the field of ER is that of whether the methods used to obtain the simple proof-of-concept results to date can be extended and generalized to produce more sophisticated behavioral control. In turn, a key issue related to the successful evolution of complex behaviors is the specification of a training fitness function or objective function.

The most complex fully evolved behaviors to date include no more than three or four coordinated fundamental sub-behaviors. Examples were reported in Section 2.1 above [40][43][42][44]. In each of these, the fitness functions were fairly complex, and relatively selective for a pre-defined solution.

The main value of the proof-of-concept ER work to date has been that it has shown that neural controller structures can be configured to produce complex and functional behaviors in robots. Said another way, neural networks can be evolved into self-regulating sensory/motor close loop systems for use as autonomous mobile robot controllers. What has not been shown is that ER methods can be generalized to complex behaviors. In particular, no ER work to date has shown that it is possible to evolve arbitrarily complex controllers for the general case.

In 1992, Brooks suggested that evolved controllers would need to be one to two orders of magnitude more complex (than automatically generated programs of the time) to compete with hand coding techniques [9]. In the intervening decade, this has in no way been achieved. Many of the initial concerns and criticisms of the field of ER that were related to embodiment [8] and transference from simulation to reality [26] have been addressed. However, very little fundamental progress has been made in ER in the last 5 years. The problems do not lie in representation, transference or embodiment. Rather, they lie in the most basic elements of the process of automatic development of intelligence. As will be discussed in Section 2.3 these problems largely stem from the fundamental problem of arbitrary controller fitness measurement during selection and replication over the course of evolution. Currently, there are no truly generalized methods of fitness selection that could be applied to evolve arbitrarily complex behaviors. Concerns related to fitness selection remain largely unresolved. The development of methods for general fitness selection during evolution of controllers is crucial. This view is reflected in some recent ER literature [56] and had been pointed out earlier in [22].

2.2.4 Sensor Complexity

An additional issue related to the further development of ER is that of sensor complexity. Almost all ER work to date has involved the use of IR sensors, photo detectors and sonar. These generally constitute less than 10 total real valued inputs to the robot controllers. The sophistication of sensory systems in ER has remained relatively constant since the mid 1990's. A very few research groups have reported on the use of video sensing, but in almost every case, information from the video

images has been reduced to only a few input values, usually less than 10. Really, these must be considered glorified photo-detector sensor systems. For example, in [57] Nolfi et. al. (2002) evolved a two-wheeled robot behavior that used a linear vision system made up of 8 photoreceptors to home in on the larger of two rectangular shapes. One example in which a more sophisticated vision system was use might be the work described in [58]. There, video images were reduced to a 5 by 5 grid of summed values, but these were fed into a preprocessing network and only 2 sensor inputs were passed on to the neural network controller. One other instance of the use of video sensing in ER is the work of [59]. In that case, images were processed to detect only a single object (a yellow ball) using knowledge-based methods. Two inputs, the angle and "size" of the object, were then given to the evolvable portions of the controllers. Additional such work is discussed in [88].

Robot controllers receiving much more extensive senor information may produce qualitatively different and more advanced behaviors. A concern related to number and type of sensors has been raised in some research. It has been assumed that systems of large size, and with many sensor inputs are difficult to evolve because of their high dimensionality. This is often referred to as the "curse of dimensionality". However, and perhaps surprisingly, evolutionary RL methods are not as susceptible to high-dimensionality problems related to input dimension as are error back-propagation training methods. It may be the case that evolved networks learn to use what they need, and ignore extraneous sensor data and internal complexity. In the research reported on in this dissertation, issues related to sensor complexity have been

address by supplying large complex evolving neural controllers with 150 or more processed video sensor inputs. This is an order of magnitude more than has been studied in other work.

2.3 An Overview of Fitness Functions for Selection in ER

Artificial evolution and evolutionary computing (EC) methods require the formulation of functions to be used for measuring fitness of individuals (or solutions) in evolving populations. A fitness function, (sometimes called an objective function) allows the individuals in a population to be ranked from fittest to least fit. Ranking is integral to the artificial evolution processes used in ER for selection and propagation of populations of evolving controllers. In this section, evolutionary robotics is addressed in the context of fitness selection. A summery of fitness functions used in current and past ER research is presented. These are categorized into several broad classes. In addition, the pros and cons of methods of selection are addressed in a comparative fashion. We especially focus on aspects of fitness selection that have ramifications for the application of ER methods to more complex problems.

The process of controller evolution in ER is ideally one of primary generation rather than optimization. In most cases the goal of an ER application is to evolve an autonomous robot controller that will produce a desired complex behavior or be able to complete a desired task. The robot controller must receive sensor signals and generate actuator commands over many iterative cycles to produce the behavior. At

each iteration, the controller processes sensor data, and alters its relationship to its environment by producing actuator commands that move the robot. This movement in turn alters the incoming sensor data. This process can be thought of as a complex sensor-controller-actuator-world loop that is mediated by the controller in such a fashion as to generate the desired behavior. For all but the simplest of robot behaviors, the dynamics of the related sensor-controller-actuator loop are not known. If they were known, there would be little point in resorting to evolutionary computing methods to derive controllers as they could be formulated directly from such information. More importantly, the actuator signals that will result in the good expression of a complex behavior are generally not known. Hence, it is not possible to formulate an error back-propagation training scheme to train controller structures. Said another way, a sensor-actuator training data set is not available in the behavioral robotics case. Populations of behavioral robotics controllers must be evolved using selection based on the *expression* of behavior, rather than by the optimization of a known input-output mapping.

Although the issue of fitness function specification is absolutely fundamental to the application of ER to complex problems, it has been largely overlooked or glossed over by researchers up to this point. Two important conflicting factors arise during the process of fitness function specification. These are 1) the need to select for fitness in initial populations that have no measurable ability to complete an overall (complex) task, and 2) the desire to produce fitness measures with a minimum of human bias. If individuals in a population have no detectable level of fitness, then

they cannot be ranked in order from fittest to least fit. In such a case, selection is no better than random and artificial evolution cannot proceed. This is commonly referred to as the Bootstrap Problem. The second factor, the need to limit the injection of human bias into evolved solutions, is a requirement (rather than a desire) if ER methods are to be extended to general non-trivial problems in which humans have incomplete domain knowledge. Most ER work to date has neglected the second factor in order to address the first. Indeed, for very simple behaviors, on which much of the early ER work was based, this issue did not arise. This is because initial controller populations generally do have some detectable and differentiatable fitness with regard to the performance of very simple tasks. Unfortunately, this is rarely the case for complex and non-trivial tasks.

There has been some work focused on developing taxonomies of fitness functions [56] [60]. In [56] the authors propose a "fitness space" to help define and compare fitness functions: this was presented as a continuum between functional and behavioral elements. In [60] the authors present a methodological framework for reinforcement learning fitness function design. In that work, the method was applied to a very simple corridor following behavior using a tank-like mobile robot. These frameworks however, do not aid in the formulation of generalized fitness functions for arbitrary complex behavioral robotics tasks. This is especially true for cases in which the human designer has limited knowledge of the behavioral dynamics related to the robot task in question.

In the following several sections (2.3.1-2.3.4) a high-level general classification of fitness functions used for selection in ER is presented. This is done to give a general context for the presentation and discussion of the bimodal fitness function developed and applied in the research presented in the following chapters. The categories considered here are fairly standard within the research field. We consider four main classes of fitness functions. These are 1) functional (complex functional), 2) incremental, 3) aggregate (success/failure), and 4) competitive.

2.3.1 Functional Fitness Functions

Functional fitness functions (or complex functional fitness functions) can include terms that measure simple response-behaviors, as well as sensor-actuator responses and any other factors the human designer may choose to include to improve selective power. In this context, a simple response-behavior is one in which a predefined action is taken in response to a particular known sensor input pattern. For example, a simple object avoidance response-behavior might be, *if* the value of a particular sensor drops below a predefined threshold, *then* generate a turn-left drive motor actuator command. It was noted earlier that for most complex autonomous robot behaviors, the optimal sensor-actuator relationships that will generate a given behavior are unknown. In the very simple case though, these relationships might be known, or could be handcrafted by a human designer. For many of the proof-of-concept ER experiments reported in the literature, sensor-actuator relationships are partially selected for and appear as terms in functional fitness functions.

Functional fitness functions are almost always formulated by trial and error and/or based on the human designer's expertise (and often a combination of both).

Much of the ER research reported to date has used functional fitness functions to investigated the evolution of extremely simple behaviors. These include phototaxis [52][50] and navigation with object avoidance [24][22][13]. With difficulty, and with sufficient knowledge of the dynamics of a behavior, functional fitness functions can be extended to evolve controllers for somewhat more difficult robot tasks. For example, in [61] the authors describe the evolution of a coordinated movement task involving several robots. Other examples of relatively complex behaviors evolved using complex hand-formulated fitness functions include [40] and [41]. Respectively, these report the evolution of an object collection and deposition task (garbage collection) and a task in which a robot must collide with objects (collect) in one zone and avoid objects in another zone.

Additional examples of the formulation and implementation of functional fitness functions are found in [17][62][63][64].

2.3.2 Incremental Fitness Functions

Incremental fitness functions overcome the problem of sub-minimally competent initial populations (the Bootstrap Problem) by augmenting the difficulty of the task during evolution. Often this is a process of explicit training of simple sub-behaviors followed by more complex behaviors. The main criticism of the use of incremental fitness functions is that they restrict the course of evolution to the degree that

resulting controllers cannot be considered to have evolved truly novel behaviors. They represent the optimization of hand-designed solutions. Even so, several of the most complex evolved robot behaviors reported on in the ER literature were developed using incremental fitness selection functions. In [65] the authors report on the evolution of a prey capture behavior using incremental evolution. In that work, the researchers compare their incremental fitness selection to pure aggregate success/failure selection (using a predefined competent opponent) and report that only the incremental approach is able to produce fit controllers. Further examples of the use of incremental fitness functions in ER include [39][38] and [24].

One form of incremental evolution involves augmenting the difficulty of the environments in which the robots must operate while using a single aggregate success/failure fitness function. This is referred to as "environmental-incremental" evolution. This form of incremental evolution may not constrain the controllers search space to the degree that evolution must converge on a particular predefined solution. Very little work has been done using purely environmental-incremental evolution. In [66], the authors use this type of selection to evolve controllers for a peg collection task similar to the garbage collection task in [41]. That research shows that environmental-incremental evolution using a pure aggregate success/failure selection function can produce functional controllers. However, it is not clear to what degree the selection and augmentation of training environments shaped the final evolved controller population. Other examples include [67] and [64].

2.3.3 Aggregate Fitness Selection

Purely aggregate fitness functions select for high-level success or failure of the complete behavior that the controllers are being evolved to perform. This type of selection reduces injection of human bias to a minimum by aggregating the evaluation of benefit (or deficit) of all sub-behaviors into a single binary value. This is often called "all-in-one" evaluation. Aggregate fitness selection had been largely dismissed by the ER community because in many instances initial populations of controllers have no detectable level of overall competence. Even so, aggregate fitness selection in one form or another appears to be the only fitness selection method that can be applied to generate complex controllers in the general case without injecting restrictive levels of human or designer bias into the resulting evolved controllers. A rare example of aggregate fitness selection applied to the evolution of a complex task in ER is found in [30]. For truly complex behaviors, functional fitness selection, and incremental fitness selection result only in the optimization of human designed controller strategies. They are not examples of the primary evolution of intelligent behavior. At first glance, this appears to present a rather bleak outlook for the future of ER. And in fact, ER has progressed very little in the past five years. However, it is possible in many cases to overcome some of the problems associated with aggregate selection. One such method involves utilizing competitive evolution to present a continually increasing task difficulty to an evolving population of controllers.

2.3.4 Competitive and Co-competitive fitness selection

In this work we promote the hypothesis that competitive fitness selection methods have the potential to overcome current limitations in ER related to fitness selection. Competitive fitness selection utilizes direct competition between members of an evolving population. Robot controllers compete against one another so that the behavior of one robot directly affects the fitness evaluation of another. This concept has received a limited but growing amount of attention. Several examples of co-competitive evolution involving populations of predator and prey robots exist in the literature [30][31]. As noted in those works, two co-evolving populations, if initialized simultaneously, stand a good chance of promoting the evolution of more complex behaviors in one another. The hypothesis is that as one population evolves greater skills, the other responds by evolving reciprocally more competent behaviors. The research presented in [30] and [31] shows this effect only to a modest degree, but there are results from other areas of evolutionary computing that suggest that given the correct evolutionary conditions, pure aggregate selection combined with competitive evolution can result in the evolution of very competent behaviors [68][89]. For example, in [68] neural networks were evolved to play computer checkers at the expert level using pure aggregate win/loss selection in a competing population.

A reciprocating ramping up of competitive environmental difficulty is also seen in competitive selection in a single population. This is referred to as the “Red Queen

Effect" in the literature and indicates a situation in which an evolving population alters its own fitness landscape over the course of evolution.

One of the main goals of the research presented in this dissertation was to investigate the application of aggregate selection and competitive selection to evolve very large neural networks using numerous processed video sensor inputs for behavioral robotics control. In order to address the issue of initial populations having no detectable level of fitness, we also introduce the concept of multi-modal fitness selection. This is discussed in detail in Chapter 5.

2.4 Chapter Summary

This chapter contained an introduction to the field of evolutionary robotics (ER). A review of the ER research literature was presented and important contributions to the field were identified. In addition, challenges and issue relevant to the current state-of-the-art were discussed. The issue of fitness selection was discussed in detail in Section 2.3. In that section, a taxonomy of common fitness selection functions was outlined, and the pros and cons of function types were discussed.

CHAPTER 3. EARLY WORK

This chapter focuses on preliminary work that predated the main body of later research. In particular, a simulated robot navigation behavior generated using imitative learning (behavioral cloning), and a later physical robot behavior using tactical sensors will be discussed. These experiments chronicle the development of an evolutionary robotics research platform. The early configurations of this ER platform and associated experiments are also included here so that their results can be compared to later more advanced work.

In addition, this early work constitutes independent replication of results from the larger field of ER research, to a degree. Even so, these early experiments have novel elements. It is interesting to note that some results generated in the field of ER are not replicated or verified by independent groups. In some cases, a single result, or work from a single research group stands in the literature as the only example of a particular experimental procedure, even though that result may be included in numerous papers. An extreme example of this was noted in the research of one of the most prominent ER research groups in which the same experimental data appeared in one form or another in over 30 published works. This may reflect the nascent nature of the field.

The main results and analysis from later research are detailed in Chapters 5 and 6.

3.1 Imitative Behavior

Imitative learning or behavioral cloning, can be applied to develop autonomous robot control if sufficient examples of a desired behavior are available. For instance, a person might remotely control a robot to perform a task. During the course of the robot's movements, robot sensor inputs and actuator outputs can be recorded into a coupled set of training data. The resulting sensor/actuator data set can then be used to train a controller using error back propagation methods. Such data sets can also be used with evolutionary computing (EC) based training methods.

The task investigated was robot navigation with object avoidance. This work is described in more detail in [69]. Since later planned work was to rely on evolutionary training of neural networks, we used EC in this early work. Rather than using a human to control a robot, a knowledge-based object avoidance controller was constructed to drive a simulated robot agent through a simulated environment. All sensor input data and actuator outputs were recorded over a period of time as the agent moved through its environment. These were used to train a neural network to approximate the sensory motor mapping from the training data set. Note that the rules used to drive the knowledge-based controller were not applied to train the neural network. Only the observed resulting sensor/actuator data were used.

3.1.1 The Simulation Environment

This early work was carried out entirely in simulated environments. The simulation environments consisted of *m* by *m* planar grids in which each grid element was either solid or space. Although the matter arrangement in each environment was discretized, the space itself was continuous and robot agents could exist at any real valued point within the range of the environment.

Each simulated robot agent consisted of a data structure that contained the robot's current position, orientation, sensor input readings, and actuator output values. In addition a controller structure was associated with each robot. In this work, all controllers were time independent mappings from the robot's sensor inputs to the robot's actuator outputs. This can be written in functional form as follows:

$$\mathbf{A}_n = f_{cn}(\mathbf{S}_n) \tag{3.1}$$

where $\mathbf{A}_n$ and $\mathbf{S}_n$ are the sets of actuator values and sensor values of the *n*th robot agent, respectively, and f_{cn} is the controller mapping associated with the *n*th robot agent. Controllers of this type are purely reactive. They base actuator commands only on current sensor readings and integrate no information from the past. In later research, it was found that purely reactive controllers were limited in their abilities, but for this early work, they were adequate for reproducing simple behaviors.

Each robot was simulated with two motor-wheel actuators, one on each of the right and left sides of the robot. Such a configuration allows a robot agent to turn in any

direction or move along any diameter arc by varying the inputs to the wheel motors. This arrangement is commonly referred to as differential steering.

Laser range finding sensors were simulated so that each sensor was associated with a fixed orientation with respect to the robot's body-attached frame of reference. Each sensor returned a scalar value that corresponded to the linear distance between the sensor and the nearest solid element directly in line with the sensor's orientation. This produced a whisker-like array of sensors centered along the central axis of each robot body frame. A schematized top view of a robot agent and range sensor array detecting matter in a simulated world is shown in Figure 3.1. The object on the left in Figure 3.1 is the robot agent. The lines represent the magnitude and direction of current sensor readings. The small shaded blocks in the figure represent simulated matter.

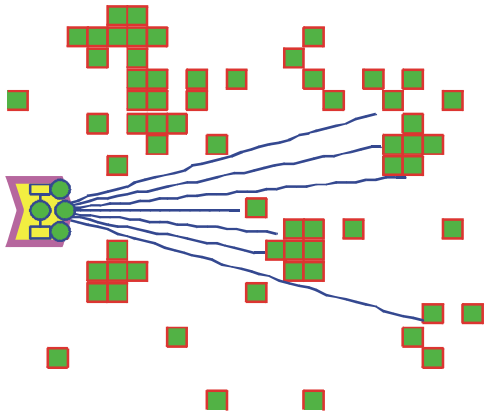


Figure 3.1. A simulated robot agent with sensor and wheel actuators in a matter-containing environment. The object on the left is the robot agent. The lines represent the magnitude and direction of current sensor readings. The small blocks represent simulated matter.

Three planar simulation environments were used for this work: an environment with linear walls that divide the space to form a maze-like structure, an environment filled with aggregates and clusters of matter, and an environment containing only space. These will be referred to as (a) Maze World, (b) Aggregate World and (c) Empty World respectively, and are shown in Figure 3.2.

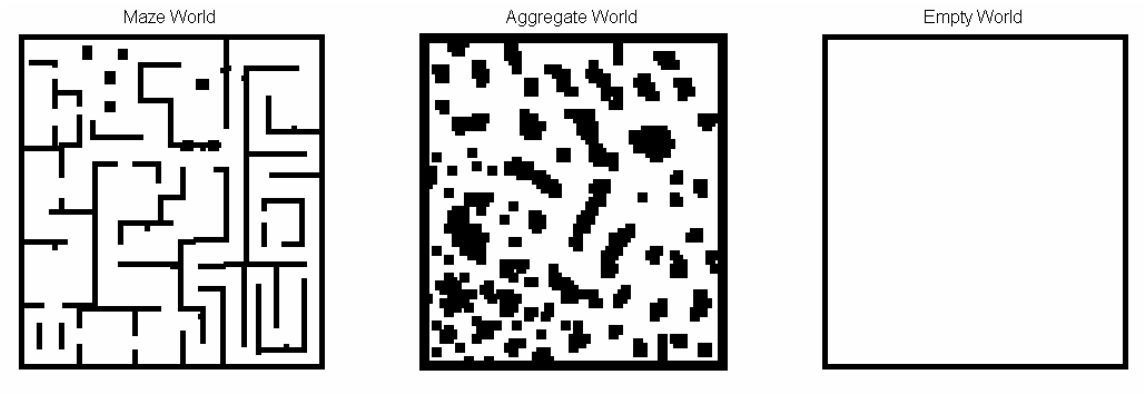


Figure 3.2. Planar simulation worlds

3.1.2 Construction of the Imitative Training Data Set

Robot agents were initially controlled with a knowledge-based controller that produced wheel speed values as a function of range-finding sensor input values. Two simple rules were found to be sufficient to keep robot agents from getting stuck (i.e. becoming immobilized while in contact with material) in most environments. These were formulated as follows:

- Rule 1: Each wheel motor speed was made proportional to the sum of the range-finding sensor readings on the opposite side of the robot.

- Rule 2: If the total sum of all the range finding sensor readings was less than a pre-defined threshold, then the right hand side wheel motor was given a negative speed command.

Rule 1 caused robots to veer away from objects or to remain centered between the walls of corridors. Rule 2 allowed the robots to escape from corners.

The method used to extract behavioral traits observed in the robots agents operating with the knowledge-based controllers was as follows. A simulation world was arbitrarily constructed. Robot agents were initialized to random positions within the simulated world. A simulation was preformed using the simple rule-based controller. During the simulation, all robot agent sensor readings and associated motor output speed commands were recorded. These data were recorded as correlated real-valued input and output vectors. These were in turn used to train an artificial neural network using a supervised evolutionary training. The training data set for the neural controllers discussed here was derived from 50 time steps in Maze World from panel (a) of Figure 3.2.

3.1.3 Neural Controller Formulation and Training

The neural network based controllers were trained using an evolutionary computation based procedure. Similar work has been done using genetic programming (GP) [70]. That work required the careful design of an evolvable robot control syntax. Here, we used standard simple neural networks. Fully connected single hidden layer

feedforward networks were used. Training neural networks of this type involves manipulating scalar weighting functions that operate on the inputs and outputs of the individual neurons in the network. The genetic algorithm used here operates directly on the neural network's set of weights; hence, the chromosome data structure is composed of a set of real-valued scalar numbers.

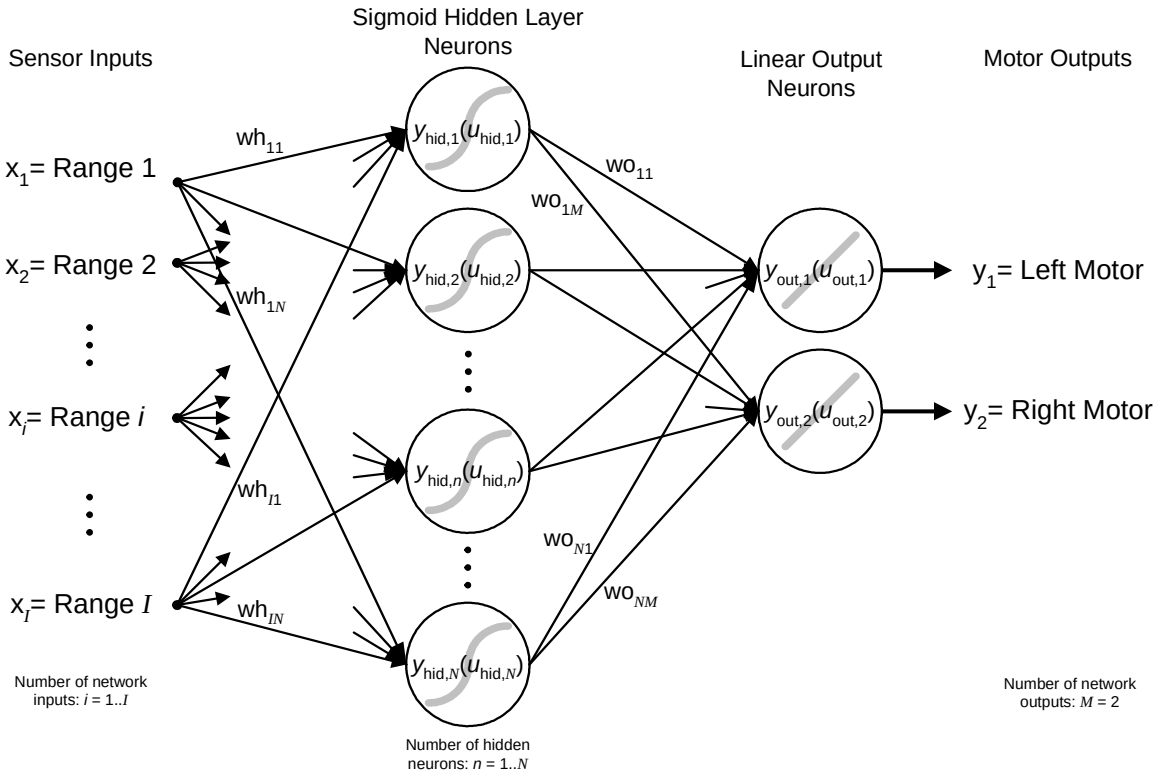


Figure 3.3. The robot controller feed-forward single hidden layer neural network architecture. Range sensor data are fed into the network input connections on the right and drive motor speed commands are generated at the outputs on the left

A typical network structure of the type used in this early work is shown in Figure 3.3. This network is a standard single hidden layer sigmoidal perceptron. The robot agent's current sensor readings, x_1, x_2… x_I are scaled by the hidden layer neuron

weights, wh_{in}, summed and fed into each hidden neuron activation function, $y_{hid}(u_{hid})$. The summations of weighted inputs are given by equation (3.2)

$$u_{\text{hid},n} = \sum_{i=1}^{I} (x_i * wh_{in}) \tag{3.2}$$

Similarly, outputs from the hidden layer neurons are scaled by the output neuron weights, wo_{im}, summed, and fed into the output neurons.

The hidden layer is composed of sigmoid neurons while the output layer is made up of linear neurons. The activation functions for each of these neuron types are given in equations (3.3) and (3.4) respectively.

$$y_{sig}(u) = \frac{1}{1+e^{-u}} \tag{3.3}$$

$$y_{lin}(u) = u \tag{3.4}$$

There can be an arbitrary number of hidden layer neurons; however, each output neuron produces a single actuator value, so the number of output neurons matches the number of actuators in each robot agent.

The genetic training algorithm used was a population of one, mutation based, greedy gradient decent algorithm similar to that described in [71]. Instead of using bit strings to represent candidate solutions, sets of real numbers representing the weights of the neural networks were used. Using the notation of Figure 3.3, a chromosome structure is given by:

$$\mathbf{c} = [wh_{11}, ..., wh_{1N}, ..., wh_{in}, ..., wh_{IN}, ..., wo_{11}, ..., wo_{1M}, ..., wo_{nm}, ..., wo_{NM}] \tag{3.5}$$

At the beginning of the training, all weights were initialized with small random numbers. At each iteration of the training algorithm, perturbations (mutations) were randomly made to one or more of the elements of **c** to make a new chromosome, **c**'. The altered neural net was then tested with the weights specified by **c**' and its performance over the set of training data was compared to that obtained with the weights specified by **c**. If the performance was improved, the mutations in **c**' were kept; otherwise the original **c** was retained and the process was repeated. Performance was measured using the mean squared error of output commands calculated with respect to the training data set recorded from the behavior of the controller driven animat. This is given by E, as:

$$E_j = \frac{\sum_{m=1}^{M} (y_m(\mathbf{x}_j) - y_{tm}(\mathbf{x}_j))^2}{M} \tag{3.6}$$

where $y_m(\mathbf{x}_j)$ is the output produced by the mth output neuron when it is presented with the jth set of training inputs, $\mathbf{x}_j$, and M is the total number of outputs. Further, y_m is the actual output of the mth output neuron, while y_{tm} is the desired, or training output of the mth output neuron associated with the jth set of training inputs.

During training, mutation occurred after presentation of each example in the training set rather that after presentation of the full training set. This allowed the solution (weight set chromosome) the possibility of stepping out of a local minimum with respect to one training example with the occurrence of mutation that reduces error for another training example.

The probability of mutation of each element in the chromosome at each generation is dependant on the current quality of the solution (here, a solution refers to a neural net and associated weight set as specified by the current chromosome **c**). Early in training when solutions are very poor, the mutation probability will be high. As training continues, and the solutions become more refined, the mutation probability will decrease. The following formula was used to calculate the probability of mutation the elements of **c**:

$$\text{MutateProb} = \text{BaseMutateProb} + HB * \frac{E_{current}}{E_{base}} \tag{3.7}$$

$E_{current}$ is the present training error, E, as calculated by equation (3.6). In this work, E_{base} was set to be the training error calculated at the first iteration of training. HB is a scaling factor used to regulate the degree to which the mutation probability was affected by training error. Values used for HB in this work were generally close to unity. BaseMutateProb, the minimum mutation probability, was set to be one mutation out of all the weights in **c**, on average. The effect of (3.7) is that early in the training, while the error is high, the mutation probability is near one so that most elements of **c** are mutated at each training iteration. As E decreases, fewer and fewer mutations occur during each training iteration. When E is very small, the BaseMutateProb term dominates and only 1 mutation occurs at each training iteration, on average.

Making the mutation probability dynamic and related to the quality of the current solution is thought to allow initial, poorer solutions to move more quickly through the solution space. As the solution improves, smaller steps in the solution space are

taken. A low mutation rate is desirable near the end of training as the solution is fine-tuned.

The magnitude of mutation was random, and normally distributed. The center of the distribution was kept constant for the duration of a particular training. Mutations magnitudes were on average between 1% and 5% of the weight magnitudes at the start of training.

3.1.4 Testing of Evolved Controllers

Evolved controllers were compared to both the original knowledge-based controller, and also to a nominal random controller. This nominal controller produced random wheel speed commands that had no relationship to sensor inputs. The wheel motor rates were normally distributed values around the mean of the robot agent speeds when operating under the knowledge-based controller in the simulation world used to derive the data used to train the neural network controller. The random controller was included in the performance evaluation process to provide something of a base-line minimum performance level.

To drive robot agents with the trained neural controllers in simulation, robots sensor readings were fed into the neural network inputs and the resulting network output values were applied to the wheel motors.

Each of the three controllers was used to control robot agents in each of the three simulation environments (shown in Figure 3.2). The results of these simulations are presented in this section.

In each simulation environment, three simulations were preformed, using ten robot agents in every case. The positions and orientations of the robots were initialized randomly for each environment, but were kept the same for all the controllers used in a particular environment so that the results could be compared. The robot agents do not interact; thus, this is equivalent to ten repetitions of a simulation with a single agent.

The Empty World and the random controller were included as comparative experimental controls.

During each simulation, the velocity, position, and orientation of each of the robot agents was recorded at each time point. The metric used for comparison of performance of the controllers was total distance traveled during a simulation. For a particular environment, robot agents using each of the three controllers were simulated for an equal amount of time. Simulation times were 100 time steps for Empty World and 1000 time steps for Maze World and Aggregate World.

Figure 4 presents data in bar graph form comparing the mean and standard deviation of distances travel by the robot agents during each simulation.

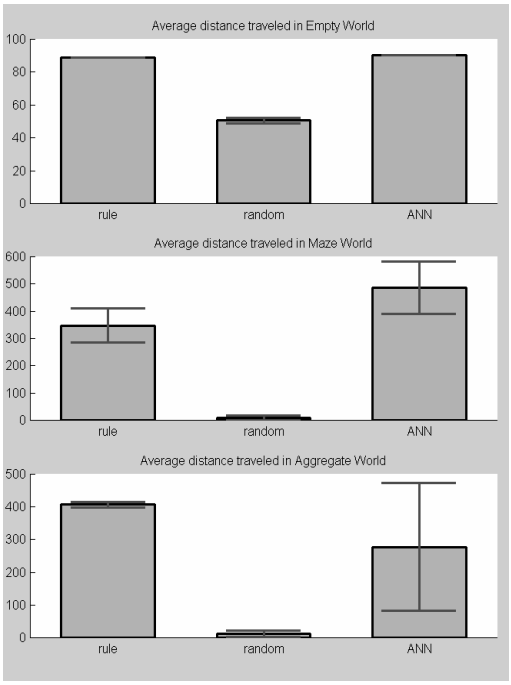


Figure 3.4. Mean distance traveled by robot agents using the rule-bases controller, the random controller and the neural network based controller, in each of the three simulation environments

3.1.5 Discussion of Imitative Controller Evolution Results

The results shown in the first panel of Figure 3.4 indicate that all three of the controllers produced motion of the robot agents in Empty World. Since there was no matter in this environment, the robots could travel without the need to avoid obstacles. The speed of the random controller was set to have a mean value equal to that of the knowledge-based controller operating in Maze World. With the knowledge-based controller, wheel motor speed was proportional to range-finding sensor input values; hence robots being controlled by this controller moved more

quickly in Empty World than they did in either of the other environments because there was no material to detect.

In Maze World and Aggregate World, obstacle avoiding behavior was much more important. The random controller caused the robot agents to become stuck or ensnared very quickly, resulting in very short net travel distances. Both the knowledge-based and the neural controllers, produced successful obstacle avoidance, resulting in longer total travel distances. These results are shown in the second and third panels of Figure 3.4.

The second panel of Figure 3.4 shows that the neural network-based controller performed as well as the knowledge-based controller in Maze World, although the variability in individual robot agent performance was greater. This is reflected in the greater standard deviation of distance traveled. The training set for the neural controller was derived from 50 time steps in Maze World. The simulations above were performed for a period of 1000 time steps. This indicates that, at least in Maze World, the neural controller was able to generalize its performance to many situations not seen by the training set data. In fact, robots using the neural controller were able to operate in Maze World indefinitely without getting stuck on walls or in corners.

In Aggregate World, the neural controller did not perform as well as the original rule-based controller, but it did significantly outperform the random controller. It is likely that the data set used to train the neural controller did not reflect some aspects of the

rule base when applied to Aggregate World, since the training set for the neural controller was derived in Maze World.

3.2 Maze Wandering with Tactile Sensors

In this section, early work involving neural controllers with temporal processing abilities will be discussed [72]. As in the previous section, this work investigated the evolution of controllers for a benchmark basic navigation and object avoidance behavior. Here, though, robots used simple tactile sensors rather than simulated laser range-finding sensors. Very little, if any, other work has been conducted in the field of ER using only tactile sensors. There are several examples of research involving tactile sensors in conjunction with other sensor types [17][29][73]. In these cases, tactile sensors were used in a secondary or back capacity. Simple binary-response (on/off) sensors provide limited information to controllers, thus increasing the difficulty of control. From a research point of view, this provides an opportunity to investigate the evolution of more complex control in a simple system.

In this particular experiment, robot agents relied exclusively on very simple tactile sensors. These give far less information at any one moment than do the range-finding sensors used in the simulation environment discussed in the previous section. Because of this, purely reactive controllers produced sub-optimal behavior. Controllers that make use of temporal information have the potential to outperform

reactive controllers. To accommodate this a temporal neural network controller architecture was introduced into the developing ER research platform.

Populations of controllers were evolved using a functional fitness metric for selection. Robot behavior was evaluated based on performance in simulated environments, and was not the result of an imitative process. A true population-based evolutionary algorithm was implemented. Evolved controllers were transferred to and tested on real robots.

3.2.1 The Temporal Artificial Neural Network Controller Architecture

In this subsection, we describe a temporal neural network architecture that was used in these preliminary experiments. The networks made up a class of multi-layered recurrent and time delayed neural networks. The delayed and recurrent connections imparted the possibility of developing temporal processing. Two example networks are shown in Figure 3.5. These networks can be considered as fully connected generalizations of Elman and Jordan networks [74][75] and include recurrent connections from both hidden layer and output neurons. The layered structure was specified before training and remained constant during the course of training. Only the weights of the networks were evolved. These layered networks represent a large class of network topologies, but they only very sparsely cover the space of all possible networks. However, setting a connection weight in a fully connected network to zero is equivalent to removing that connection, hence the weight-only search space contains all network architecture of lower dimension. In effect, manual

selection of a particular network layered structure and feedback level imposes an upper limit on the complexity of the evolved architectures.

Sigmoid neurons were used in the hidden layers while output layers could consist of either sigmoid or linear neurons. Standard formulas for these were given in equations (3.2) and (3.3).

All the weights associated a particular network layer *m*, time delay level *t*, and recurrence level *s*, were represented and stored in a separate *N* by *I* array. The entire weight set for a given network can be given by the multidimensional matrix

$$\mathbf{W} = [\mathbf{W}_{m,t,s}] \,|\, m \in \{1..M\}, t \in \{1..T\}, s \in \{1..S\} \tag{3.8}$$

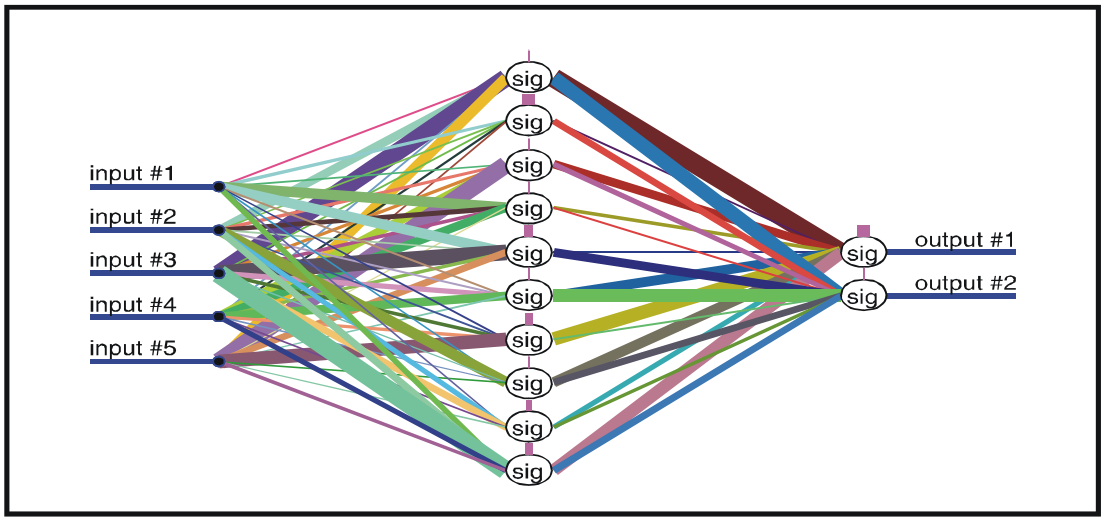
where each sub-matrix of **W** is an *n* by *i* matrix of weights given by

$$\mathbf{W}_{m,t,s} = \begin{bmatrix} \mathbf{w}_1 & \mathbf{w}_2 & \cdots & \mathbf{w}_N \end{bmatrix}^{\mathrm{T}}_{m,t,s} \tag{3.9}$$

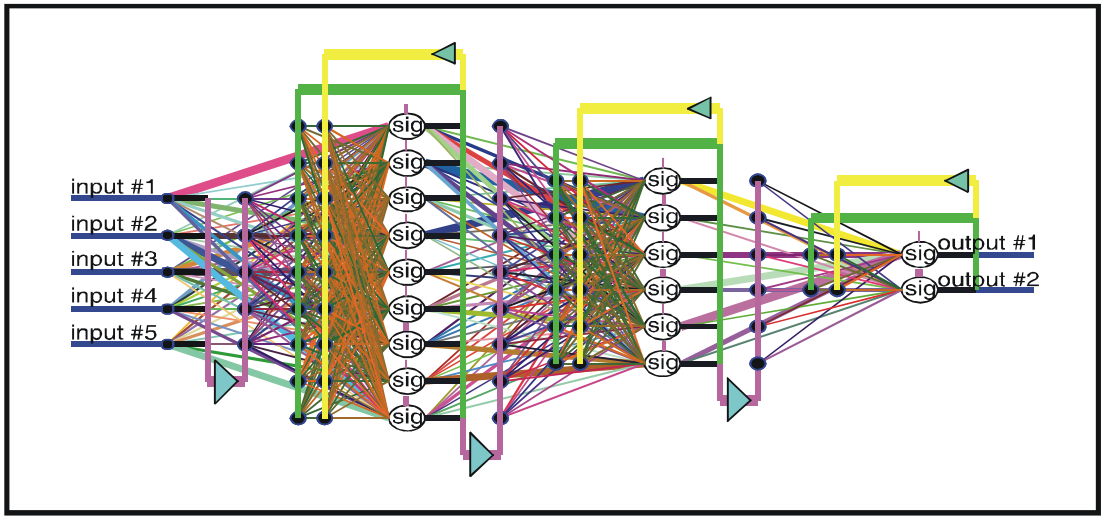
and each column vector is an ordered set of weights associated with a set of inputs subscripted by $i \in \{1..I\}$ to the *n*th neuron ($n \in \{1..N\}$) of the *m*th layer, for the *t*th time delay level, and *s*th recurrence level:

$$\mathbf{w}_n = \begin{bmatrix} w_1 & w_2 & \cdots & w_I \end{bmatrix}. \tag{3.10}$$

M, *T*, and *S*, are the total number of network layers, maximum degree of time delayed connections, and the maximum feedback depth to previous layers of the recurrent connections, respectively.



(a) Network A



(b) Network B

Figure 3.5. Example schematic representations of two neural networks developed by the evolutionary neural computing environment. Network A (a) is a simple feed forward single hidden layer Perceptron. Network B (b) includes two hidden layers and both time-delayed and hidden layer feedback connections.

In Figure 3.5 Network A and Figure 3.5 Network B, the thicknesses of the connection lines are proportional to the absolute values of their particular associated weights.

These network connection graphs are generated by the evolutionary training environment and can be used show changes occurring in the networks during training.

3.2.3 The Evolutionary Training Algorithm

The neural network controllers were trained using evolutionary computing methods in conjunction with the robot simulation environment. The simulation environment was similar to that described in Section 3.1.1 above. Figure 3.6 shows two simulated worlds containing simulated robot agents. The controllers were evolved based on their performance in such simulated worlds.

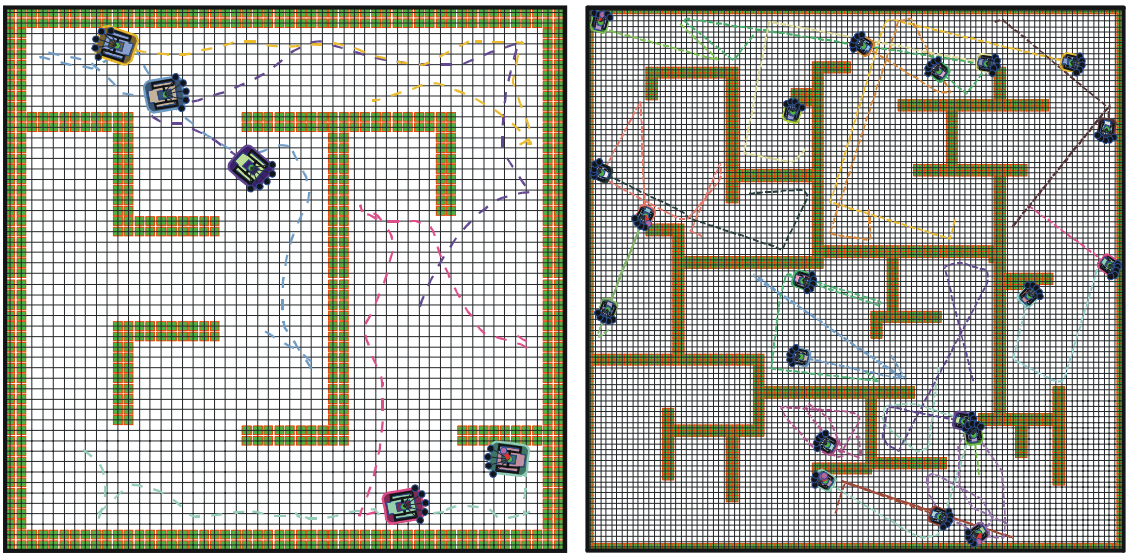


Figure 3.6. Two example simulated maze environments including simulated mobile robot agents with tactile sensors and trained neural controllers. The dotted lines indicate the paths taken by the robots during the course of the simulations.

At the beginning of training connection weights were initialized to small random values from a single random distribution, or from different distributions that

depended on the recurrence and degree of time delay of the particular connection. For the general case, weights were initialized using the following equation:

$$w = R\mu(m)\tau(t)\sigma(s) \tag{3.11}$$

Where $\mu(m)$ is linearly decreasing in m, and $\tau(t)$ and $\sigma(s)$ are monotonically decreasing exponentials with maxima of 1. Here m, t, and s represent the layer depth, the degree of temporal delay, and the degree of spatial feedback respectively associated with a particular connection/weight. R is a number from a flat random distribution in the range (–1, 1). R is re-sampled for each weight initialization. The effect of the weight initialization equation (3.11) is that initial weight values closer to the input layer and with a lower degree of recurrence, have larger magnitudes then those farther into the network, and with a greater degree of recurrence. When $\mu(m) = k$ is constant, and t and s are constant unity functions, all weights are initialized to random numbers in the range ($–k$, $+k$).

The chromosome data structure **C** is a set of real valued scalar numbers where each number corresponds directly to a particular weight w in the neural network weight set **W**, from equations (3.9) and (3.10). An individual chromosome is specified as follows**:**

$$\begin{aligned} \mathbf{C} &= [c_1, c_2, ..., c_g] \\ &= [w_{1,1,1,1,1}, ..., w_{m,t,s,n,i}, ..., w_{M,T,S,N,I}] \end{aligned} \tag{3.12}$$

The rate (probabilistic frequency) of mutation for each weight, w, is dependant on the size of the sub-matrix of **W** to which it belongs, and is given by:

$$Rate = \frac{1}{N * I} Base\_rate \tag{3.13}$$

where N and I are the dimensions of the sub-matrix $\mathbf{W}_{m,t,s}$ to which w belongs and *Base_rate* is a whole number.

Mutation magnitudes, for the weights in an ANN, are scaled in a similar manner to the weight initialization values equation (3.11). Each weight mutation magnitude depends on the location of the weight within the network structure. Hence, for each member of the robot controller population selected to be mutated, the new chromosome elements c' of $\mathbf{C}$ are given by

$$\begin{aligned} c' &= c + \Delta c \\ &= w + \eta R \mathrm{P} \mu'(m) \tau'(t) \sigma'(s) \end{aligned} \tag{3.14}$$

Where $\mathrm{P} \in \{0,1\}$ is determine by the rate of mutation (equation (3.13)), $\mu'(m)$ is linearly decreasing in m, $\tau'(t)$ and $\sigma'(s)$ are monotonically decreasing exponentials with maxima of 1, and η is a base mutation magnitude or step size. Also, R is a number from a flat random distribution between –1 and 1.

During evolution, the next generation population, *P(k)*, is constructed from the union of the following four sets derived from the current population:

$$\begin{aligned} P(k) = \{ & p_1(k-1) .. p_{\mathrm{m}}(k-1) \} \cup \\ & \{ p'_1(k-1) .. p'_{\mathrm{m}}(k-1) \} \cup \\ & \{ p_{2\mathrm{m}+1}(k-1) .. p_{\mathrm{n}-1}(k-1) \} \cup \\ & p_1(k-2) \end{aligned} \tag{3.15}$$

Where $p_{\mathrm{n}} \in P(k)$ is the nth individual of the population at generation k, p'_{n} is a mutated version of p_{n}, m /n is the fraction of the population that is mutated and replaced, and n is the total number of individuals in the population.

The population *P*, is always ordered from fittest to least fit before equation (3.15) is applied. The result of equation (3.15) is that 1/m of the fittest controllers are transferred un-changed to the next generation, this same fraction of the controller population is mutated and added to the next generation, the single fittest member from two generations past is included, and the remainder of the next generation population is made up of the fittest remaining members of the current controller population. The parameters n and m are set at the beginning of each evolutionary run. Selection of values for n and m reflects a trade-off between evolutionary speed and chaos during training. It was found that an m/n = 1/4 value giving a replacement rate of 25% produces functional controllers for population sizes from n = 20 to n = 100. The evolutionary algorithm described in (3.15) is a form of greedy mutation-only ($\mu + \lambda$)-EA [44], with the inclusion of the fittest member of the population 2 generations previous to $P(t)$ (n-elitism).

Performance evaluation at each generation was based on the weighted sum of several factors, including the net offset between a robot's starting position and its final position, (*net_offset*), and whether or not the robot became stuck on material within the simulated environment, (*stuck*). The robots were required to make as much progress through the maze as possible. This was measured by the distance a robot could travel through a maze in a given number of time steps. Implicitly, robots must learn to negotiate walls to maximize their progress through a maze with many walls. A robot that couldn't avoid walls would soon become immobilized when its path was

blocked by a wall. The following functional fitness metric was used to select for navigation behavior in an environment containing walls (a maze).

$$\begin{aligned} F(p_i) = k_1 * total\_curve\_length \\ + k_2 * net\_offset \\ + k_3 * max\_offset + k_4 * stuck \end{aligned} \tag{3.16}$$

where k_1 to k_4 are weighting factors. *total_curve_length* is the line-integral of the full path followed by the robot and *max_offset* is the greatest linear distance obtained by the robot and any time during its travel. The weighting factors were derived empirically through trial and error. The desired behavior is represented by the third factor: maximum offset achieved by the robot from its starting position. It was found, however, that inclusion of two other distance measures, and an explicit penalty for becoming stuck, were required to achieve evolution of navigation behaviors in a reasonable amount of time. It should be noted that in all cases of performance evaluation during evolution, run times were limited so that the best possible performance would result in travel from one side of the environment to the other without time for a return trip. After evolution, resulting robot controllers were allowed to operate for much longer periods of time to demonstrate the dynamics of acquired behaviors. Performance fitness's over several simulation runs were averaged before each generational selection to smooth effects of random robot position initialization.

A number of neural network architectures were found to be evolvable to perform the benchmark navigation and object avoidance task in simulation and to retain ANN controller functionality when transferred to real robots. In the earlier work described

in Section 3.1, it was found that simple single hidden layer feed forward networks could be trained to navigate robot agents in simulation when a model of simulated laser range-finding sensors was used. Those evolved ANN controllers had no capacity for temporal processing. The resulting robot agents were effectively simple Braitenberg vehicles [5] in that they produce motor actuator commands in direct response to current range-finding sensor readings. The level of information provided by range finding sensors was sufficient so that purely reactive controllers could perform the task reasonable well in this environment. Simple binary tactile sensors, on the other hand, required controllers to make use of information from the past in order to overcome perceptual aliasing. For example, a robot would receive all zeros (off) from its set of 5 tactile sensors before it came in contact with a wall, and then again after it had backed away from that wall. Controllers must make use of information from sensor readings from the past in order to distinguish between these two conditions and avoid getting caught in behavioral response loops. That is, a robot must do something different when it is backing away from a wall then when it is approaching a wall, even though it "sees" the same thing (nothing) in both cases. Binary tactical sensors were used here in part to study an evolutionary system that would benefit significantly from the acquisition temporal processing abilities.

It was found that networks of moderate complexity produced the best results in the least amount of simulation time. Such networks had 1-3 hidden layers with 5-10 processing units per layer, with all connections duplicated and time delayed for 2-4 time steps. For example, the best-evolved ANN controller tested in the real robots

and discussed below has 220 evolvable weights (see Figure 3.8). Although our benchmark maze navigation behavior requires some degree of sophistication at the control level, an overly complex neural network structure was not found to be beneficial for this task. It is likely that quite simple specially formulated networks could be trained to accomplish the task studied in this work. We specifically focused on networks of greater complexity to show that larger more complicated network architectures could be readily evolved to perform these behaviors.

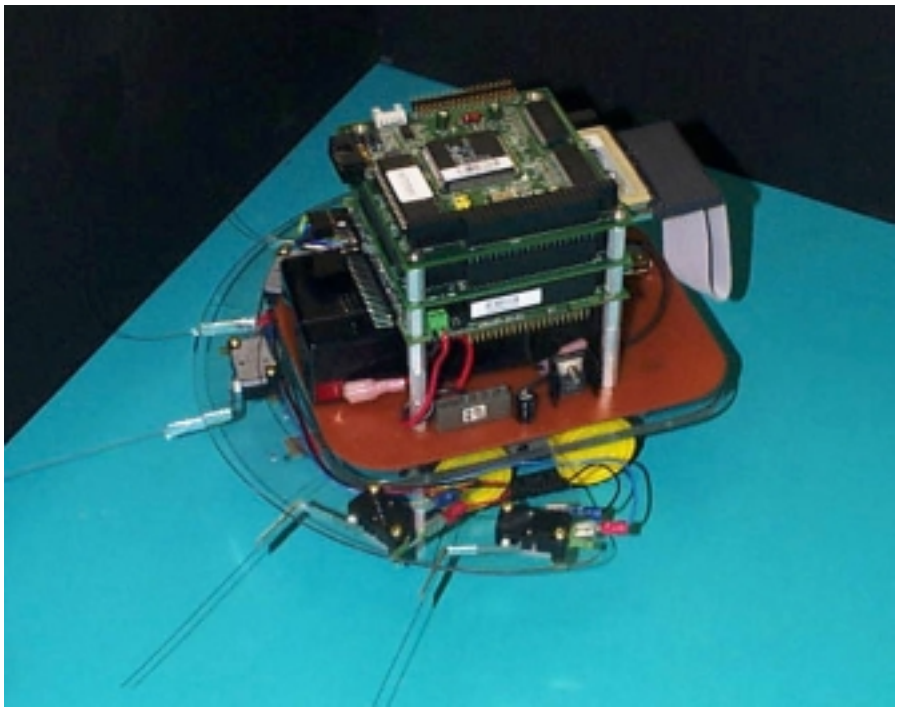


Figure 3.7. An EvBot mobile robot agent fitted with a whisker tactile sensor array. The robot is in contact with a wall and the two left-most tactile sensors are active.

The evolved controllers were transferred to real robots and tested in a real maze test-bed. Figure 3.7 shows a photograph of a real robot fitted with a tactile sensor array.

The architecture of the real robots used in the research, the EvBots [76][77] is discussed briefly in Chapter 4.

The best performing ANN controllers were found to allow the real robots to wander through the real physical maze indefinitely without getting stuck on maze walls, and to allow the robots to make continual progress through the maze, i.e. the robots didn't start spinning perpetually in one spot, or bump up against the same wall over and over again. Demonstration using the real robots was done mainly to show that evolved controllers transferred to the real world and functioned qualitatively similar to their simulated counterparts. The quality of transference from simulation was evaluated in several ways. The responses of the controllers to sensor signals in simulation and in the real robots were compared and found to be identical. This was to be expected because the evolved controllers are identically similar in both cases. This similarity is made possible by the platform architecture, which allows direct transfer of evolved controllers from simulation to real robots without the need for any modification. In addition, the simple binary sensors used provide only logic values of 0 or 1 and inject no noise into either the simulated or real systems. There were however, two differences that caused divergence between real and simulated behaviors. These were 1) differences between real and simulated motor/robot-kinetic responses to a given motor command, and 2) differences in sensor triggering when real and simulated robots are in proximity to objects (and simulated objects).

The real and simulated motor outputs were calibrated to within 15%. The simulated tactile sensors always trigger at a distance of exactly 2 inches from the point at which the sensor whisker would be attached to the robot body. In the real environment, and with the real tactile sensors, the binary switch may trigger when the base of the tactile whisker is anywhere between 1 an 3 inches from an object.

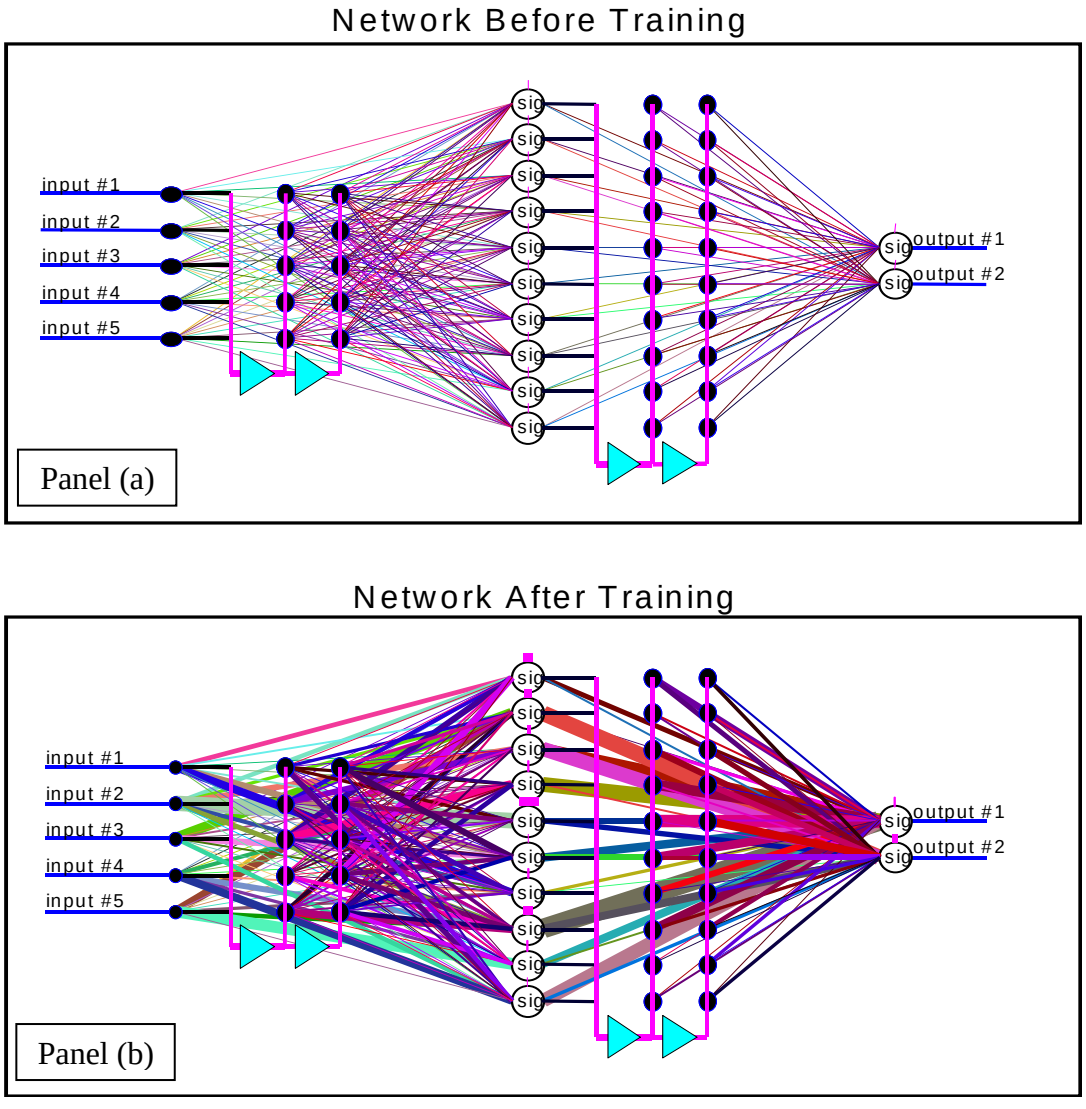


Figure 3.8. Two schematic plots of a single hidden layer time delayed neural network. In panel (a) the network is shown before training. Panel (b) shows the final trained version of the network.

Figure 3.8 shows the states of the best network controllers before and after evolution (training). The performance metric given in equation (3.16) was used with the following parameter settings: $k_1 = 0$, $k_2 = 20$, $k_3 = 20$, $k_4 = 50$. Training performance was averaged for each controller for 3 simulations of 40 time steps before selection and mutation occurred. 350 generations were required to produce the functional controllers tested on the physical robots. The best trained network was found to retain functionality when transferred to a real robot operating in a physical maze similar in dimensions to the one used in training. Figure 3.9 shows the results of three separate tests of the evolved controller operating in the simulated environment and controlling a simulated robot agent equipped with tactile sensors.

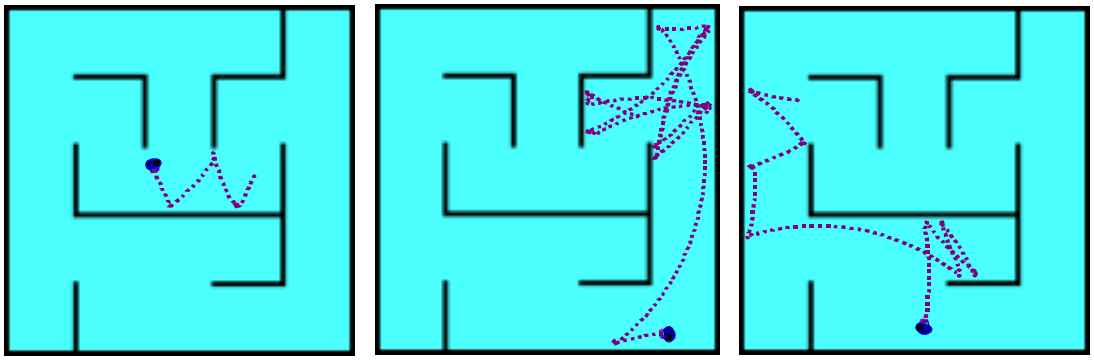


Figure 3.9. Three simulation runs using an evolved controller to navigate a simulated agent with simulated tactile sensors through a maze. The robot agent is shown in its final position after each run in the maze. The dotted line indicates the path taken by the robot.

The robot agent in the panels of Figure 3.9 displayed several different behaviors that allowed it to avoid walls and extract itself from corners and to make progress through the environment. After encountering a wall and backing out of sensor range, robot agents displayed sequences of moves of up to five time steps in duration before stabilizing to a steady-state motor output. This indicates that the neural controllers

evolved responses that could make use of, or at least responded to, sensor information receive up to 5 time steps into the past. Figure 3.10 shows a close-up detail of the path of a simulated robot agent in an encounter with a wall. In the figure, the ovoid square shape is the robot, the black bar at the bottom of the figure is the wall, and the small dotted line indicates the path taken by the robot. The light dots indicate the backward moves made by the robot directly after encountering the wall and two subsequent moves the robot made while it was still in sensor contact with the wall. The sequence of dark dots indicates the sequence of moves (all rotating and/or forward) taken by the robot after it was out of sensor contact with the wall, but before its actuator commands had stabilized to a steady state. The sequence of post-wall-encounter moves indicates the evolved neural controller made use of past sensor information to determine wheel motor commands.

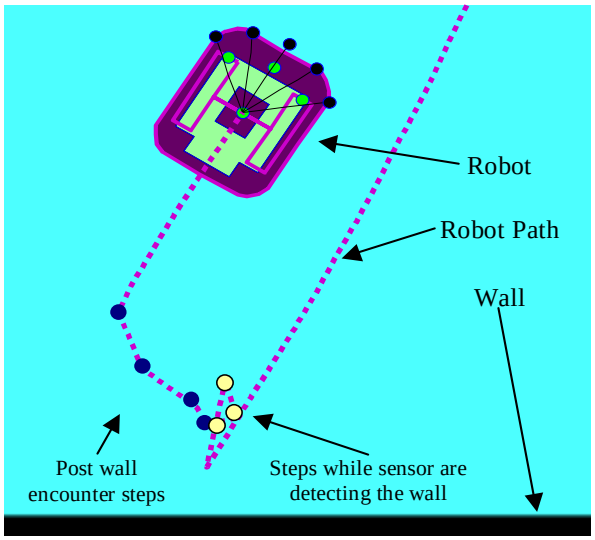


Figure 3.10. Close-up of a simulated robot encountering a wall. The dotted line indicates the path taken by the robot. The light dots indicates the backward move made by the robot directly after encountering the wall and two subsequent moves the robot made while it was still in sensor contact with the wall. The sequence of dark dots indicates the sequence of moves taken by the robot after it was out of sensor contact with the wall, but before its actuator commands had stabilized to a steady state.

Figure 3.11 shows several overhead views of the real maze environment. In each panel of Figure 3.11 a real robot controlled by the trained neural network controller displayed in Figure 3.8 is shown. The dotted line indicates the path taken by the robot in each case.

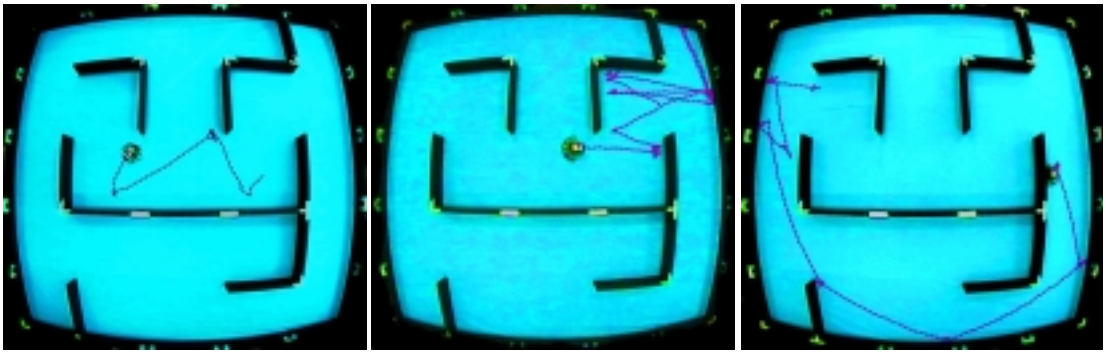


Figure 3.11. Three views of the real maze test bed. In each panel, an EvBot mobile robot with the same evolved neural controller used for the simulation results displayed in Figure 3.9 is shown. The robot is shown in its final position after each run in the maze. The dotted line indicates the path taken by the robot. Qualitatively, the sets of behaviors observed are similar to those displayed using the same controllers in simulation.

The initial positions used for the real robot runs displayed in Figure 3.11 correspond to those used in the simulated world shown in Figure 3.9. It is clear that the real robot agents in the physical environment deviate from the paths taken by the simulated agents after a few encounters with walls. This is expected since very slight differences in approach angle and order of sensor contact can produce markedly different responses on the part of the evolved controllers. The controllers were found to be quite dynamic and displayed a wide variety of similar but not identical responses to encounters with walls. Qualitatively, robots behaved similarly in the

simulation environment as in the real world. It should be noted that when given exactly the same time-sequence of sensor inputs, the real and simulated agents do produce the same motor output command sequences. Sensor input sequences can be quite dynamic in both the simulated and real worlds, however. For instance, a robot may encounter a wall, back up and turn slightly before moving forward into the wall again causing one or more new sensors to be triggered. Very slight differences in the original approach angle can in the end result in different tactile sensor input sequences, which in turn can result in different behaviors. The exact responses for the controllers evolved for this work were not fully characterized. Assuming a possible sensor history of five time steps, there would be 2^{25} possible sensor input sequences. (Note that a controller with five binary tactile sensors and no temporal processing ability would be capable of producing only $2^5 = 32$ responses.)

3. 3 Chapter Summary

This chapter discussed early ER experiments preceding the main body of research presented in this dissertation. Two general sets of experiments were discussed. These were representative of early configurations of the ER research test-bed that was developed over the course of this research. The experiments reviewed in Sections 3.1 and 3.2 were published in [69] and [72] respectively.

Section 3.1 discussed experiments related to the training of neural controllers to produce a simple maze navigation behavior in simulated mobile robots using laser

range-finding sensors. An imitative learning process was used. Robot controllers learned to duplicate the sensor-actuator relationships of a hand-designed controller. Resulting controllers were simple feedforward single hidden layer perceptrons and had no temporal information processing abilities. The simulated laser range-finding sensors provided sufficient depth of information so that purely reactive controllers could navigate successfully.

In Section 3.2, a similar navigation behavior was investigated, but both the robot sensor configuration, and the underling learning strategy were different. In addition, evolved behaviors were transferred to real robots and tested in a physical environment. Here, the robots were made to rely on very simple binary tactile sensors to perform their navigation task. Also, no knowledge-based controller was involved. Controllers were trained with a process of reinforcement learning based on evolutionary computing. Populations of neural network controllers were evolved to maximize performance as measured by a fitness function that selected for maximum travel distance in an environment with obstacles. Using the selection function taxonomy of Chapter 2, this function falls into the "functional fitness function" class described in Section 2.3.1. The neural network architecture was more complicated than that used in the earlier work and allowed for both time-delayed and recurrent connections as well as multiple internal layers. This was necessary because the very simple tactile sensors did not provide sufficient information to produce an adequate purely reactive controller.

A comparative interpretation of these two sets of experiments is that given similar navigation tasks, but using sensors providing different levels of information, more sophisticated controllers can be evolved to compensate for a very reduced level of sensor information.

The second set of experiments also marks a step forward in the complexity of the evolutionary neural network architecture. In later work, large fully-generalized network architectures with arbitrary connectivity and variable size are evolved.

CHAPTER 4. AN EVOLUTIONARY ROBOTICS RESEARCH ENVIRONMENT

A portion of the effort involved in this research was invested in the design and implementation of a research platform (test-bed) that supported the evolution of behavioral robotics controllers. In this chapter both the platform hardware and software components used in this research are discussed.

In overview, the evolutionary robotics research environment developed at the Center for Robotics and Intelligent Machines (CRIM) consists of a colony of small mobile robots and physical reconfigurable maze environment, a coupled multi-robot simulation environment, and an artificial neural network genetic algorithm base controller evolution application. These elements are tied together by a vision-based sensor system that partially processes images into range and substance-type data. Here, substance-type is akin to object type, but no discrete individual objects are implied, only that material of a particular type is detected at a particular range (and angle). This vision processing system plays and important role in increasing the tractability of the simulation implementation.

4.1 The EvBot Platform and Environment

In this section, we will give an overview of the design and capabilities of the real mobile robot colony used in this research. The CRIM has recently developed a new computationally powerful colony of small mobile robots [76][77][78]. These robots have been named EvBots from EVolutionary roBOTs.

The EvBots make up a colony of eight small fully autonomous mobile robots. Each robot is 5 in. wide by 6.5 in. long by 6 in. high and is constructed on a two track treaded wheel base. Each robot is equipped with a PC/104 based onboard computer with an X86 software compatible 32-bit CPU core operating at 133 MHz. The Robots also use non-violate solid-state memory systems including a Disc-On-Chip and an ATA Flash Memory PC-Card. For communications, each robot in the colony is linked to the Internet via a Linksys Wireless Ethernet PC-Card.

A custom Linux distribution derived from RedHat Linux 7.1 is used as the operating system and is capable of supporting MATLAB 5.3 in addition to other high-level software packages. The robots are linked to one another and to the Internet via a Linksys wireless network access point that can support up to 21 devices. Each robot also supports video data acquisition (up to 640x480 live motion resolution) through a USB video camera mounted on each robot. Photographs of several fully assembled EvBots are shown in Figure 4.1. Panels (a) to (c) show pictures of EvBots in various configurations, while panel (d) shows an EvBotII. The recently developed EvBotII is reported on in [78].

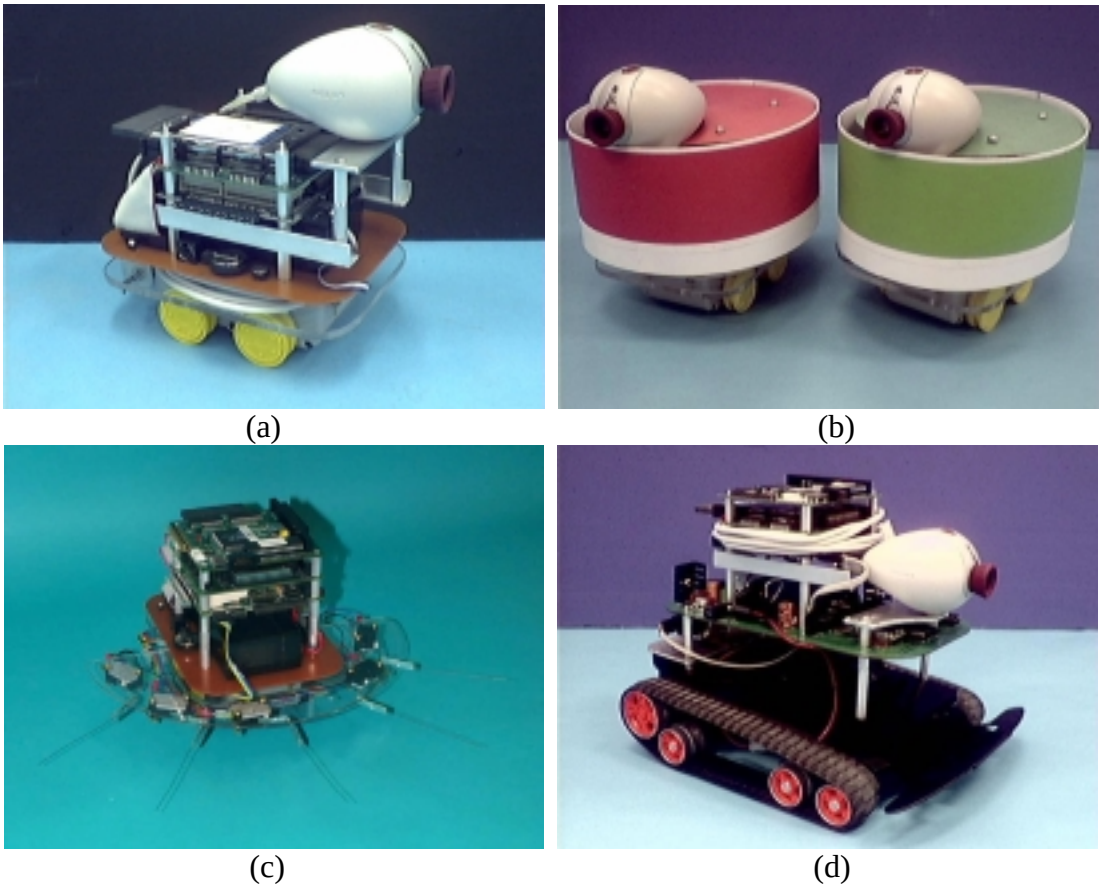


Figure 4.1. Pictures of EvBots in various configurations. The panels show a fully assembled EvBot (a), two EvBots fitted with color shields (b), An EvBot fitted with a tactile sensor array (c), and an EvBotII (the next generation of EvBots) [77] (d).

Each robot in the colony is fully autonomous and capable of performing all computing and data management on board. Most of the controller computer code associated with the evolvable neural controllers resides in the PC-104 and is maintained in MATLAB running under Linux. For the experiments described in this work, robots operated using trained neural networks containing 50 to 100 neurons and 3000 to 8000 weighted connections and a memory of network states of between 5 and 20 time steps. At each time step during controller operation, a single video image is acquired and processed. The processed information requires 30 to 250 ANN inputs.

For the results presented in Chapters 6 and 7, video images were processed to produce 150 ANN inputs. The data from the processed image are then given to the neural network application, which in turn calculates a set of actuator. The resulting actuator commands are processed into PWM signals and sent to the drive wheel DC motors.

The drive wheel motors are controlled by a BasicX micro-controller. An RS232 serial port interfaces the PC/104 CPU to the Basic-X on a custom PCB to control locomotion. The BasicX receives actuator commands and converts them into PWM signals that are sent to the actual drive wheels.

A physical reconfigurable maze environment was constructed for the mobile robot colony. To facilitate vision-based control, the maze was surrounded by a blue backdrop. Robots and other objects in the environment were also fitted with colored shields to aid in identification and positioning. The entire maze environment is viewable from a video camera mounted above. This camera can capture video streams and sequences of images and store them on a remote computer for later analysis. Figure 4.2 show the physical maze environment with several EvBots.

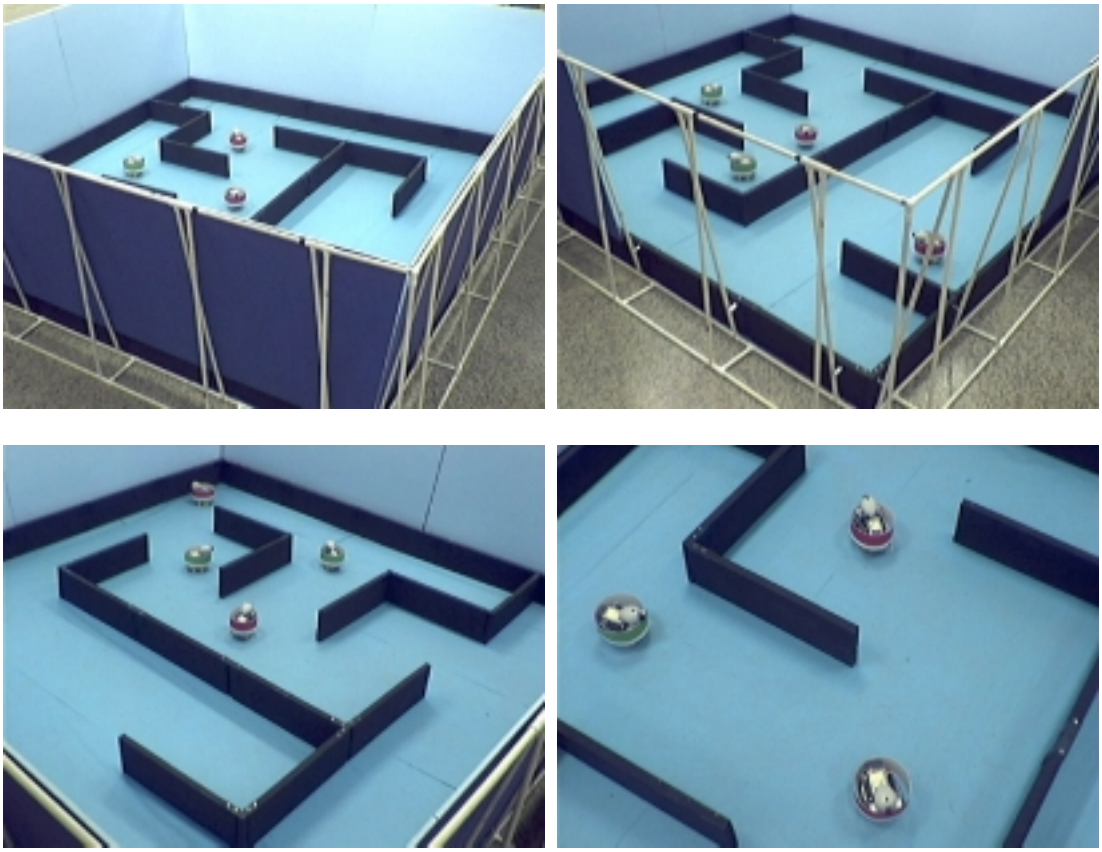


Figure 4.2. Views of the real maze environment with robots.

4.2 Video Range-Finding Emulation Sensors

In the experiments presented in this dissertation, all robotic sensing of environments was accomplished via video (with the exception of the early research presented in Chapter 3). The trained neural networks were fed range data. In order to make this transition from video to range we implemented a simple range-finding sensor emulation system using video images captured from the EvBot USB cameras. This is a simple vision system and it is used to extract information in a form that is useful for mobile robot behaviors including navigation and reaction to spatial situations. The goal of this work is not to develop sophisticated vision systems, but rather to make

use of simple methods to extract information in a form that would be presentable to a neural network based controller. The system is described here to convey the nature and sources of information given to the neural controllers as sensor inputs.

Initially, the motivation for developing the vision-based object range detection system was to emulate laser-range-finding sensors on the real robots so that we could begin to work with controllers that relied on more advance sensing capabilities. Later it was found that video emulation of range-finding sensors provides an advantage over real range finders in that object color can be used to identify object type in addition to distance. This range-finding emulation system provides an important unifying crossover point between the simulated and real environments. Simulation of the emulated range-finding sensors is a much more feasible task than direct simulation of video images.

The vision system takes advantage of fixed geometric elements within the physical maze environment to calculate the ranges and angles of walls and robots. Each robot camera is attached at a fixed angle and altitude. Maze walls are of a constant height so distance can be calculated from a monocular image taken from a set altitude within the maze environment. In addition, each robot is fitted with a shield that has a colored band of fixed width. Robot distances can be calculated from an image by determining the relative width of the colored bands within the image. Likewise, stationary goal objects are also fitted with colored bands of fixed width. In order to distinguish between robots and goal objects, robot skirts are mounted so that they will

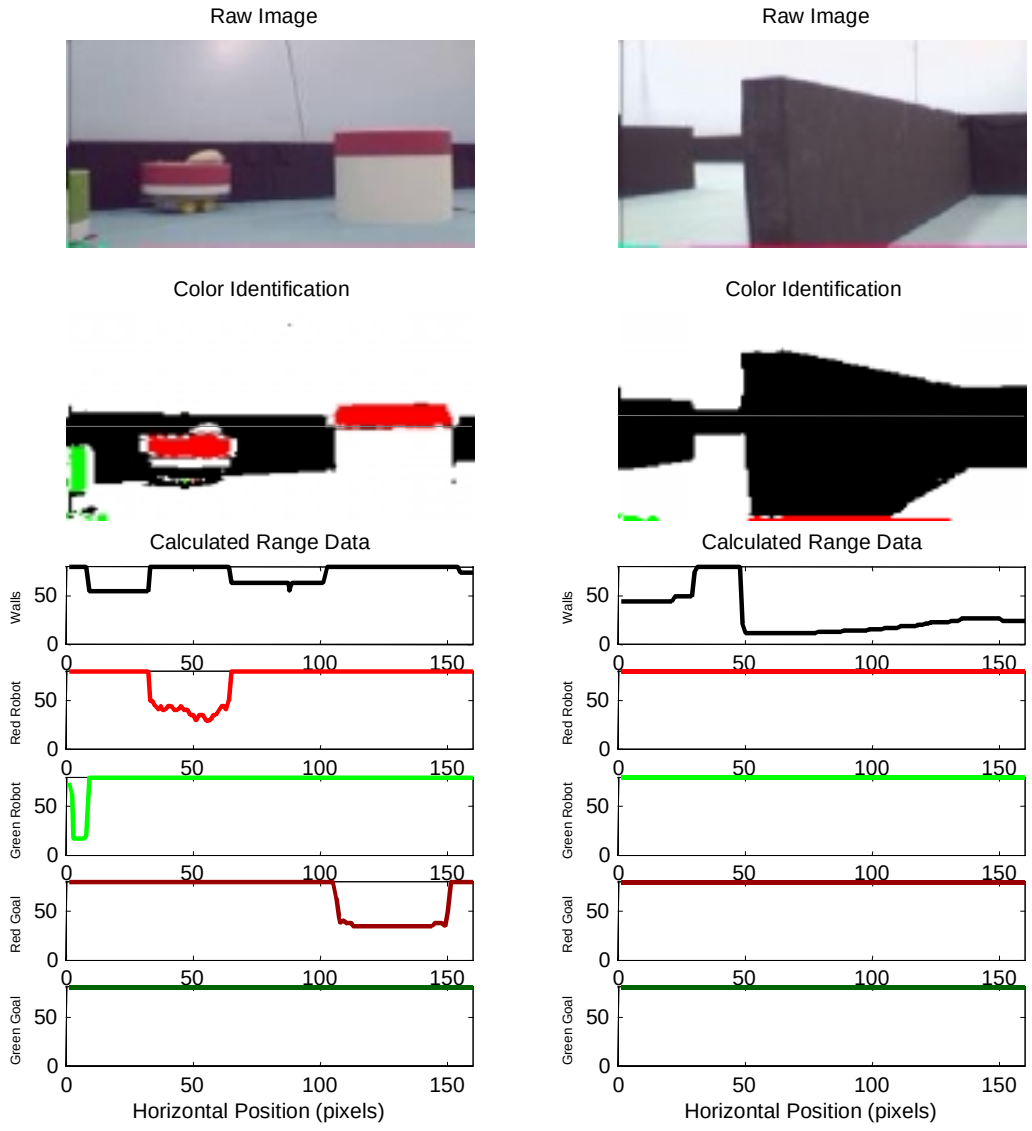
always appear below the horizon in an image while goal color bands will always appear above the horizon. The vision system can detect five object types. These are walls, red robots, green robots, red goal objects and green goal objects. Both range and angle values are reported over a spread of 48 degrees centered on the forward direction of the robot frame of reference. 120 by 160 pixel images were used so a vector of 160 range values is produced for each object type.

The system works by successively decomposing a video image of fixed resolution. First, each pixel is identified as being red, green, black or other (all 'other' colors are ignored). Next, the image is converted to a 2D numerical array where the index of each element is its xy-location in the original image, and its value is an identifying integer depending on the determined color of that pixel. The matrix is subdivided along the horizon into upper and lower regions to distinguish between goal objects and robots. The vertical sum of pixels Σp of each object type is calculated and stored in a separate array covering the horizontal spread of the image. These numerical arrays are then fed element by element through a simple distance formula to produce the final vectors of ranges d for each object type (Equation (4.1) below).

$$d = K\frac{H}{\sum p} \tag{4.1}$$

Where H is the physical height of each object type and K is an empirically derived constant. H and d are in length units (inches). The final form of the data is (for each object type) a vector of numbers spanning the horizontal spread of the original image, where each number represents the distance of the closest object of that type in that direction. If no object is detected, the maximum sensor range is returned. The angle

of detected objects is implicit in the location of each numerical distance within each data vector. Each vector spans the horizontal spread of the robot camera's field of view, and each successive element represents an incremental angular step from left to right across the horizontal field of view.



(a) (b)

Figure 4.3. Examples of image decomposition into vectors of range data to be fed into neural network controller inputs. One vector of length equal to the horizontal resolution in pixels of the image is produced for each type of object in the physical robot environment.

It should be noted that object type information is not explicitly given to the robot neural controllers. Controllers are only given the resulting numerical data vectors. All associations relating distances, angles, and object types must be learned by the neural networks. Object data vectors are always presented to the networks in the same order so a particular scalar input resulting from the distance of an object type in a particular direction will always be presented to the same input. Figure 4.3 shows two example robot-eye-view images and their successive decomposition into range data vectors.

In the reported range data shown in the lower 5 panels of Figure 4.3, closer ranges supersede farther ranges. If one object (or pixels thereof) is found to be closer than another, it shadows the further object. This produces a consistent sensor representation that can be simulated in two dimensions and will be discussed further in the next section.

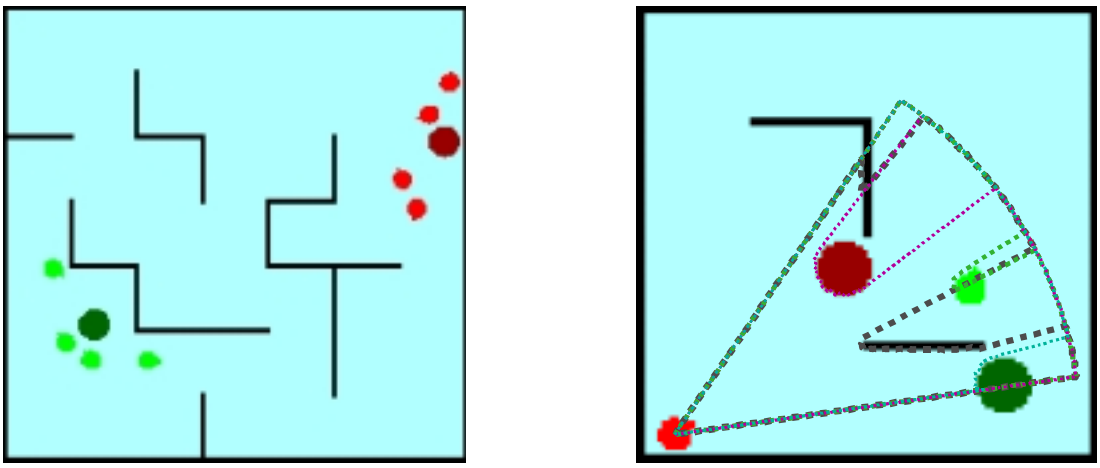
It was found that feeding all 5*160 elements of the full data vector inputs into the neural network evolution application produced networks of large size with very long training times (as to make them untrainable). In most cases, the object range data vectors shown in Figure 4.3 were further reduced in length by extracting the minimum distance over successive groups of horizontal elements. The end results are sets of data similar to those that would be obtained from 5 groups of 30 laser range finding sensors that were selective for a particular object type (5 object types times 30 groups gives 150 total sensor inputs). This further reduction was performed as a

preprocessing step by the neural controllers and was common to both the simulated and real systems.

4.3 The Simulation Environment

Part of the ideology used in the development of both the simulation environment and the physical environment was to identify and utilize points in the simulated and real systems that could be used to tie them together. Such points include sensor data formats, actuator outputs, and elements of calibration. The real sensor inputs (video camera images) were processed to a point so that simulation would be feasible. Here an architecture was implemented that allowed simulated and real controllers to receive sensor data and send actuator commands of the same format. This allowed controllers to be developed in simulation and to be transferred directly to the real robots without any alteration. Although it is generally the goal of simulation to match the real world as closely as possible, in this approach a conscious effort was made to preprocess real world sensor data into a form that would be amenable to simulation and at the same time provide a high level of information to the controllers. The simulation environment does not duplicate raw images similar to those from the real robot video cameras. Rather, it uses elements in the simulated environment to directly produce object range data that is in the same format of the fully processed data from the real cameras.

All objects in a simulation are represented using a 2-dimensional cellular grid. Simulated robot positions are stored as real valued x, y and angle triplets, but when instantiated into the simulated world, they are filled into the closest fit of cells. This allows for the simulation of real valued speeds and rates of rotation without requiring the vector representation of objects. Figure 4.4 (a) shows a graphic representation of a simulated world containing several simulated agents. The smaller circular objects are the robot agents while the lager circular objects are stationary goals. The black lines indicate stationary walls. The positions of the goals and robots are automatically generated at the initialization of a particular simulation.



(a) (b)

Figure 4.4. Views of simulated environments containing robot agents, goal objects and walls. In (a) robots are shown clustered around their respective goal objects. Panel (b) shows a graphical representation of simulated sensor data received by the robot agent in the lower left corner of the environment.

The simulated range finding sensors return one vector of range values per each object type represented in the simulation. Each range data vector spans an angular range constituting the field of view of a robot. In all of the work presented in this

dissertation, the robot vision view span was set to be between +24 and –24 degrees centered on forward orientation of the simulated robot body-attached frame of reference (defined here to be in the positive x direction). This span was divided into 160 equal elements to match the resolution setting of the real robot video cameras. This means that each successive element in a range data vector represents an offset of 0.3 degrees from the preceding and following elements. The value of each element of each vector is the distance of the closest cell containing an object in the direction associated with that element. If no objects of a particular type are found, distance values are set to a fixed maximum range. For all object types, only the closest element associated with a direction is reported. All other elements in that direction in all of the other vectors are set to the maximum range. This effectively restricts the robot to a two dimensional view. Any further reduction or processing of sensor data is done by the robot controllers and is common to both the simulated and real controllers. Figure 4.4 (b) shows a graphical representation of sensor data plotted over a simulated world. This representation is somewhat akin to what a person would see if he or she were looking at a picture of an aerial view of a robot in a foggy field with a searchlight attached facing forward. Light rays would be blocked by the objects as they diverge from their source on the robot. If one were to trace around the illuminated portion of such a picture, a pie-piece-like shape similar to that shown in the figure would result. One such ‘pie-piece’ shape is superimposed onto the figure for each object type. Also, note that the ‘pie piece’ is cropped off at a certain radius. This represents the maximum sensor range setting.

The simulated environment extracts range data from the environment and converts them into the same format as is reported by the video range sensors. In addition, drive wheel speeds are given in physical units (in/sec). The simulation converts these into motion of the simulated agents while the real robots convert them into a calibrated set of actual motor commands in the form of DC PWM levels. This allows for the tuning of both the simulation environment and the real robot actuators to couple the two systems.

Collisions between moving objects or between a moving object and a stationary object are modeled by disallowing object overlapping. If a moving object's next calculated position intersects a second object, it is not allowed to move. Rather, the time step is successively reduced and the next position is recalculated and the move is reattempted. This continues to the point that calculated next position is within one world-cell size or less. If this still produces an overlap, the object (robot) is considered stuck, at least until the next simulation iteration cycle, when it is possible that the robot agent will back up. This affectively models 100% friction at all contact points and is more restrictive than the real world. It was found, however, that the real robots do become stuck when involved in a wall collision. Robot-robot collisions in the real world can eventually result in one robot pushing the other out of the way. This is not accurately modeled in the simulation, where robots would just push against each other without producing any further motion.

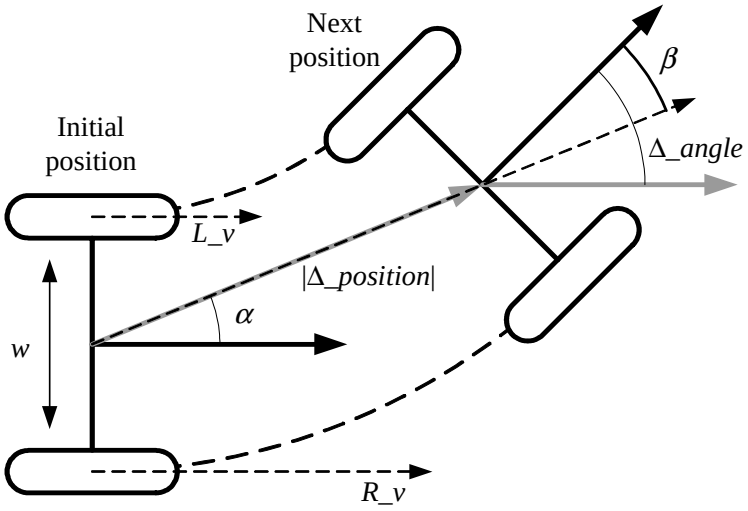


Figure 4.5. Robot differential steering model.

The simulated agents model a deferential steering method of locomotion. Each tread on a real robot is represented by a single point contact in the 2-dimensional simulation. The simulations are necessarily discretized with respect to time with time step size Δt. The next position of each robot agent is a function of its current wheel speeds and of the length of the time step. The calculations used to determine the simulated robot's next position are:

$$\begin{aligned} L\_dist &= L\_v * \Delta t \\ R\_dist &= R\_v * \Delta t \\ \alpha &= \sin^{-1}\left(\frac{R\_dist - L\_dist}{w}\right) \\ \beta &= \frac{\text{mean}(R\_dist, L\_dist)}{\max(R\_dist, L\_dist)} * \alpha \\ \Delta\_angle &= \alpha + \beta \\ \left|\Delta\_position\right| &= \text{mean}(R\_dist, L\_dist) \end{aligned} \tag{4.2}$$

where *L_v*, *R_v*, Δt and *w* are the left wheel linear velocity, the right wheel linear velocity, the time step size and the robot wheel-to-wheel body width respectively. The resulting direction of position offset, magnitude of position offset and angular

offset of the next position are given by α, $|\Delta\_position|$ and Δ_angle respectively. Figure 4.5 shows the results of the next position calculations in (4.2) for a robot agent wheel set with respect to a body-attached frame of reference. Note that although the EvBots actually have four drive wheels, the front and back wheels always act in tandem (like drive wheels in an army tank tread), and do not steer individually. Hence, the two wheels acting together on one side of a robot can be modeled as a single wheel or a single tank tread. The interface between the controllers and the real robot drive systems allows for calibration at a fine level: the real robots can be calibrated to match the motions of their simulated counterparts and vice versa for a given speed command. This allows for a great deal of flexibility in modeling methods and in actuator command formats.

4.4 Simulated Vs. Real Sensors

Figure 4.6 (a) shows an image of the real maze environment with a graphical representation of real sensor readings superimposed onto the image. Here, the sensor data were gathered by a robot in the center of the maze. In part (b) of Figure 4.1, the environment configuration is duplicated in simulation. Again, sensor data were taken from the center of the simulated maze and from the same orientation as the real robot in the real maze. The simulated sensor data were also superimposed onto the simulated maze graphic. The graphical representation of the sensor data is similar to that of Figure 4.4 in the previous section. Additional comparative real and simulated sensor data examples are shown in panels (c) to (f) in Figure 4.6.

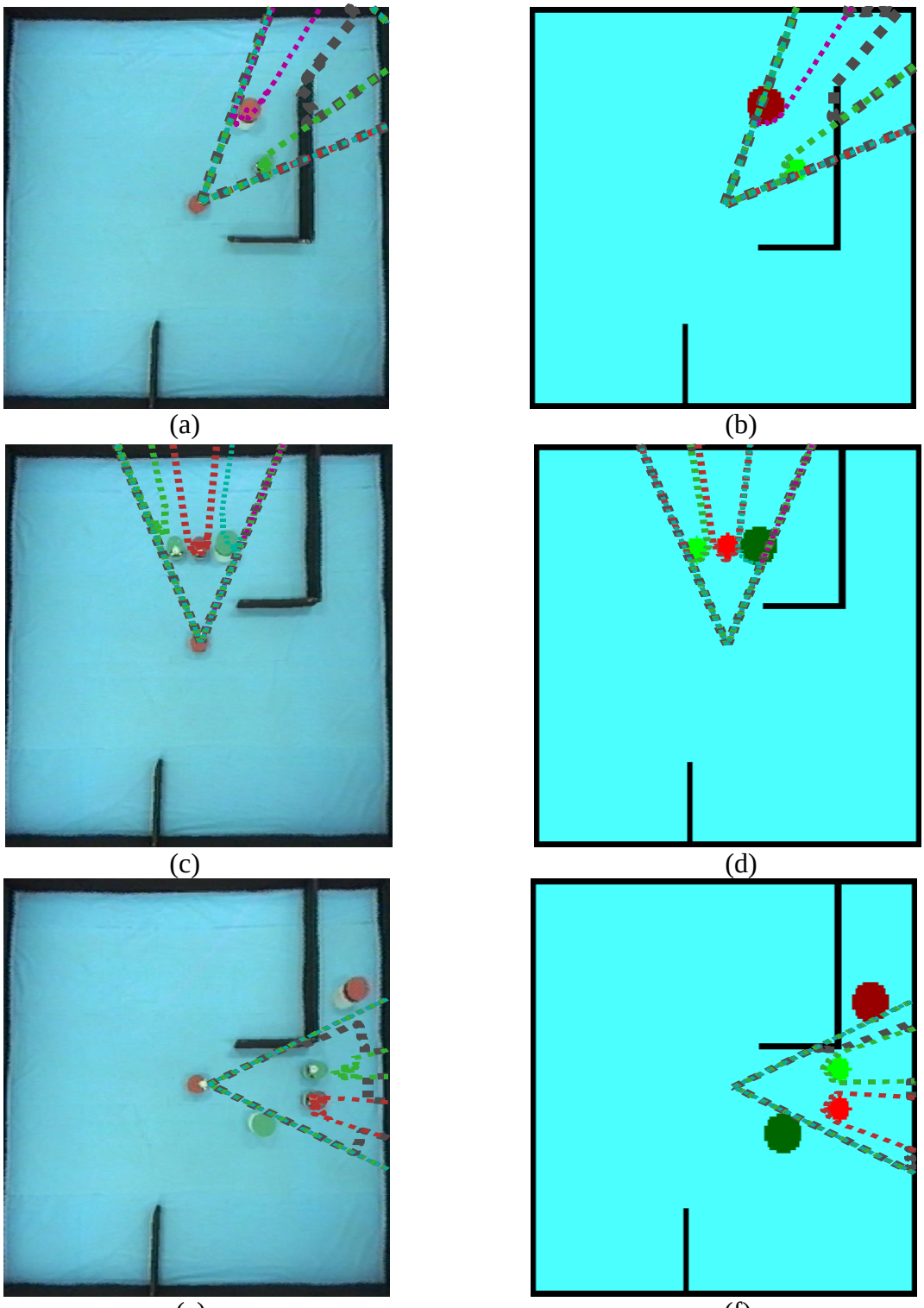


Figure 4.6. Comparative real and simulated sensor plots. Real sensor readings are plotted on images of the real maze environment (a) (c) (e) These are compared to simulated sensor readings generated in the simulation environment (b) (d) and (e). For each image pair, the real and simulated worlds were configured similarly.

To investigate and quantify the fidelity of the video-range emulation sensor system, the sets of real and simulated sensor reading were compared. A set of 10 images similar to those in Figure 4.6 (b) (d) and (f) were taken of the real maze environment with real robots. These were correlated with sensor data produced by the robot in the center of the maze in each image. The maze environment configurations were then duplicated in the simulation environment and simulated sensor readings were calculated. The real and simulated sensor reading data sets were compared to generate a measure of quality of the real sensors. The real vision based sensors produced and error of 21.0% when compared to simulated values. This is about 12.44% error with regard to the maximum range of the sensors.

This sensor system was primarily used for robots being controllers by neural networks trained to play a competitive team game (*Capture the Flag*). In that case, rather than seeing red and green, robots see "my color" and "opponent color". This was achieved by swapping the red distance sensor inputs and green distance sensor inputs for controllers on opposing teams both during training, and during testing.

4.5 The Evolutionary Neural Network Architecture

The vast majority of ER research involves the evolution of artificial neural networks (ANN). Some other robot controller structures have been used and have yielded preliminary results. For example A limited amount of ER work has focused on the evolution of genetic programming (GP) structures for robot control [54][55]. GP

syntactic constructs may restrict the controller search space so that the evolution of controllers for general complex tasks is not feasible. Other evolvable control structures such as Q-learning, evolvable state transition structures and evolvable logic hardware have seldom been used in ER work. In the research reported on in this work, ANN controller structures were evolved. ANN controller structures provide a continuous search space and are easily encoded into a variety of genome representations.

Because the dynamics of behavioral robotics tasks are not well characterized, we believe it is important to apply artificial evolution on a broad relatively unrestricted network morphology search space. Much ER work to date has made use of small static networks [53][45][50][22][40][79][67]. The results reported on in those works show that small networks are evolvable to perform simple tasks, but it is not clear that those results can be generalized to the complex case. Other researchers have used more complex networks [15][29][17][90] and it is this path that is pursued herein. In several case studies, networks containing recurrent connections were shown to significantly outperform feed forward networks [41]. We have developed a generalized neural network architecture capable of implementing a very broad class of network structures. Networks are not limited to any particular layered structure and may contain feed forward and feedback (recurrent) connections between any of the neurons in the network. Networks may contain mixed types of neurons, and a variable integer time delay may be set on the inputs of any neuron in the network. Internal neuron activation function types include sigmoidal, linear, step-threshold,

and Gaussian radial basis functions. For purposes of population based evolutionary training, populations of heterogeneous robot controller networks can be specified, altered, and evaluated within an artificial evolution environment. In the next section, a formal description of the evolvable neural architecture used in this research is given.

4.5.1 Network representation

The connectivity and weighting relationships in a given network are completely specified by a single two-dimensional matrix **W** of scalar weighting values. Information specifying neuron types is given in a vector data structure **N** with one formatted field per neuron. All inputs, internal neurons and outputs are considered to be types of neurons as far as the network specification matrices are concerned. Each of these is given a field in **N** to make a total of N fields. Current and past network inputs and neuron functional levels (outputs) are stored in an ordered matrix, **I**. Each row of **I** contains the inputs and activations associated with a particular time delay starting with the current time (delay of 0) and progressing into the past with successive rows to the maximum time delay supported by the network. The maximum level of time delay supported by a network is specified by a scalar integer, δ. This makes **I** a matrix of constant size determined by δ and N with element values that vary with time. This formulation allows for the efficient implementation of a variety of evolutionary training methods and for the formulaic specification of a very broad class of network topologies.

Activation functions take the form:

$$f_n(u) = f_n(\mathbf{w}_n, \mathbf{i}(t, \tau(n))) \tag{4.3}$$

Where $n \in \{1..N\}$, $\mathbf{w}_n$ is the nth row of the weight matrix **W**, $\mathbf{i}(t, \tau(n))$ is the τth row of the input/activation matrix **I** at time t, and f_n is the activation function type specified in the nth field of **N**. The time delay, $\tau(n)$ is also defined in the nth field of **N** and is written as a function of n. In the majority of neuron activation functions, u takes the form of the weighted sum (dot product) of the inputs and the associated weights:

$$u = \sum_{m=1}^{N+1} w_m i_m \tag{4.4}$$

where $N+1$ is the width of **W** and of **I**. These activation function types include sigmoid, linear and step functions. Note that biases are accounted for by the addition of a column of inputs in **I** that are always 1, and by an additional column in **W** of associated weights. For the radial basis activation functions, u is the Euclidian distance between **w** and **i** in N-space given by:

$$u = \sqrt{\sum_{m=1}^{N} (w_m - i_m)^2} \tag{4.5}$$

where **w** and **i** have been reduced in dimension to match that of the non-zero weighting connection elements of **w**.

Network inputs are considered to be linear neurons with all zero connecting weights except for a single self-self connection with a unit weight. Networks with more complex 'input functions' are possible but were not explored in this work. There is no distinction between hidden and output neurons except that outputs are specified as such and their function outputs can be selected and read from the matrix **I** after a

network updating cycle. The input-output relation for a given network can then be specified as follows by:

$$\mathbf{I}(t+1) = Network(\mathbf{I}(t), \mathbf{N}, \mathbf{W}) \tag{4.6}$$

and,

$$\mathbf{o} \subset \mathbf{i}(t+1,1) \tag{4.7}$$

Where **o** is a vector of values from specified output neurons and is a sub-set of the first row of the new **I**(t+1). Initially, the network inputs are read into the first elements of the first row vector of **I**(*t*). The functional *Network* calculates the outputs of each neuron specified in **N** in order, placing resulting values in successive elements of **I**. The values of **I** are thus altered by *Network* during calculation of the network. The resulting state of **I** is considered **I**(*t+1*) after shifting each row of **I** to allow for the incrementing of time. Specifically, the functional *Network* can be expanded as follows:

$$\mathbf{I}(t+1) = \begin{bmatrix} [i_1,...i_L, f_{L+1}(\mathbf{w}_{L+1}, \mathbf{i}'(t+1,\tau(L+1)),... f_N(\mathbf{w}_N, \mathbf{i}'(t+1,\tau(N))] \\ \mathbf{i}(t,1) \\ \mathbf{i}(t,2) \\ \vdots \\ \mathbf{i}(t,\delta-1) \end{bmatrix} \tag{4.8}$$

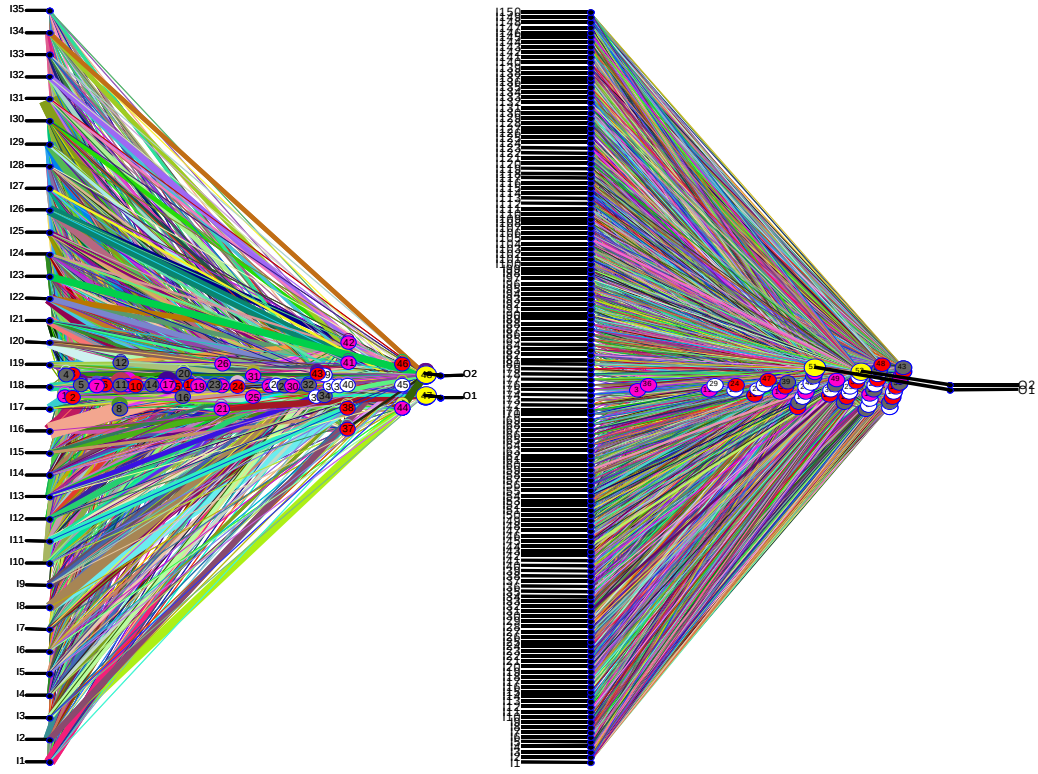
where L is the number of external inputs into the network and $\mathbf{i}'(t+1, \tau(m))$ is the input/activation vector containing elements of $\mathbf{i}(t+1)$ up to the mth element and elements of $\mathbf{i}(t)$ in the remaining portion. δ is the maximum level of time delay memory explicitly supported by the network. An ordered neuron update sequence such as this is required because networks containing arbitrary feedback connections can be made to produce different outputs simply by altering the order in which

internal neurons are activated. Note that the first L elements of the matrix in equation (4.8) could also have been written in the form of activation functions but since their associated functions are linear with a single unit weight, they remain unaltered by the calculation iteration sequence.

It should be noted that a particular network's theoretical memory limit is greater than the explicit memory length δ. For a network with only feed forward connections, the maximum possible memory is given by the length of the longest sequence of connected neurons terminating in an output multiplied by δ. The dynamics of memory resulting from connectionist systems is closely (and perhaps inextricably) coupled with network performance. It is unlikely that networks developed in this work evolved behaviors that approached their theoretical memory limit. Nonetheless, we wanted to implement a controller network architecture that would provide at least the possibility of development behavior that involved memory.

4.5.2 Graphic Representation of Neural Networks

Figure 4.7 shows two evolved networks from different populations. In the figure, network inputs are shown on the right while driving motor outputs are shown on the left. The two motor outputs are used for differential steering control of the mobile robots upon which the evolved controllers in this research were implemented (the EvBots [76] [77]).



(a) (b)

Figure 4.7. Two examples of evolved neural networks. The network in panel (a) has 35 inputs, while the network in panel (b) has 150 inputs. Both example controllers networks have two motor outputs that deliver speed commands to the robot's drive motors.

In the graphic representations of the neural networks, neuron positions are calculated as a function of degree of input connections and relative neuron order in the ANN representation structure. The representation used for calculation and evolution of neural networks is discussed in the next section of this chapter. Here, we briefly discuss the format of the graphic representation to clarify the structural representations in Figure 4.7. It is important to note that the selection of a particular graphic representation has no effect on network function and serves only to provide a human observer with some sense of a network's connectivity relationships. Each neuron is given an integer order number in relative to its position within the 2

dimensional connectivity matrix **W**. Each neuron position in the graphic representation is dependant on that neuron's input connections: the input originating from the neuron or sensor connection with the highest order number determines position. The greater the order number is, the further to the right within the overall structure) the neuron will be drawn. In the case that two or more neurons have highest numbered inputs from the same neuron, they are stacked from bottom to top in their order of representation numbering, and with a slight offset from left to right. This graphic display format provides for the illustration of some characteristic patterns that arise in large arbitrarily connected networks with many inputs. For instance, a network that had a layered structure would appear as such. Also, a network that has no recurrent connections but is otherwise randomly connected will have a cone-like distribution of its neurons. The networks in figure 4.7 have both forward and backward connections. This gives rise to a sort of oval distribution of neurons. Note that the overall cone shape of the connections is due to the large number of input connections, which always appear in a vertical layer on the left.

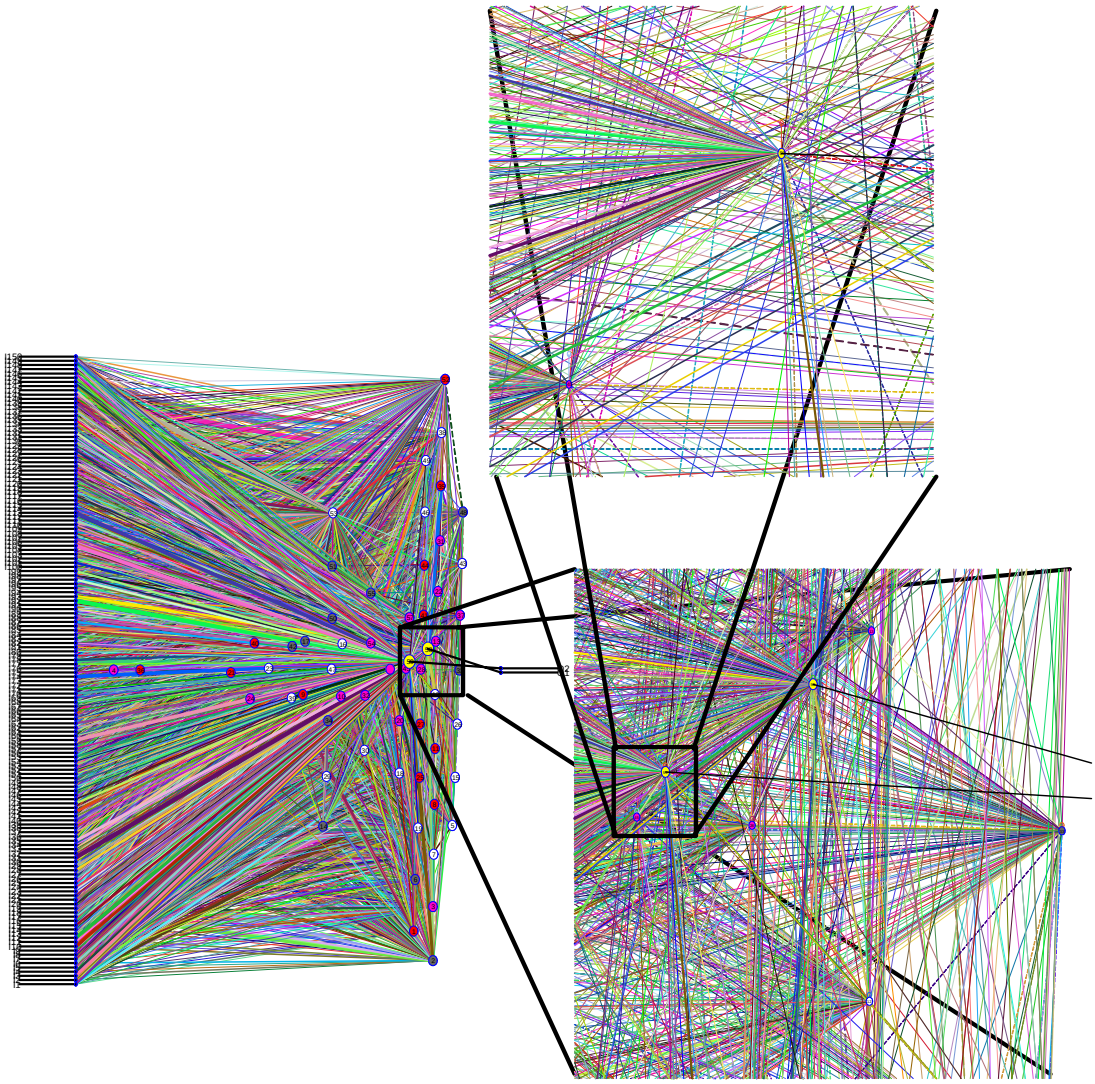


Figure 4.8. Here a fully evolved controller network is shown in several magnifications.

Figure 4.8 shows the depth of complexity possible in the evolved neural controllers. This network is of the same type as those shown in Figure 4.7. Here, though, the internal neurons are spread over a greater distance in the vertical dimension. The figure shows two successive magnifications of the connectivity structure of the network.

4.5.3 Network Representation and The Genetic Algorithm

These generalized architecture neural networks are intended to be trained using stochastic and evolutionary computing methods. Such methods are often used for neural network applications in which a well-defined error function is not known. The words "evolution" and "training" are used interchangeable here. This use is representative of the evolutionary process being used as a method to implement reinforcement training or learning in a population of neural network controllers.

Behavioral robotics neural network based controllers are prime examples of systems where it is difficult to exactly define error metrics for determining the quality of the network output. For example, we may want to train a mobile robot controller to follow a corridor without hitting the walls. In this case, the behavior of the robot is evaluated, not the output of the neural network. There is no defined training input-output data set. The desired network output is in fact not known; hence it is not possible to formulate error measures for error back propagation (BP) based training algorithms. Even if desired outputs were known for a given set of inputs, analytical error BP methods require that the network structure be simple enough to formulate functional equations that can be analytically differentiated in an automated fashion. The complexity of implementing BP on networks of arbitrary structure would be quite high: possibly beyond the point of tractability. In addition, BP searches a given network's weight space, but does not search network topology space. Genetic and Evolutionary algorithms offer and alternative to error back propagation training

methods and can be formulated to make use of behavioral fitness evaluation for selection.

In this section, the genetic representation and algorithm use to evolve the controller networks for this research are discussed. The training fitness selection function used to drive the selection process is the subject of the next chapter.

The genetic algorithm used in this research acts directly on the data structures that encode the neural networks. There is no secondary encoding into a symbolic chromosome. This means there is necessarily a one to one correspondence between the genome and possible neural structures that can be supported by the neural network application. If a certain class of structures is less likely to occur in the genetic space, it is equivalently less likely to occur in the network space. Although this is the case with most GAs that operate directly on an ANN weight set, it is not necessarily so for cellular encodings such as the ones used in [15][12][24].

As discussed in Section 4.5.1 above, the connectivity and weighting relationships in a given network are completely specified by a single two-dimensional matrix **W** of real valued scalar weights. Additional information specifying neuron types and time delays is given in a data structure **N** with one formatted field per neuron. **W** and **N** from the basis of the genetic encoding for each network. Formally, the genome (sometimes referred to as a chromosome) **C,** for a network can be specified by the two dimensional matrix of real numbers.

$$\mathbf{C} = [\mathbf{W} : \mathbf{N}'] \tag{4.9}$$

where **N**' is a matrix of scalars extracted from the formatted data structure **N**.

During evolution, networks are mutated in three ways. First, connection weight values can be perturbed. Second, connections can be added or removed. Finally, neuron units can be added or removed. Mutation of a network can be formalized by the compound relation

$$\mathbf{C}' = M_s(M_c(M_w(\mathbf{C}))) \tag{4.10}$$

where **C** is the chromosome of the parent network and **C**' is the resulting mutated offspring network chromosome. M_w M_c and M_s are genetic operators that mutate the weights, the connections, and the neuron structure of the network respectively. Any or all of the different types of mutation can occur during propagation.

Populations of 40 networks were evolved. Each generation consisted of a competitive tournament of games played between the controllers in the evolving population. The genetic algorithm applied in this work used 50% selection and replacement. Members of the fittest 50% of the population were mutated to create one offspring each. These offspring then replaced the least fit members to make the next generation. Single time-elitism was also used to allow the fittest individual from the two generations back to be included in the current generation. For all of the evolved ANN controllers populations reported on in this work, mutation rates were set at 25% while weight mutation magnitudes were set to be in the linear distribution (-1, 1). In

addition, during mutation networks had a 70% chance of adding or removing a neuron and a 70% change of adding or removing an arbitrary connection and associated weight. All networks had a fixed number of inputs (150 for this work) and a fixed number of outputs (2 for the robot wheel actuators). Networks had between 40 and 100 neurons and on the order of 5000 arbitrary feed forward and feed back connections during training. Network size was not explicitly constrained and variations in size reflect differences arising due to selection.

A note about the controller search: The ANN controller space is a variable dimension real-valued space that is continuous at each n-dimensional level and possibly discontinuous between dimensional levels. (Adding and removing neurons and connections introduces the variable dimensionality.) As with any space with real-valued variables, our controller space is uncountably infinite. Also, in the general case and for non-trivial behaviors, the proportion of fit solution candidates in the space is (most likely) infinitesimally small.

4.6 Conclusion

In this chapter, the major elements of the CRIM evolutionary robotics research test-bed were discussed. These included 1) a colony of real robots, 2) a vision based range-finding sensor system, 3) a physical reconfigurable maze environment, 4) a closely coupled simulation environment that simulates multiple robots, the vision based sensors, and the maze environment, and 5) an evolutionary neural computing

environment that supports the evolution of populations of ANN-based behavioral mobile robot controllers.

Also, results comparing simulated and real vision based range-finding sensor values were presented. It was found that the real vision system generated an approximately 15% error or noise level when compared to similar noiseless simulation sensor values. As will be seen in the next two chapters, evolved controllers were robust enough to compensate for this level of error when transferred from simulation to reality.

CHAPTER 5. EVOLUTION OF ROBOT NEURAL CONTROLLERS USING COMPETITIVE FITNESS FOR SELECTION

In this chapter a fitness selection metric (fitness function) for competitive team robot tasks is defined and used to evolve populations of neural network controllers to operate teams of mobile robots playing the game *Capture the Flag*. The function is based on relative competitive selection. At each generation, a tournament of games is played involving each of the members of an evolving population. A bimodal fitness function is used. The initial mode of the fitness function accommodates the Bootstrap Problem early in training by selecting for basic a motive behavior. This basic motive behavior is defined to be the ability to travel half way through the robot's environment. This definition reflects human bias and requires a degree understanding of the robot and system dynamics to formulate. However, it is desirable to limit the amount of human bias needed to evolve complex controllers because such bias can greatly limit the controller search space. In the case of relatively complex robot tasks, human biased and specialized hand formulate fitness functions can in effect force evolving controller populations to converge to predefined solutions. To remedy this, later in training, selection is based on win/lose information only. This is accomplished by defining a second mode to the fitness function. The second mode uses only aggregate task completion to calculate fitness. Controllers in the population that are able to win games are not affected by information from the initial mode. Once controllers evolve the ability to win games, they are allowed to regress with

regard to the first hand-formulated selection mode. So long as they continue to win games, they will continue to be selected and propagated based on high level task completion. Hence, later evolution continues with less human bias than if a single mode fitness function were used.

In Section 5.1, the team robot game used for this research is briefly described. In Section 5.2, a formal definition of the bimodal fitness function is given. Section 5.3 defines experimental and evolutionary conditions used to evolve the populations of controllers. Sections 5.4 and 5.5 present training data collected over the course of evolution of controller populations. In Section 5.6 evolved behaviors are analyzed qualitatively and in Section 5.7 evolved controllers are tested in real robots.

For this research, a number of lengthy and computationally expensive evolution runs were conducted. The evolution runs required three to four weeks of computation time each, and were conducted on several desktop computers using 0.7Ghz and 1.4Ghz processors. Data from four populations of controllers evolved under different conditions are presented. The evolution conditions compare environmental-incremental and "all-in-one" evolution. In addition, and secondarily, two forms of opponent selection for games within the generational tournaments were investigated.

5.1 The Task: Robot *Capture the Flag*

Before giving an explicit formulation of the bimodal relative competitive fitness function, we define the competitive game used in this research.

Populations of robot controllers were evolved to play a robot version of the competitive team game *Capture the Flag*. In this game, there are two teams of mobile robots and two stationary goal objects. All robots on team one and one of the goals are of one color (red). The other team members and their goal are another color (green). In the game, robots of each team must try to approach the other team's goal object while protecting their own. The robot which first comes within range of its opponent's goal wins the game for its team. The game is played in maze worlds of varying configurations. In general, larger environments with more walls and corridors are more challenging. Robot controllers were evolved and tested in various world configurations.

There are no explicit rules for this game except that the robot first coming in contact with the goal object of its opponent's team ends the game with a win. The only other constraint is that robots are not allowed to break any physical laws. This can lead to the evolution of some seemingly odd strategies, especially early in the evolutionary process. For example, in a game in which one robot on a team crashes into its own goal, but another on the same team finds the opponent's goal will still result in a win for that team.

5.2 The Bimodal Fitness Selection Function

This research demonstrates the use of a bimodal fitness function. The function is applied to competitive fitness selection in a population of evolving robot controllers. The function is designed to accommodate sub-minimally competent initial populations early in evolution, but to relax selection to be based on overall competitive success/failure or win/lose information as populations become minimally competent.

Here, the term "sub-minimally competent" means that controllers in a population have no detectable ability to complete the target behavior fully. Also the terms, fitness function, selection function, selection metric and so on, are all used interchangeable. These terms refer to the function that is applied to the behavior of each individual in a population in order to measure its relative fitness for survival and selection during propagated to the next generation.

Selection algorithms that base fitness purely on high level success or failure of complex behaviors have much greater potential for generalization to complex behaviors than specialized hand-tuned functional fitness functions. In the ideal case, controllers are selected based only on frequency of completion of a given high-level task. Success/failure fitness selection of this type represents an aggregation of evaluation of all sub-behaviors into a single all-encompassing measure. This type of selection is important to the future application of ER to difficult real-world problems

because it can be applied toward the evolution of behaviors that humans do not understand well enough to formulate functional fitness metrics for.

In the competitive evolution case, evolving controllers continue to improve and thus produce a changing fitness landscape over the course of evolution for the population as a whole. This phenomenon of fitness landscapes being subject to change due to evolving elements within the evolutionary process is sometime referred to as the "Red Queen Effect" in evolutionary computation. Competitive selection can allow for the continued ramping up of task difficulty due to improvements of competing members in a population. As an individual becomes more competent, it represents more of a challenge to other members in the population, and visa versa.

The problem with pure success/failure based fitness selection is that in the initial stages of evolution, random seed populations often show no detectable level of fitness to perform an entire complex task or behavior. They are sub-minimally competent. This is commonly referred to as the Bootstrap Problem in ER and related research fields.

Rather than trying to form a complicated function to optimize a specific behavior, we formulated a function with two modes of selection. In the first mode, controllers are selected so that they have the potential to complete the overall behavior a small fraction of the time. Controllers need not be well evolved to complete the task. They must only be able to complete the task on occasion (win a game in this case). Once

this is achieved, fitness is based on pure success or failure. This reduces the amount of system knowledge needed to formulate the training metric. Designers must only be able to identify a minimal level of performance that would allow robot controllers to produce a detectable level of completion of the overall task. Once this level of fitness is achieved, further evolution can be based on aggregate success or failure of the task. The designer must also be able to design a function that will detect successful overall completion of a given task.

In summary, we implemented a training function with an initial mode that accommodates sub-minimally competent seed populations and a second mode that selects for aggregate fitness based only on overall success or failure. Additionally, we applied this selection metric in a relative competitive form in which controllers in an evolving population compete against one another to complete their task -to win a competitive game.

Formally, in this research fitness $F(p)$ of an individual p in an evolving population $\mathbf{P}$ ($p \in \mathbf{P}$) takes the general form:

$$F(p) = F_{\text{mode\_1}}(p) \oplus F_{\text{mode\_2}}(p) \tag{5.1}$$

where $F_{\text{mode\_1}}$ is the initial minimal-competence mode and $F_{\text{mode\_2}}$ is the purely success/failure based mode. Here $\oplus$ indicates dependant exclusive-or: if the success/failure based mode's value is non-zero, it is used and any value from $F_{\text{mode\_1}}$ is discarded. Otherwise fitness is based on the output of $F_{\text{mode\_1}}$.

$F_{\text{mode\_1}}$ returns only negative values in the range {-a, 0} while $F_{\text{mode\_2}}$ returns only positive values in the range {0, a}. This insures that successful completion of the task achieved by an evolving individual during a generation will supercede the fitness of any population member not fully completing the task. This also gives an indication of which mode of the metric is in operation for any given fitness calculation.

The first mode of the fitness function operates primarily during early training. Its purpose is to select for minimal competence to successfully complete the task (win the game) in a detectable fraction of the trials, and in a finite amount of time.

We define a minimal competence for this game task to be the ability to travel half way through the current training environment, (or more precisely, the ability for at least one robot of a given team to travel a distance equal to half the length of the environment's greatest dimension). The rational is that for this task, and many other mobile robot tasks, a robot that can travel a minimal distance through its environment has the possibility of completing a more complex locomotion-based task. A robot controller will maximize this mode of the fitness function if it can travel halfway through its environment. Further navigation is not rewarded. The general form of mode 1 is as follows:

$$F_{\text{mode\_1}} = F_{dist} + s + m \tag{5.2}$$

where F_{dist} calculates a penalty proportional to the difference between distance d travel by the best robot on a team, and the minimal competence distance D:

$$F_{dist} = \begin{cases} \alpha * (D - d) & \text{if } d < D \\ 0 & \text{otherwize} \end{cases} \quad (5.3)$$

D was defined above as being half length of the current training environment's greatest dimension and α is a constant of proportionality. In Equation (5.2), s and m are penalty constants applied in the case that robots on a team becoming immobilized or stuck (by any means), and in the case of controllers producing actuator output commands that exceed the range of the actuators (the wheel motors) respectively. Note that m is unrelated to performance of any particular task, but produces selective pressure for ANN controllers that generate actuator commands in ranges that the motors are capable of producing. Controllers that produce out-of-range actuator commands still function, but the actuators produce outputs at the edges of their ranges as near as possible to the given out-of-range command.

F_{dist} and s required some level of knowledge on the part of the designer to formulate. Hence, human bias has not been completely eliminated and is injected into the selection process. Even so, this mode need only select for robot teams that have the potential to win games some of the time, however poorly. Any team controller wining a game will have a fitness based only on $F_{\text{mode\_2}}$ and will not be subject to any penalty from mode 1 of the fitness function. This means that the human-bias induced constraints introduced by $F_{\text{mode\_1}}$ are lifted as evolving populations evolve past the minimal competence level.

The second mode of the fitness function is classified as aggregate (using the classification presented in Chapter 2) because it produces fitness based only on

success or failure of controllers to complete the task at hand (competitive team game playing). The formulation of the success/failure mode (F_{mode_2}) of the fitness function is determined by the competitive nature of the training algorithm, and the behavioral task. In this research, competitive games were played, so success or failure was determined by winning or losing games. In each generation, a tournament of games involving all the individuals in the population was conducted. Each individual played two games against one other member of the population (the opponent). The opponent can be selected in various ways. For example, the opponent could be selected at random from the population. The possible outcomes of these games incurred different levels of fitness and are summarized in Table 5.1 below.

Table 5.1. Fitness points awarded by the aggregate success/failure mode, F_{mode_2}, for pairs of reciprocal games during a generational tournament.

Game Pair Outcomes	Fitness Points Awarded
win-win	**3**
win-draw	**1**
win-lose	**.5**
draw-draw	0 (F_{mode_1} dominates)
draw-lose	0 (F_{mode_1} dominates)
lose-lose	0 (F_{mode_1} dominates)

Note that in cases where no win occurs F_{mode_1} is used to determine a negative fitness value.

This function operates in the context of a 50% selection and replacement evolutionary algorithm (EA) (see Chapter 4, Section 5.3.2). Each controller receiving a fitness ranking in the fittest 50% of the population produces a single mutated offspring that replaces one individual in the least fit 50% of the population. An important

ramification of this is that in the case that 50% or more of the population receives a positive fitness value, then selection will be based entirely on success/failure information and the minimal competence mode will have no bearing, i.e. all individuals not achieving success will be eliminated.

5.3 Evolution Conditions

5.3.1 Incremental Verses All-in-one Evolution

In this section and the next, experimental evolutionary conditions used to evolve four populations of robot controllers are discussed. In particular, "all-in-one" evolution was compared to environmental-incremental evolution. Additionally, two forms of tournaments were used. These were 1) tournaments in which every controller played against the same competitor throughout the tournament, and 2) tournaments in which every controller played against a different randomly selected controller.

Incremental evolution in ER can involve incremental fitness selection metrics, incremental environment difficulty, or a combination of both. Although the use of incremental training fitness selection functions has been shown to produce functional controllers for moderately complex tasks [39][40][24][55][38], in most cases the process of incremental fitness function specification has required detailed knowledge of the dynamics of the task to be learned (or evolved). Such functions curtail the controller search space to the degree that resulting evolved controllers cannot be considered to have evolved truly novel controller strategies. Rather, resulting

controllers represent an optimization of a control strategy formulated by the designer of the incremental fitness function, at least to a large degree.

The case of environmental-incremental evolution is not so clear-cut. While it is true that the designer must use some knowledge about the dynamics of the desired behavior to construct environments of increasing difficultly, this does not necessarily inject globally restrictive human bias into the controller search space.

To differentiate the effects of "all-in-one" evolution from environmental-incremental evolution four populations of robot controllers were evolved to play the game *Capture the Flag*. Two populations (population 1 and population 2) were evolved entirely in a very challenging environment with many walls and corridors. These "all-in-one" populations were evolved entirely in World #7, in Figure 5.1 below. Two other populations (population 3 and population 4) were evolved in a set of worlds of incremental difficulty. These environments ranged from a simple environment with no dividing walls, to a vary challenging environment of larger size with many walls and corridors. The sequence of environments is shown in Figure 5.1.

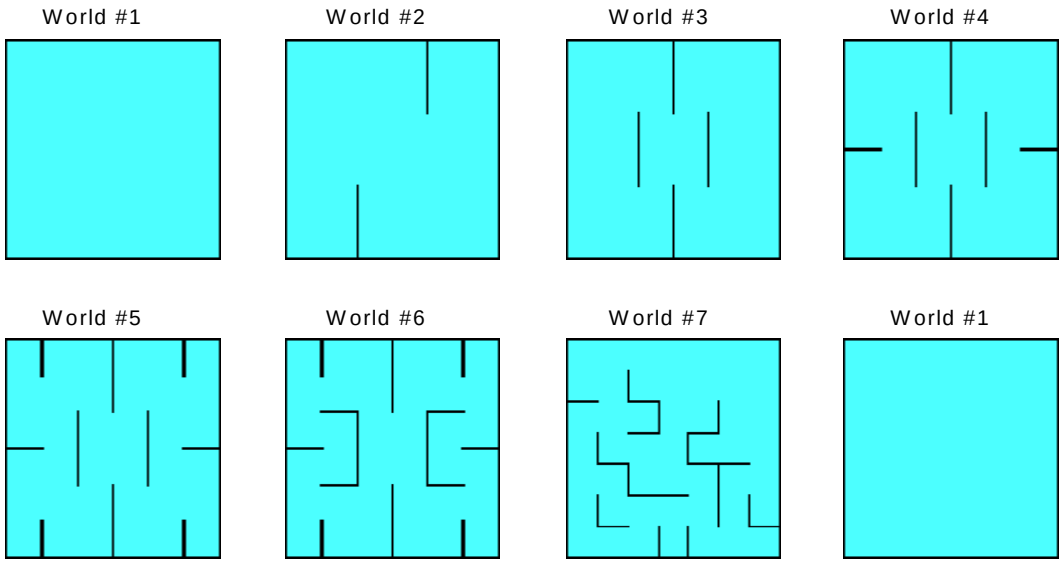


Figure 5.1. The sequence of maze world configurations used for the evolution of populations utilizing environmental-incremental evolution. "All-in-one" evolutions were performed entirely in world # 7.

The environment incrementation process was automated and based on a measure of whole-population fitness: this was the frequency of wins over the course of a single tournament. It should be noted that this metric was used only to increment the environment complexity and was not directly involved with selection of individuals or propagation of the population at any generation.

Selection in the incremental and non-incremental cases was performed using the bimodal fitness selection function described in Section 5.2. All parameters related to initialization, selection, mutation and propagation were the same in all four populations.

5.3.2 Tournament Organization

One tournament of games was played every generation. The results of that tournament were used to calculate the relative fitnesses of each of the controllers in the evolving population using Equation 5.1. At the beginning of each tournament, a set of initial positions for all the robots on both teams, and the stationary goal objects was randomly generated. That set of initial positions was then used for every game in the current tournament. This was done to limit selection resulting from wins due to "lucky" starting positions (as opposed to evolved skills). In each tournament, each member of the population played only two games for scores. For those two games, an opponent was selected from the current population. In the four evolved populations discussed in this chapter, two different types of opponent selection were use. In two of the evolved populations, an opponent was selected from the current population at random, and that same opponent was used in every game throughout the tournament. This case will be referred to as the "constant opponent" case. In the other two evolved populations, a new opponent was randomly selected for each game. That case will be referred to as the "random opponent" case.

5.3.3 Genetic Algorithm and Population Settings

The various permutations of incremental vs. non-incremental and single constant opponent vs. random selection of opponents lead to four evolutionary conditions. These are 1) Evolution in a single difficult world with a single constant opponent being used through a tournament, 2) Evolution in a single difficult world but with a new opponent being randomly selected for each of the games in a tournament, 3)

Evolution in worlds of incremental difficult with a single constant opponent being used through a tournament, and 4) Evolution in worlds of incremental difficulty with a new opponent being randomly selected for each of the games in a tournament. Populations were evolved under each of these conditions.

In each evolving population, all parameters and conditions except those listed above were kept constant. Each evolution was initialized with the same random seed population of neural controllers. Populations were of constant size 40 throughout evolution. Table 5.2 contains a list of other important evolution settings and parameters.

Table 5.2. Parameter settings common to all of the evolved populations.

Parameter	**Setting**
Population Size	40
Sensor range	60
Sensor inputs neurons	150
Initial Internal neurons	60
Chance of adding or removing a single neuron (during network mutation)	70%
Weight initialization range	[-1 1], linear distribution
Weight mutation magnitude range	[-1 1], linear distribution
Weight mutation rate	25%
Initial feedforward connectivity level	60%
Initial feedback connectivity level	20%
Chance of adding or removing a single connection (during network mutation)	70%
N-elitism level (per generation)	Single best from previous generation
Population replacement rate (per generation)	1/2 + 1/40

Each population was evolved to 450 generations. Populations were evolved in simulation. Each evolution required approximately 3 to 4 weeks of computation time on a 1.4 GHz Pentium 3 Processor. Approximately 80% of the computation time was spent simulating the vision based sensor inputs. All other aspects of the simulation, including neural network calculation, population selection and mutation, and simulation of kinetics and physical interactions were accomplished in the remaining 20% of the computation time. All random numbers were generated using the MATLAB 5.3 random number generator. A single random seed was used to initialize each evolutionary run. Because the amount of computation time required to evolve each population was quite large, only one random seed population was investigated.

All robots on a team have homogeneous controllers. This means that one network from the population is copied onto all of the robots on a team. This team then competes against another team containing copies of another controller network. This does not preclude cooperative behavior, because each robot occupies a different position and receives different sensor values.

The current chapter and the next chapter present results generated primarily using the simulation environment. Note that controllers from the fully evolved populations were tested using teams of real robots in a physical maze environment to verify controller performance qualitatively. Those results are discussed at the end of this chapter.

5.4 Evolution of controller populations

In this section, training data collected over the course of evolution are presented for each of the four evolutionary conditions described in the previous section. These conditions are summarized in Table 5.3 below. Note that the words "evolution" and "training" are used interchangeable in this chapter. This use is representative of the evolutionary process being used as a method to implement reinforcement training or learning in a population of neural network controllers.

Table 5.3. Summary of four evolution conditions.

Population Name	Evolution Environments	Tournament format	Training Data Figure
Population 1	Single world	Constant opponent	Figure 5.2
Population 2	Single world	Random opponent	Figure 5.3
Population 3	Incremental worlds	Constant opponent	Figure 5.4
Population 4	Incremental worlds	Random opponent	Figure 5.5

The data for the comparative evolution conditions are shown in four consecutive figures, each of which is of the same format (Figure 5.2, Figure 5.3, Figure 5.4, Figure 5.5). Each figure contains three panels. In every panel of all four figures, generation or epoch number is shown along the x-axis. The top panel in each figure shows relative fitness values used for selection as measured by the bimodal fitness selection function. The y-axis indicates fitness's for the best, the average, and the least fit controller network of the current population. Note that the top panel plots the population best, average and worst fitness curves together on the same axis. The fitness selection metric generates a relative ranking of individual controllers in a population, rather than an absolute one. Because of this, the absolute quality of

evolved controllers cannot be determined from the reported relative fitness of the best individual. As soon as controllers are able to win games, it is possible that one of the controllers will be able to win its games in competition with other individuals in that population and receive the highest possible tournament score. This indicates that the winning controllers are more competent relative to the losing controllers in a given population, but does not indicate absolute competence. In order to demonstrate improvements in overall population fitness, a second purely passive fitness measure was used and is described in the next paragraph. Later, in Chapter 6, a post evolution absolute fitness metric involving competition against controllers of know abilities will be discussed.

The center panels of Figures 5.2 to 5.5 indicate the total number of wins achieved by all members of a population during a particular generation. For the tournament structure used in this research, and with a population size of 40, the maximum possible number of wins per tournament is 80 since each controller plays two games for a score every tournament. The metric is purely passive in the cases of populations 1 and 2 and has no effect on fitness selection whatsoever. In the evolution of populations 3 and 4, the world difficulty was augmented when the population was able to win at least 18 wins. This threshold level of wins is indicated on each plot by a horizontal dashed line. This threshold also corresponds to the number of wins in a population per tournament at which selection is dominated by the win-lose selection mode of the bimodal fitness function. Again, the wins per tournament metric is not used in fitness evaluation and selection, but in the environmental-incremental

evolution cases, it did have an effect on the course of evolution because it was involved in the automatic incrementation of training world difficulty. In the second panel of each of the four figures a rolling average is plotted along with the unaveraged data. The averaged values are used for the automatic incrementation of training world difficulty in populations 3 and 4.

The world difficulty level is shown in the third panel of each of Figures 5.2 to 5.5. In populations 1 and 2, the world difficulty level remains at 7 throughout the evolution (the 'all-in-one' cases). The world difficulty levels correspond to the world numbers in Figure 5.1. Populations 3 and 4 cycled through the set of incremental world several times during the course of evolution.

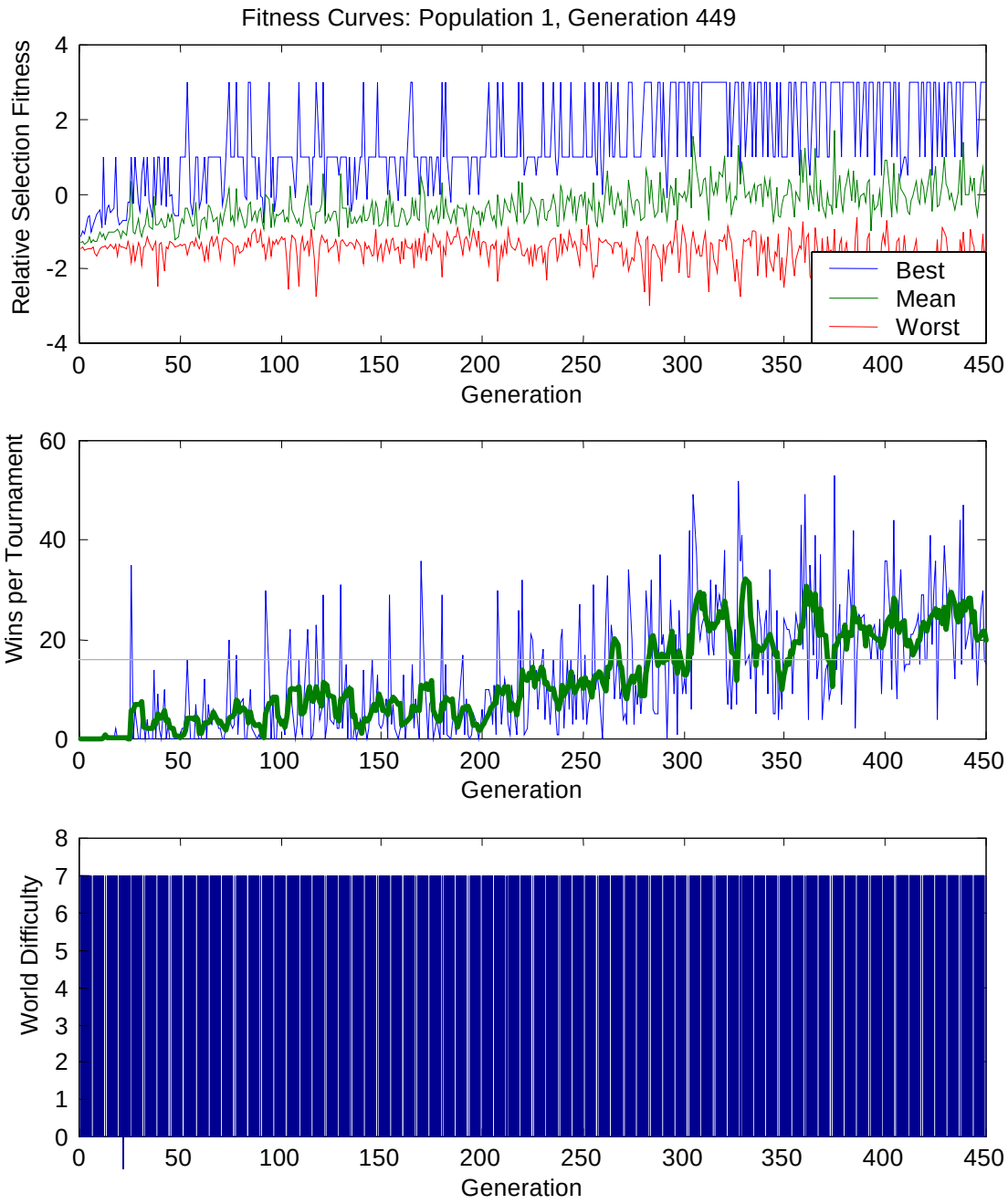


Figure 5.2. Training data from Population 1: Evolved in a single difficult world and using a single constant opponent for all games in a tournament.

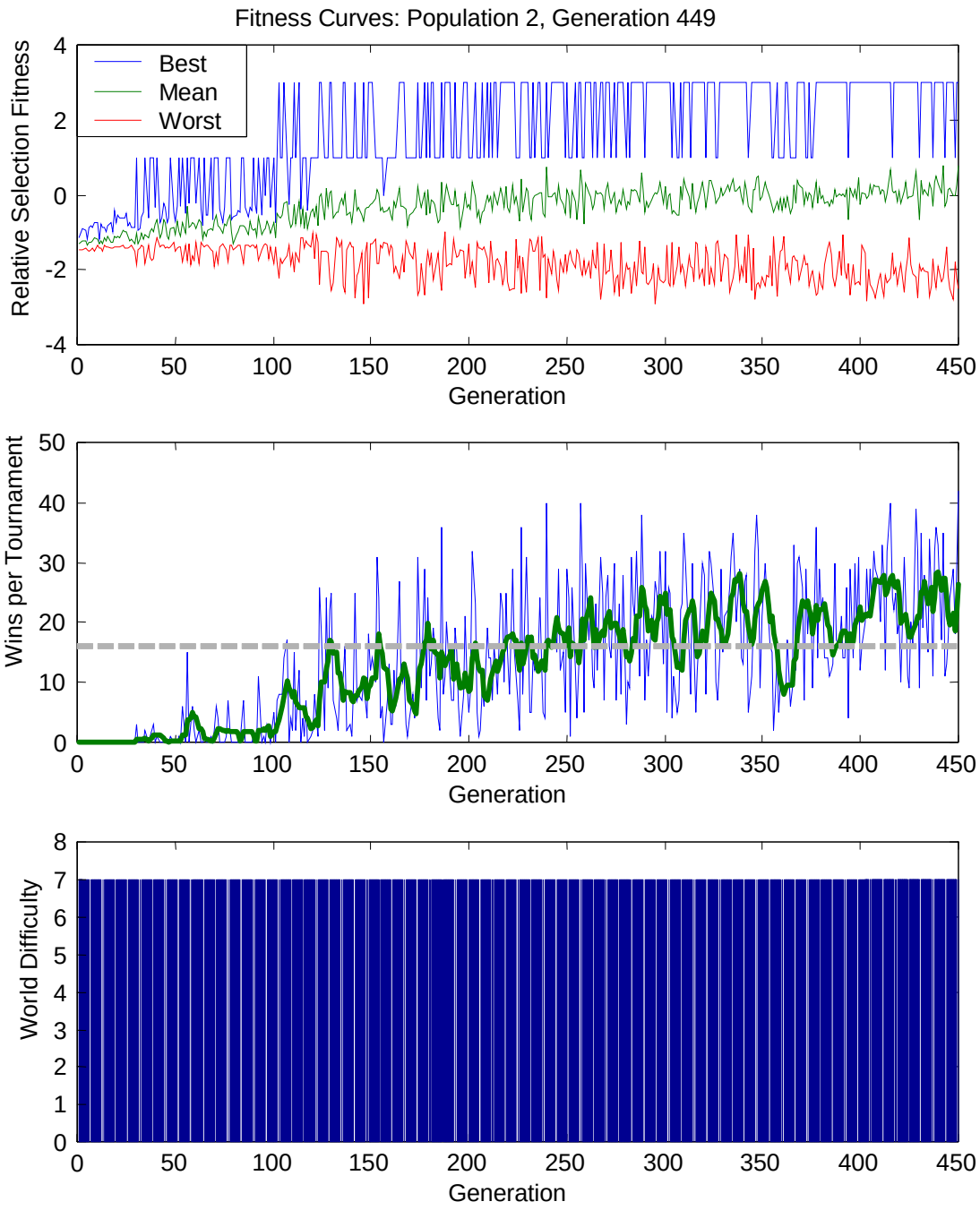


Figure 5.3. Training data from Population 2: Evolved in a single difficult world and using random opponent selection for each game in a tournament.

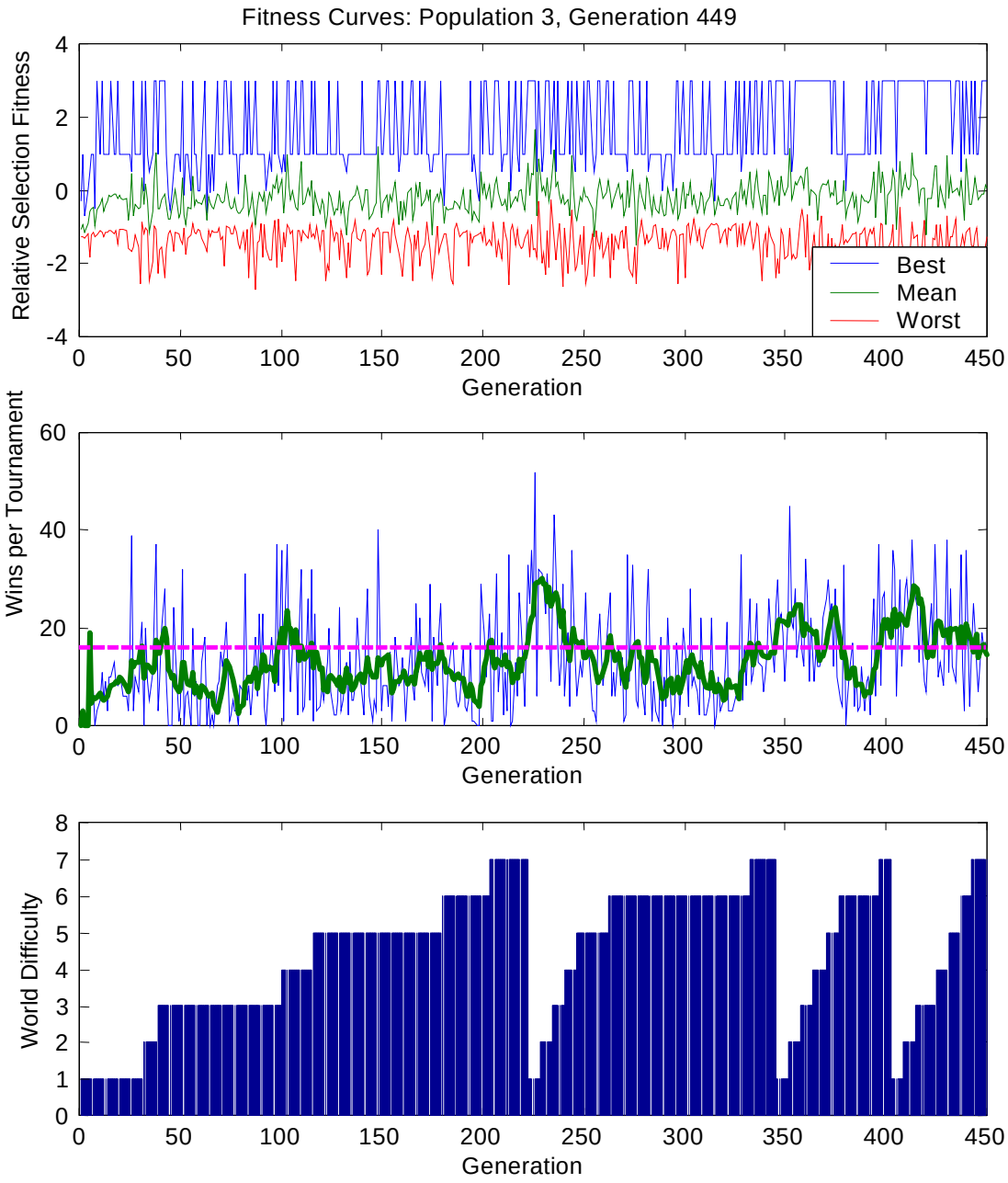


Figure 5.4. Training data from Population 3: Evolved in incremental worlds and using a single constant opponent for all games in a tournament.

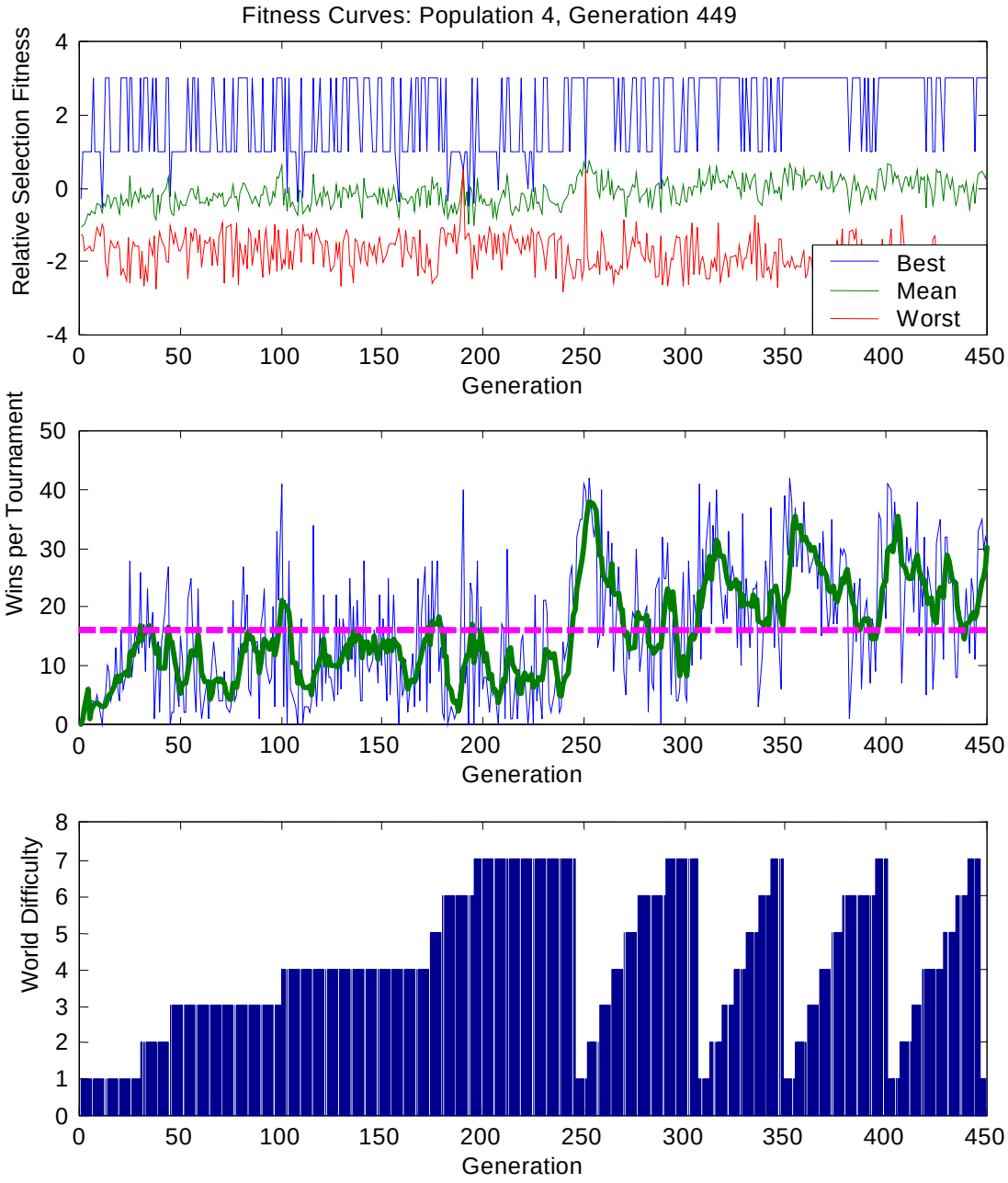


Figure 5.5. Training data from Population 4: Evolved in incremental worlds and using random opponent selection for each game in a tournament.

5.5 Discussion of evolved populations

First, we will discus the "all-in-one" cases. Figures 5.2 and 5.3 show training fitness data from the evolutions performed completely in a single difficult world (world #7 from figure 5.1). In the top panel of both figures, the best controller fitness (as measured by the bimodal fitness function) was not maximized until after generation 50 in population 1, and generation 100 in population 2. This reflects the fact that controllers in populations 1 and 2 were evolved from the beginning in the most difficult world, and no controller capable of winning both of its games in a tournament arose before the 50^{th} (100^{th}) generation. Before the 50^{th} (100^{th}) generation the fitness of the best (fittest) controller in each evolving population can be used to demonstrate improvement in populations even though the selection is based on relative rather than absolute fitness.

Once the relative selection metric has been maximized in one or more individuals in a population, it no longer demonstrates increasing competence in the population as a whole. This reflects the relative competitive nature of the metric. The relative competitive fitness function continues to provide a ranking of individuals within the population. Hence selective pressure will continue to be generated and the performance in the population can continue to improve. In any tournament (generation), a controller wining two games in a tournament will receive a relative maximum score. A fit individual that wins a game in one generation's tournament is not however guarantied a win in the next tournament. A better controller might arise in the next generation or the initial conditions used in the games of the next

tournament may not favor controllers that won in the previous generation's tournament. Only controllers (or their offspring) that consistently win games over many generations will be retained and further evolved.

In order to demonstrate improvement in the population as a whole, the passive performance metric shown in the second panel of figures 5.2 and 5.3 was used. This measures the net number of games won during a tournament. Both figures are similar, so only 5.3 will be discussed. No wins occurred before the 25^{th} generation of evolution. During this period, 100% of the selection was due to the initial mode of the bimodal fitness function. With no wins occurring during the first 25 tournaments (2000 games) it is very unlikely that the second aggregate win/lose selection mode alone could have produced sufficient selective pressure to evolve the population from its randomly initialized seed. From the 25^{th} to the 300^{th} generation, there was a slow incremental increase in the number of wins per tournament. After the 300^{th} generation, selection was dominated by the win/lose aggregate mode of the bimodal fitness function. This indicates that even though populations evolved entirely in difficult worlds initially win very few games, they eventually outgrow the initial human-biased mode of the bimodal fitness selection function.

Figures 5.4 and 5.5 show data from the environmental-incremental evolution runs. Here, as in the "all-in-one' cases, both figures are qualitatively similar so only Figure 5.5 will be discussed. In Figure 5.5 the best fitness at each generation was maximized very early in evolution. This indicates that the best individuals in the population

evolved relatively quickly to be able to win games in the simpler worlds. Even so, panel 2 of Figure 5.5 indicates that at the first generation, no controller in the in the initial random seed population was capable of winning a game. By the 25th generation, the population as a whole was capable of winning enough games during a tournament to graduate to the next level of training world difficulty. This is indicated in panel 3 of figure 5.5 as a step in the plot from 1 to 2. Over the course of training up to the 250th generation, the population became competent in each of the progressively more challenging environments. This is indicated in panel 3. The cycle of training worlds repeats after it is completed. Panel 3 of figure 5.5 indicates that the first cycle through the training world sequence required 250 generations. There after, the population cycled though the sequence nearly as quickly as possible, completing four full cycles between the 250th 450th generations. I should be noted that evolving populations are required to spend a minimum number of generations in each environment so that the rolling average of wins per tournament integrates results from the current training world only.

The most prominent difference between the "all-in-one" and environmental-incremental evolutions is seen in the wins per tournament data. The "all-in-one" cases show a slow steady rise over the course of several hundred generations. The environmental-incremental cases show a series of fairly rapid increases and sudden falls in number of wins per tournament. These rises and falls correspond to the incrementation of training world difficulty. In spite of these differences over the course of evolution, all four populations seem to have achieved similar levels of

performance at 450 generations. In particular, at generation 445, each population is currently evolving in the most difficult world (world #7 from Figure 5.1). In each case, populations are capable of generating approximately 20 wins per tournament.

It should be pointed out that the wins-per-tournament metric is not a true absolute measure of population fitness either. In time, the number of wins per tournament would also come to a limit. In this research the absolute bound on number of wins is dictated by the tournament structure to be 80. This is unlikely to be achieved no matter how fit the controllers are. The measure loses is absolute meaning as populations become fitter. To make this clearer, consider a population in which no controller is able to win a game over the course of a tournament. Such a population is likely to be less evolved (less fit) than a population that produces 5 wins over the course of a tournament. On the other hand, after a population can achieve between 20 and 30 wins per tournament, further changes in number of wins may not be correlated to absolute fitness at all. It is conceivable that the number of wins could go down even while the absolute fitnesses of individuals in a population are increasing. For instance, controllers could develop better defending skills while maintaining a constant level of offensive skills.

In chapter 6, absolute controller fitness will be addressed in detail. Chapter 6 employs extensive competitions of evolved neural controllers with knowledge-based controllers of known abilities and represents a main portion of experimental work associated with this research.

5.6 Demonstration and Discussion of Evolved Controller Behaviors

In this section of the chapter we present the results of several games played with teams of robots using fully evolved controllers. These are presented to qualitatively demonstrate evolved controller behaviors. We will focus on controllers from population 4 discussed in the previous section. Although all of the populations seem to produce similar fitness levels after 450 generations, the next chapter will demonstrate that population 4 is measurably better than the others, although not to a dramatic degree. It is for this reason that we focus on population 4 here.

In Figure 5.6 a robotic *Capture the Flag* game generated in a simulated environment is shown. Fully evolved neural network controllers from generation 450 of population 4 were used to controller all of the robots. In the figure, the smaller dots with the fan-like graphics are the robots. The fan-like graphics display sensor data and are not physical objects. The paths taken by the robots during the simulation are indicated by the irregular curves. There are two robots on each team to make a total of four robots in the simulation. The larger dots represent the stationary goal objects. The heavy black line segments represent walls in the environment. The red team robots were controlled by copies of network #1 and the green team robots by copies of network #2. In this case, the game was won by the red team (paths indicated by the dark lines). The red robot that won the game for its team has made its way to the green goal object in the upper left corner of Figure 5.6.

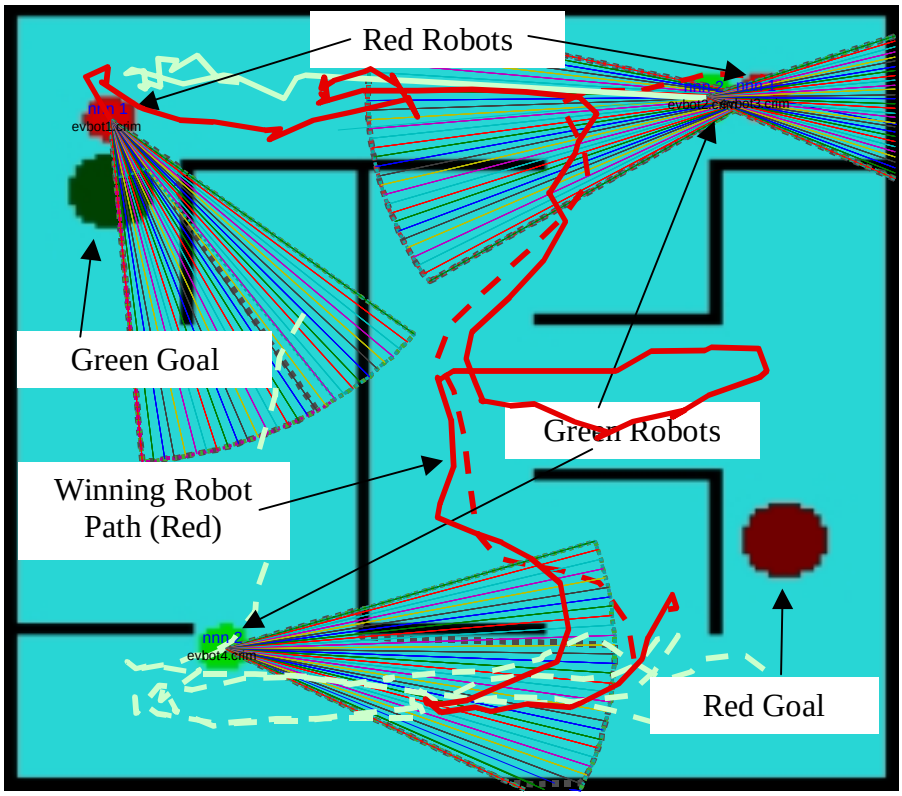


Figure 5.6. Evolved controllers playing robotic *Capture the Flag* in a simulated world. The smaller filled colored circles are the robots. The fan-like graphics are representations of robot sensor data. The larger filled colored circles are the stationary goal objects. The paths taken by the robots during the simulation are indicated by colored curves. Here, the path taken by the winning robot is shown by the dark (red) line. The light colored lines show the paths taken by the green-team robots

As was the case during evolution, all robots on a team have homogeneous controllers. This does not preclude cooperative or differential behavior, because each robot occupies a different position and receives different sensor values.

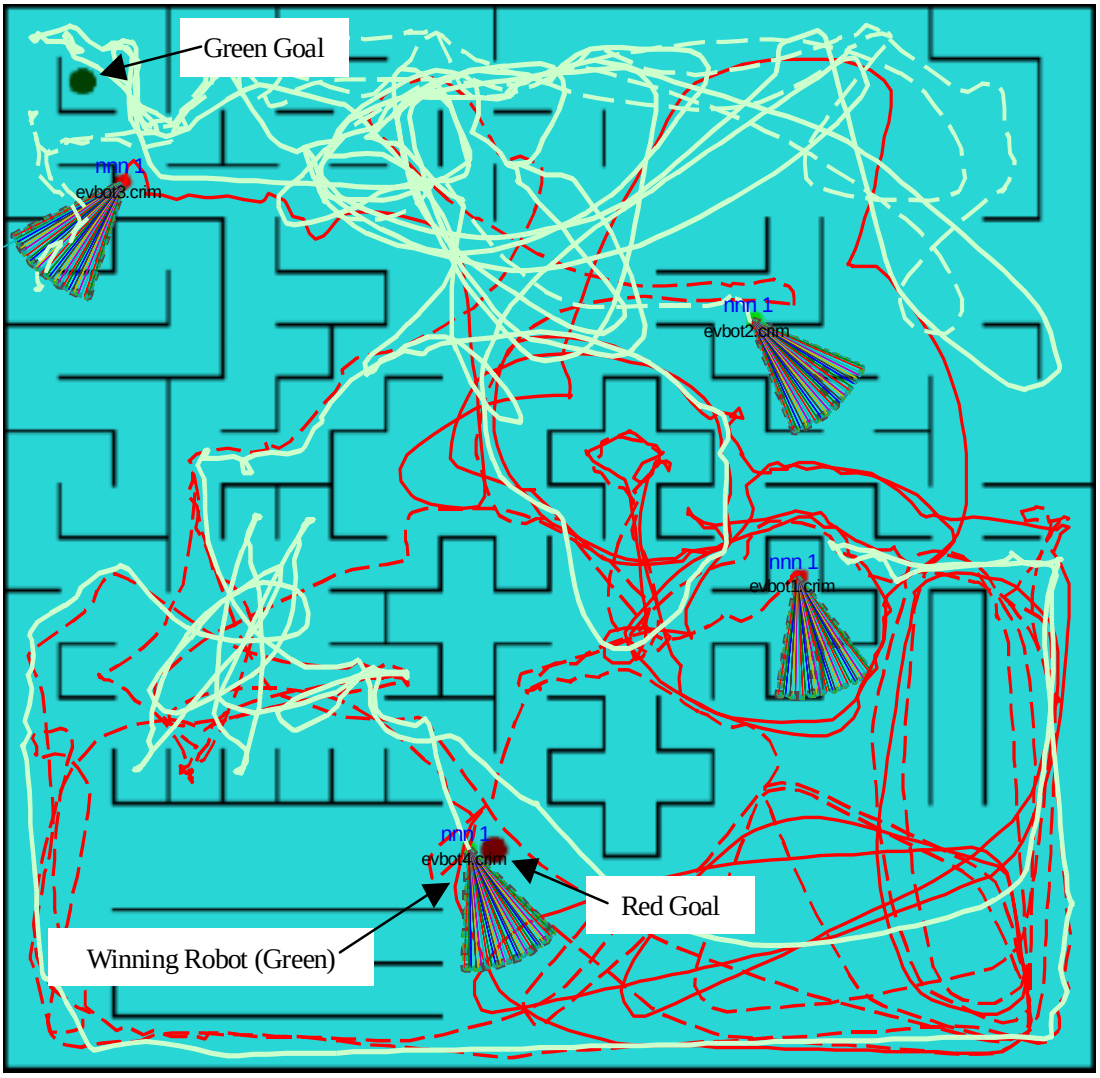


Figure 5.7. Controllers evolved in population 4 competing in a very large complicated world. All robots are using the best controllers from population 4. The game was won by the green team. The solid light line indicates the path taken by the green robot that eventually located the red goal.

Figure 5.7 shows the results of another simulated game in a very large complicated environment. The figure demonstrates that evolved controller behaviors generalize to novel environments. This is a particularly dramatic example. Robots are able to progressively search a novel environment. Here the world used is many times larger than the largest world seen by the controllers during evolution (world #7 in Figure

5.1). In addition, the duration of the game is very long compared to game lengths seen during evolution. The game required 1073 moves to complete. The maximum number of time steps allowed during training was 220 steps (moves). In Figure 5.7, the game was eventually won by the green team. Robots moved extensively through their environment.

Robots being controlled by the best neural network from population 4 very rarely collide with objects. A collision can result in the immobilization of the robot. Two of the robots (one from each team) eventually became stuck during the game shown in Figure 5.7. One robot on the green team collided with an object and became permanently immobilized near the 400^{th} time step. Similarly, one robot from the red team became permanently immobilized after the 700^{th} time step. The other two robots continued to travel about the environment for the duration of the game.

Robots using controllers from this population have evolved limited abilities to extricate them selves from immobilization situations (being stuck). In some instances, robots back up after some time if they detect a very close wall object. No noise was injected into the simulated robot processed vision sensor inputs. Hence, if robots remain immobile for more than one time step, their sensor inputs will be exactly repeated from the previous time step. In that case, controllers must use information from the past to generate sequences of commands that might allow them to escape. These robots clearly demonstrate this ability in the simulated environments. This demonstrates that evolved controllers are not purely reactive. In

fact, on occasion controllers were observed to remain immobile for many time steps (30 or more) and then to back up and spin around. Even so, most of the observed evolved behaviors were reactive. The neural networks can evolve to produce temporal behaviors, but purely reactive near-optimal solutions may exist for this game. A very complex strategy is likely to be only marginally more effective than the behaviors displayed by the evolved controllers in Figures 5.7.1 and 5.7.2. This will be investigated in the next chapter, where evolved neural controllers are competed in extensive tournaments against knowledge-based controllers with well-defined behaviors.

A close examination of controllers outputs revealed that actuator commands stabilized relatively quickly, but do not reach a constant steady state. In a simple experiment, the controller discussed above was repeated fed identical sensor inputs. The observed output did not reach an exact steady state even after 30 time steps (data not shown). Even so, most observed controller behaviors were effectively reactive.

An additional experiment was performed in simulation to demonstrate that controllers evolved progressively over many generations. A set of simulated games was played using controllers from different generations selected over the course of evolution. Again population 4 was used. Figure 5.8 shows eight such simulated games. Each of the eight games was initialized from the same robot and goal starting configuration. Homogeneous controllers were used in each game. The games were allowed to proceed for 400 time steps before being considered a draw and being terminated. In

the first game (panel (a)), the original progenitor controller from the initial un-evolved neural network population was used. In the remaining games, the best controller from each generation being tested was selected and copied into the competing robots. In the second game, the best controller from generation 50 was used. In the subsequent panels (c) to (h), 100 generations progressively separated the "ages" of the populations.

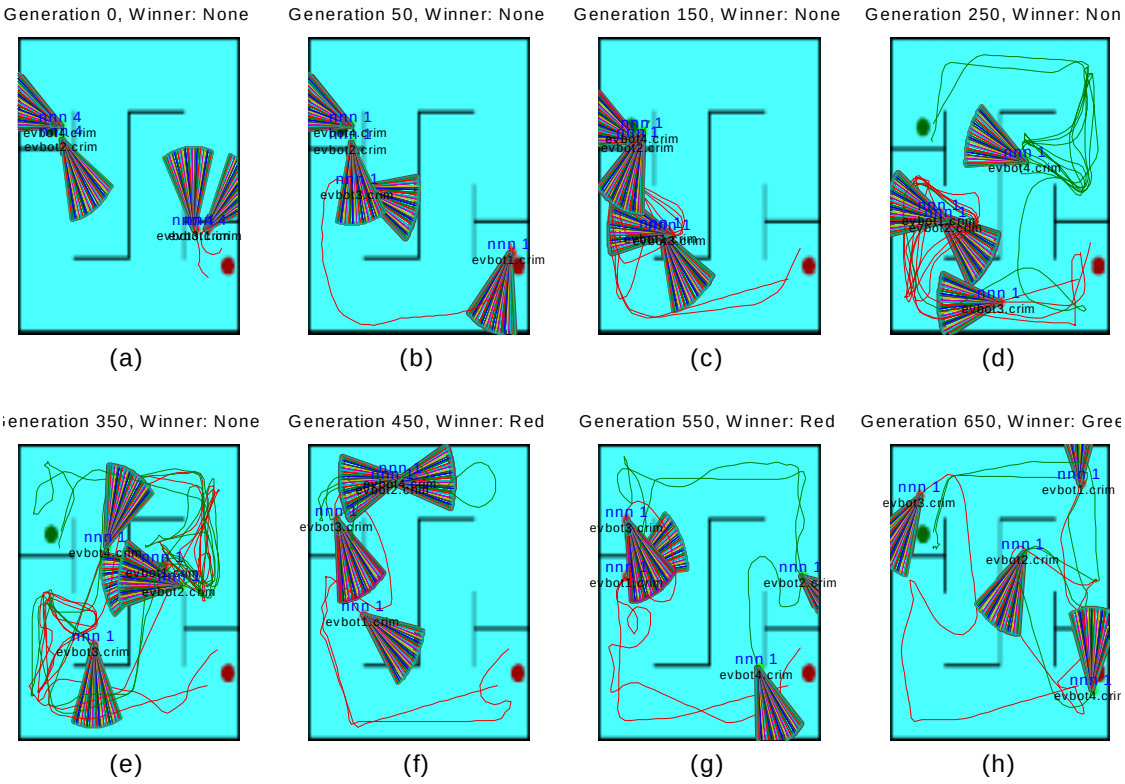


Figure 5.8. A sequence of games played with controllers from sequential generations of population 4. The same random initial positions for robots and goals were used in each game. Robots show increasing levels of performance over the course of evolution.

Recall that population 4 was generated using environmental-incremental evolution where the controllers were evolved within increasingly complex environments. The

first 4 games shown in Figure 5.8 cover the first 250 generations of evolution of population 4. These corresponds to the period of evolution devoted to the first full cycle through the set of incremental training worlds. The training data reported in Figure 5.5 indicate that the number of games won by the population as a whole during a generation (a tournament of games) over this period of evolution remained at or below the threshold at which selection incorporated information generated by the first mode of the bimodal fitness selection function. Over this period of evolution, controllers evolved increasingly better navigation skills. The un-evolved controllers in panel (a) collide with walls almost immediately. At the 50^{th} generation, one of the robots is able to make a fair amount of progress through the environment before becoming stuck. Generations 150, 250, and 350 show increasing levels of wall avoidance skills but robots are unable to win games. At the 250^{th} and 350^{th} generations (panel (d) and (e)), robots are able to travel indefinitely without getting stuck. After 400 time steps at the termination of the games, robots are still traveling, but none of them has been able find a goal object and win. The training data reported in Figure 5.5 indicate that almost all selection after the 350^{th} generation used the second win/lose selection mode. It is during 450^{th}, 550^{th}, and 650^{th} generations that controllers have evolved to be able to win games. The final three games of Figure 5.8 all terminate with wins for one or the other team. In the last three panels, robots appeared to be executing a "left hand mouse rule" search strategy in conjunction with object avoidance. In contrast, at generations 250 and 350 robots seem to avoid walls well and even to make some headway exploring the maze, but do not find their opponent's goals. It is likely that the initial mode of the selection function is

generating little selective pressure in the best robots of generations 250 and 350 because it has been maximized. These data suggest that the second purely aggregate success/failure mode of the fitness function does generate selective pressure for the evolution of simple strategies above and beyond the traveling behavior selected for by the first mode.

Table 5.4. Qualitative acquisition of behaviors over the course of evolution of population 4. The solid dots indicate that a behavior is observed in that generation. The open dots indicate that the behavior has been superseded by another.

Generation	Rudimentary wall avoidance	Effective simple navigation: (always turn left)	Avoid own goal	Qualitatively complex navigation: (left and right turns)	Left-hand mouse search strategy
0					
50	●				
150	●	●			
250	○	●	●		
350	○	○	●	●	
450	○	○	●	●	●
550	○	○	●	●	●
650	○	○	●	●	●

Table 5.4 summarizes the discussion of acquisition of behaviors over the course of evolution. The identification of a particular behavior here represents a qualitative human assessment based on observation of robots during game sequences. In the later generations, exact behavior is very difficult to predict, and behaviors observed from the distal (exterior to the robot-controller system) are not necessarily reducible to desecrate behaviors at the proximal level (from the point of view of the robot controller). See [80] for a discussion of the terms proximal and distal. Actual

behaviors at the proximal level are likely to be inextricable co-coupled with one another, and with sensor inputs.

5.7 Transfer of Evolved Controllers to Real Robots

As a final consideration in this chapter, evolved controllers were transferred to real robots and tested a physical maze environment. Further physical verification results are presented in Chapter 6. Results generated with real robots in the physical maze environment are shown here for qualitative comparison to the simulated games above. Figure 5.9 shows the results of a game played with real robots in the physical maze environment. Here, copies of the best network from population 4 controlled all of the robots. It should be noted that robots recognize their teammates and goal as a function of the visual sensor system. Rather than seeing red and green, robots see "my color" and "opponent color". The figure shows the last image in a set of images collected over the course of the robot game from an overhead camera. The image sequences were processed to track the robots over the course of game play. The dashed lines indicate the paths taken by the robots. The light lines indicate the paths traveled by the green robots over the course of the game, while the dark lines indicate the paths of the red team robots.

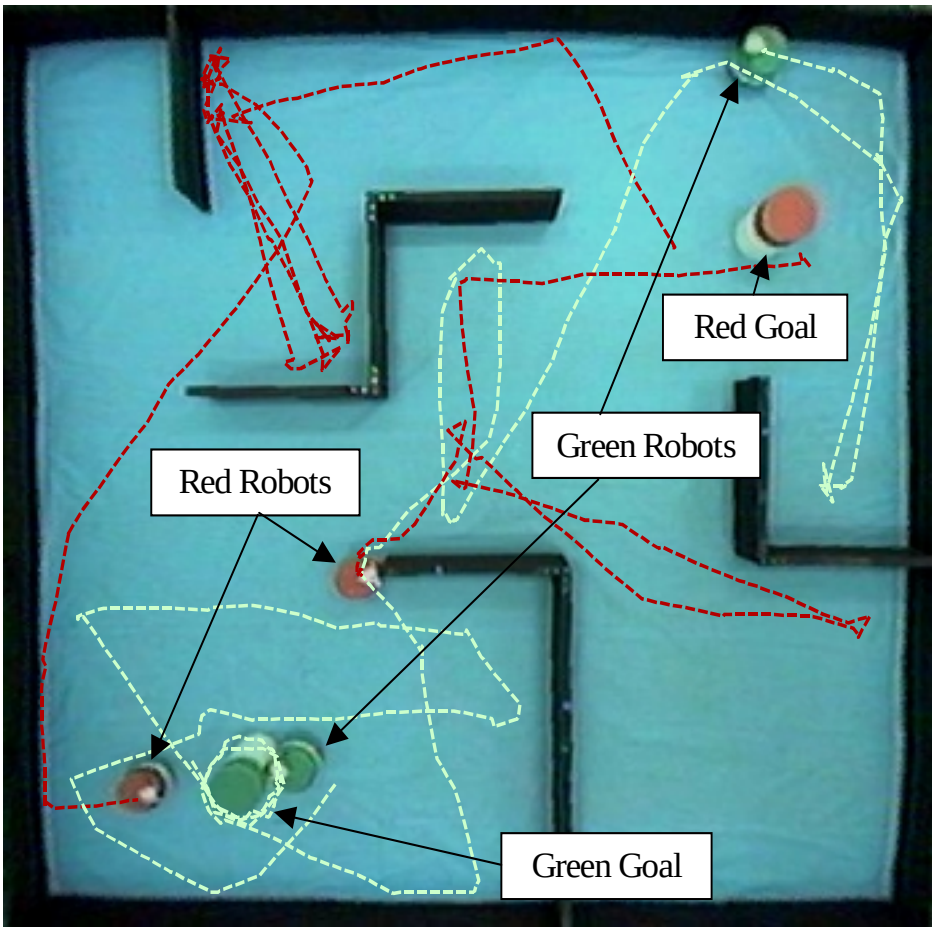


Figure 5.9. An example game involving real robots in the physical maze environment. All robots are controlled by evolved neural networks. The dashed lines indicate the paths taken by the robots during the course of the game. The light lines indicate the paths taken by robots on the green team while the dark lines indicate the paths taken by the red robots. This game was won by the red team.

As in the simulations, robots were controlled with the best performing neural network from population 4. In the real environment robots were able to avoid walls and locate goal objects. In the simulations, any contact friction was modeled at 100%. In the real world, however, robots did slip somewhat when in contact with objects. In some case this benefited robots in the real world because it could allow them to jiggle lose from a physical situation in which they had become immobilized. Even so, robot wall and object avoidance involved very little contact. In most instances, robots would

come very close to walls, but would then back away or execute a turn. As in the simulations, robots could become immobilized by colliding with an object in such a way that the visual sensors could not detect the object. For example, this can occur if a robot heads toward the edge of a corner or wall segment, then misses the edge with its forward facing camera, but hits the edge with a portion of its body that is out of view of its camera. In the simulations, robots learned to avoid such situations by executing precise turns and curves. The temporal nature of the evolved networks allows controllers to respond to objects that have recently been in view, but have passed out of the sensor field. The same turn and curve commands were produced by the real robots in the real world, but sensor and actuator noise can result in a variation of about 15%. In most cases, the controllers would re-compensate on the next time step, but in cases in which the robot had moved so that an important object (such as wall edge) was out of view, a 15% error could cause a collision.

5.8 Conclusion

In this chapter, a bimodal fitness selection metric was developed and applied to the evolution of neural network based controllers for mobile robots engaging in a competitive team game *Capture the Flag*. The fitness selection metric accommodates sub-minimally competent initial controller populations with a hand formulated mode. A second purely aggregate success/failure mode becomes active if controllers are able to complete the overall complex task to a detectable degree. Fitness measurement produced by the second mode supersedes that produced by the first mode so that

controllers able to accomplish the overall task (however poorly) are selected without regard to any human biases used to formulate the initial mode. Both of the selection modes may by active, during a particular generation and throughout evolution and interact dynamically and automatically.

Several populations of robot controllers were evolved under different evolutionary conditions. Relative fitness was recorded over the course of each evolution and was presented in several figures. Those data show related training and evolution dynamics produces by the different evolutionary conditions.

One particular population of evolved controllers was examined by playing test games with the best-evolved member of the population at set generation points over the course of evolution. Eight sequential example games were presented and the development of expression of qualitative behaviors from the distal point of view was discussed. These behaviors included rudimentary navigation, more complex navigation, goal differentiation, and the expression of a left-hand mouse-like search strategy. The exact behaviors of fully evolved robot controllers were dynamic and difficult to exactly characterize.

For nontrivial tasks it may not be practical to identify and characterize behavior. In the next chapter evolved controller behavior will be addressed in terms of ability to compete overall, rather than in terms of expression of individual behaviors.

CHAPTER 6. THE APPLICATION OF METRICS FOR POST EVOLUTION EVALUATION OF EVOLVED CONTROLLERS

The experimental work presented in this chapter focuses on defining and applying absolute metrics for evaluating the performance of evolved robot controllers. In [27] it was noted that co-evolutionary processes do not evolve within the context of a pre-determined or fixed fitness landscape. Hence, training fitness values cannot be used to monitor the progression of training past the initial phases of evolution. This is also true for the single-population competitive selection used to evolve the robot controllers described in Chapter 5. The trends reported in the training fitness plots (the top panels of Figures 5.2-5.7) do not correlate directly to absolute fitness. The fitness of each individual in a population was affected by the fitness's of the other individuals in that population, thus producing a changing fitness landscape with varying, non-absolute fitness values. To clarify this, consider the following example: Suppose, by chance a particular individual was maintained without receiving new mutations in a population over many generations. Since fitness selection is competitive and relative, the selection fitness of the constant individual would appear to go down over the course of many generations because the relative fitnesses of the other individuals in that population would be increasing (assuming the training process had not plateaued).

The passive metrics (i.e. ones not effecting the process of selection) that were used to evaluate the increasing level of performance of controllers in terms of the total number of wins achieved by a population in a generation, were also not entirely absolute. This was especially true as populations evolved higher levels of competence later in evolution. (These were shown in the center panels of Figures 5.2-5.7.)

6.1 A Metric for Post-evolution Evaluation of Controller Performance

To measure absolute performance, evolved controllers were compared to a controller of known abilities. A knowledge-based controller was designed to play the robotic version of *Capture the Flag*. This controller was hand coded and used in competition with the evolved ANN controllers. The following table briefly summarizes the knowledge-base controller's behaviors:

Table 6.1. Behaviors expressed by the hand-coded knowledge-based controller. Behaviors are given in order of precedence.

Precedence	Behavior
1	If enemy goal is detected, robot actively homes in on goal
2	Turns away from own goal
3	Avoids teammate robots by passing on the right
4	Blocks close enemy robots
5	Avoids distant enemy robots
6	Extracts from corners and close walls
7	Avoids walls and follows corridors if no other objects are detected
8	Moves forward in free space
9	Detects immobilization on out-of-view objects and responds with rotations of random degree

The four populations evolved using the methods described in the previous chapter were evaluated in extensive sets of tournaments. After evolution, the "best" performing controller in each population was pitted against the knowledge-based controller in a series of games. Tournaments of 240 games were played using controllers from several uniform generation points from each of the four evolved controller populations. Each tournament used 120 random robot-and-goal position initializations based on the first 120 random numbers generated by MATLAB 5.3 using the initial seed 1. To eliminate possible advantage to one or the other controller (robot team), each initial position was used for a set of two games. In the first game, the ANN controllers controlled the red team (team 1), while the knowledge-based controllers controlled the green team (team 2). Visa versa, in the second game, the knowledge-based controllers controlled the red robots, while the green robots were controlled by the ANN controllers. These tournaments were carried out in three different maze world configurations. These were selected to be of varying difficulty based on size and number of walls and corridors. Figure 6.1 shows the three maze worlds used for testing the evolved controllers in competition.

Evolved controllers were tested in multiple environments so that a more accurate overall evaluation of their capabilities could be made.

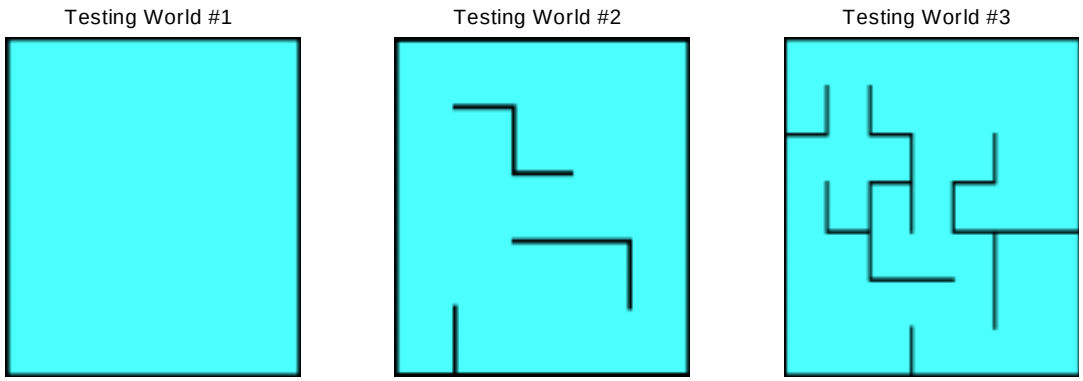


Figure 6.1. Three testing worlds.

Note that Testing Worlds #2 and #3 were not used during evolution of controller populations and represent novel worlds as far as the evolved controllers are concerned. In addition, the knowledge-base controller was not involved in any way with the selection or evolution of the evolved controller populations and its behavior had no effect on evolving controller strategies. In effect, the knowledge-based controller is a novel opponent to the fully evolved controllers.

Competition with knowledge-base controllers will be used extensively in this chapter to compare and evaluate evolved robot controllers and populations. These competitions will also be used to demonstrate increases in absolute performance over the course of training, and to identify when learning plateaus appear in training.

Selection of the "best" individual of an evolved population was done by averaging the net number of wins achieved by each controller over a series of 30 intra-population tournaments (i.e. not involving the knowledge-based controller) in Testing World #3. The controller with the greatest number of wins was selected as the "best". It should

be noted that the set of 30 random initial states used for best controller selection were generated based on sequential random numbers starting with the MATLAB 5.3 seed 31415. Hence, these differed from the set of 120 random initial states used for the paired sets of 240 testing games (in each testing world). This was done to avoid selection of a "best" controller that was fitted to a lucky set of random initial conditions. The process of "best" controller selection was very computationally expensive, and required several days of computation time. It is still possible that the true best controller was not selected, but it is very unlikely that a relatively poor controller in the population would achieve the best ranking over the course of 30 tournaments. This "best" controller selection method was not used to drive selection during evolution for several reasons. The first is of course the large amount of computation time needed. Also, because the GA used a 50% selection and replacement strategy during propagation, it was only necessary for the best controller to fall within the fittest 50% of the population for it to be retained.

The justification for using several testing worlds for evaluating the controllers is that evolved controller strategies that produce effective play in one world might not do so in the next. Simple strategies may be optimal in the simplest world, whereas more advanced strategies may actually fail in the simple world. For instance, a controller that tends to find and follow walls might do poorly in a simple world with no interior walls. A spectrum of tournaments in the several worlds gives a better profile of evolved controller performance.

6.2 Absolute Performance of Evolved Controllers

This section presents data generated in large competitions of 240 games involving teams of robots controlled by the evolved ANN controllers and by the knowledge-based controllers. These sets of games and tournaments constitute a post evolution performance metric. This metric can be used both to evaluate the absolute level of competence of controllers, and to compare the performance of several evolved controllers to one another.

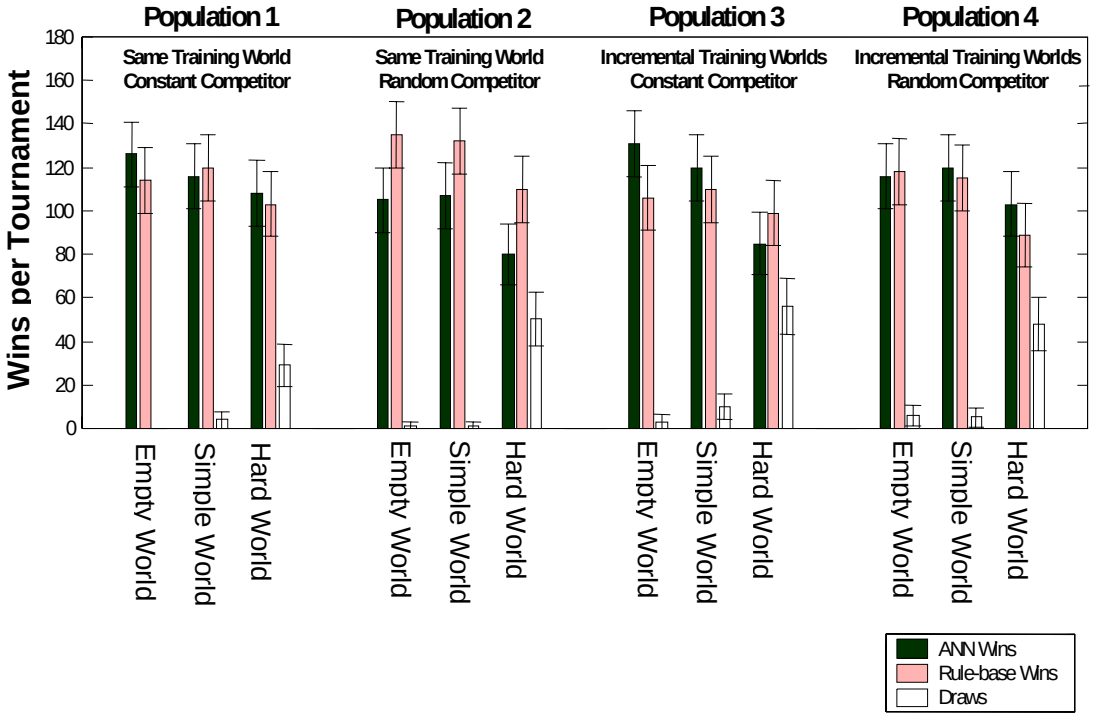


Figure 6.2. Four populations evolved under different conditions were evaluated in competition with the knowledge-based controller. The best individual from each of the populations played 240 games against the knowledge-based controller in each of the three testing worlds. Each sub-triplet of bars gives the results from a single tournament in a single world. The dark bars indicate the number of evolved controller wins in a given tournament, the shaded bars show the number of wins achieved by the knowledge-base (rule base) and the white bars indicate games that ran over the time limit (draws).

Figure 6.2 shows the results of tournaments with evolved ANN controllers competing against knowledge-based controllers. The best-evolved controller from each population was placed in competition against the hand-coded knowledge-based controller in a series of 240 games, and in each of the testing worlds. In all, the results from 12 tournaments are shown in the figure. Each small cluster of three bars in Figure 6.2 gives the results from a single testing tournament of 240 games in a particular testing world. Each super-cluster of three groups of three bars (nine bars total) constitutes the results of three tournaments in the three progressively difficult testing worlds and makes up the total competition data collected for each "best" evolved controller. Progressing from left to right, the first bar of each small group of three individual bars indicates the number of evolved ANN controller wins, the second bar indicates the number of knowledge-based controller wins, and the third bar indicates the number of games in which neither team was able to find their enemy's goal. The intervals ranges shown on each bar indicate a 95% confidence interval for the data [91].

All four controllers from the evolved populations were able to win games in all of the competitions. However, only populations 1 and 4 were able to win more games than the knowledge-based controller over the course of a set of 240 games in the most difficult world (Testing world #3 from Figure 6.1).

At first glance, the data shown in Figure 6.2 indicate that the environmental-incremental evolution produces somewhat superior results. However, population 1, which was evolved entirely in world #7 of Figure 5.1 (the most difficult training

world) was able the win a significant number of games in each of the test worlds and also out-competed the knowledge-based controller, although not to a statistically significant degree. Although the best results were achieved in population 4 using environmental-incremental evolution, these results show that environmental-incremental evolution, is not absolutely required at least in the case of the behaviors evolved in this work. It is speculated that the slightly superior results seen in the incremental case actually reflect maintenance of behavior, more than an improvement in behavior acquisition. This is evidenced by the rapid succession though the set of training worlds that occurred after population 4 completed the first cycle of evolution though the set of training worlds (see Figure 5.5 from Chapter 5).

The most salient result of the data shown in Figure 6.2 is that by the 450th generation, populations in three out of the four evolutionary conditions were able to produce behavior that was competitive with the knowledge-base in all of the testing worlds.

It was hypothesized that random competitor selection during training tournaments would add excess noise to the selection process and result in poorer or slower absolute fitness evolution. This is in fact seen in the absolute fitness's of population 2, as shown in the second super-group of bars 9 in Figure 6.2. There the evolved ANN controller was beaten by a statistically significant number of games by the knowledge-based controller in all of the testing worlds. However, the same effect was not observed when random opponent selection was applied to the environmental-incremental case of population 4. That evolution appeared to produce the fittest "best" controller in the most difficult testing world.

The metrics for absolute controller fitness of Figure 6.2 do give a numerical relationship between these particular four “best” evolved controllers with respect to the knowledge-based controller and to one another. However, those data only weakly indicate differences due to evolutionary conditions. This is because full evolutions starting from random initial controllers were performed once for each of the four conditions. Each evolution was started from an identical seed population, so all observed differences in final evolved “best” controller performances are due to differences introduced during evolution. However, it is likely that stochastic processes dominate (to the extreme) during evolution. Because the resulting ”best” controllers are all fairly competent, repeated evolutions starting with different seeds would be required to evaluate the significance of evolution conditions on resulting populations. What can be said is that incremental and non-incremental conditions both produce competent controllers.

6.3 Measuring Absolute Fitness Over the Course of Evolution

Competition of evolved controllers against a known knowledge-based controller was also used to measure the absolute fitness of evolving populations over the course of evolution. Figure 6.3 shows the progression of acquisition of game playing ability over the course of evolution of population 4. The figure show competitions performed with the best current individual taken every 100 generations starting with generation 50 and ending with generation 650. Each generation was tested in the

three test worlds, and in each case 240 games were played against the knowledge-based controller (for a total of 21 tournaments).

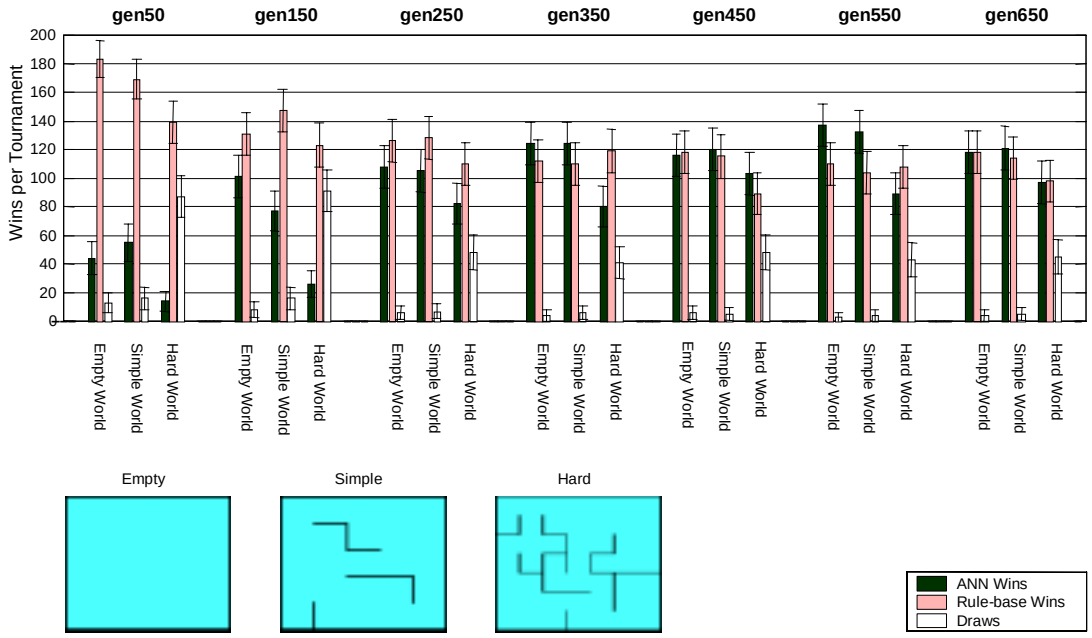


Figure 6.3. Performance of population 4 in competition against the knowledge-based controller at successive generation points during the course of evolution. Each best controller played one tournament in each of the three testing worlds. The testing worlds are repeated from Figure 6.1 in the thumbnail panels in the lower left of the figure.

The sequence of grouped tournament results shows a steady increase in game playing ability. This is especially prominent in the most difficult test world (noted as "Hard" in Figure 6.3, and Testing World #3 in Figure 6.1). At generation 50, the best individual in the evolving population was only able to win 14 games out of 250 while the Knowledge-based controller won 139 games. At generation 450, the evolving controllers were able to play competitively with the Knowledge-based controller, wining 103 games to the Knowledge-based's 89. This steady progression of fitness is not directly apparent in the raw training fitness data from the evolution of population

4 (see Figure 5.5). These results indicate that evolved behaviors were not merely cycling through a set of simple behaviors without achieving absolute improvement as was observed in [27]. In that work the researchers applied competitive fitness to the co-evolution of predator and pray robot behaviors.

Figure 6.3 indicates that a training plateau is reached near the 450th generation. It is in that generation that evolved controllers are first able to play competitively against the knowledge-based controllers in the most difficult testing world. Note that competitive play in the empty world is achieved much earlier by the 150 generation. Further, competitive play is achieved by the 250th generation in the simple world. This indicates that simpler strategies are likely near-optimal in simple environments.

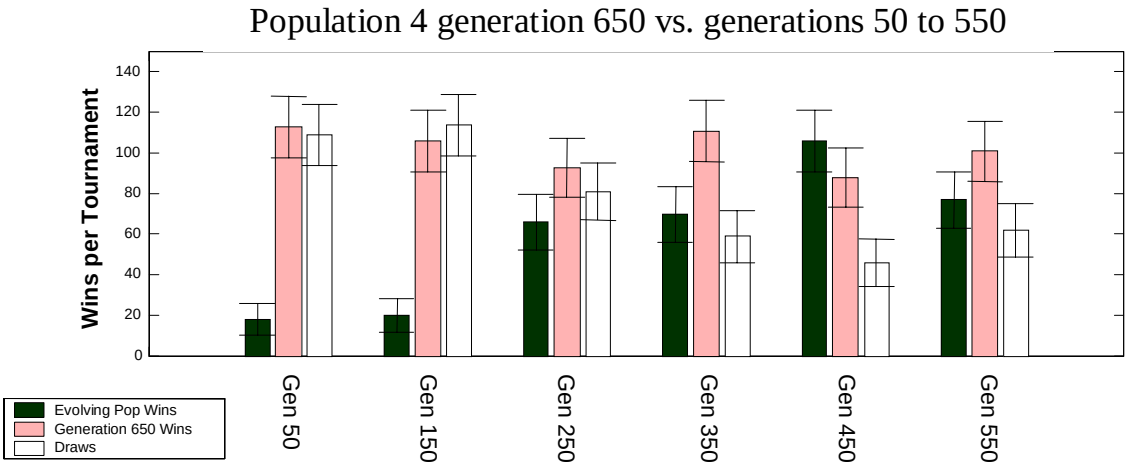
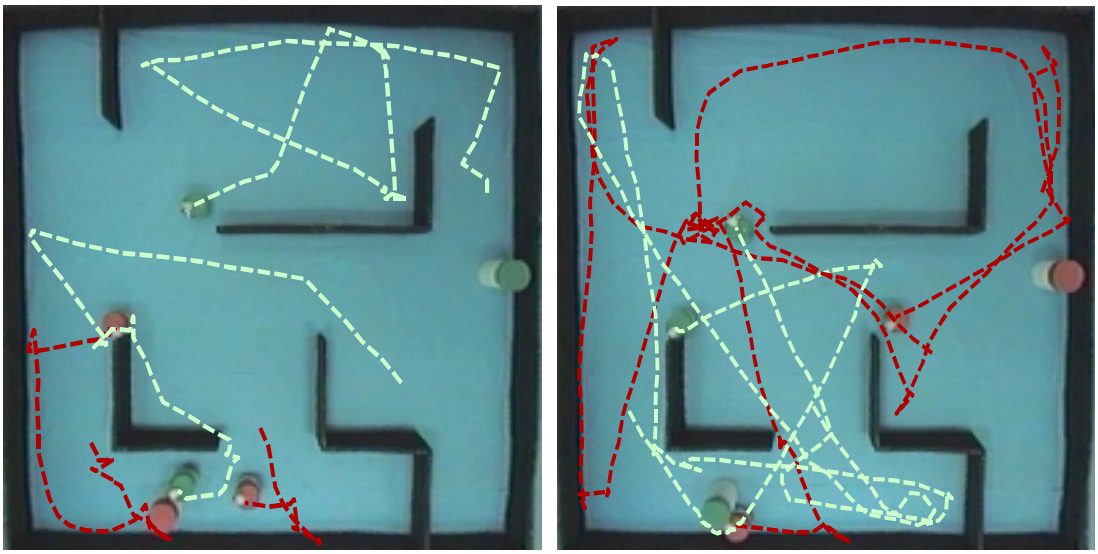


Figure 6.4. Final generation competition results. Earlier generations (50 150 250 350 450 550) of population 4 compete against the final generation (650) of population 4. All 6 tournaments of 240 games were played in the most difficult testing world (world #3 from Figure 6.1).

A further demonstration of the progressive evolution of controller performance, and of eventual plateauing is given by the data reported in Figure 6.4. Here, rather than using the knowledge-based controller, the best-evolved controller of population 4 from its final generation (650) was played in competitions against earlier generations. Here, again, each tournament was made up of 240 games using a fixed set of 120 random initializations. A total of 6 tournaments were played. In the Figure 6.4, all tournaments were played in the most difficult world. Since the goal of the experiment was to demonstrate progressive evolution followed by plateauing within a population, the simpler worlds were not included. In the figure, the dark bars record the number of wins achieved by the evolving controllers at each generation point. The light shaded bars record the number of wins achieved by the best network from generation 650. The data show an increase in the number of wins achieved by the evolving generations up to the 450^{th} generation. The 550^{th} generation appears to be slightly less competent than the 450^{th}, and the 450^{th} seems to be slightly more competent than the 650^{th} generation. However, numbers of wins reported in those two tournaments are within the 95% confidence intervals. These slight oscillations in competence after the 450^{th} generation are also seen in the data presented in Figure 6.3.

6.4 Physical Verification: Transfer and Testing of Evolved Controllers to Real Robots

This section shows results generated using real robots in the physical environment. Figure 6.5 shows the results of two games played with teams of real robots in a physical maze environment. In the games, the best-evolved ANN controller from population 4 and the knowledge-based controller were used. These were transferred to teams of green and red robots respectively. Each team consisted of two robots to make a total of four robots operating in the maze environment at one time. The robots within a team used homogeneous controllers. The evolved ANN controller tested in the real world was the "best" controller from the 450th generation of population 4.



(a) (b)

Figure 6.5. Two example games involving real robots in a physical environment. In each panel, the green robots (light dashed lines) are controlled by evolved neural networks while the red robots (dark dashed lines) are controlled by the knowledge-based controller. The dashed lines indicate the paths taken by each of the robots during the course of each game. The first game was won by the evolved neural network controllers, while the second was won by the knowledge-based controller.

The two games shown in the figure were initialized with reciprocal starting position for robots and goal positions. The first game in panel (a) was won by the neural network controlled robots while the second game was won by the robots using knowledge-base controllers (b). In both games, it was the goal in the lower left portion of the maze that was located by the opponent robots.

Given the length of a single game in the real world, it is impractical to duplicate the tournaments. The results displayed in Figures 6.2 and 6.3 required 7920 separate games. This large number of games was needed to produce results with acceptable 95% confidence intervals. The two game sequences in Figure 6.5 show that behaviors seen in simulation are also displayed in the real world. This remains a qualitative assessment, though. Robots avoid walls with various turns and sequences of backward and rotation steps. The navigation behaviors were not fully characterized. Unlike some of the evolved controllers resulting from earlier incarnations of the ER test-bed, the controllers from population 4 were quite dynamic and their exact responses were quite difficult to predict. Sensor noise (about 15%) in the real world also compounds the problem of behavioral analysis.

Behavioral analysis is not the main focus of this research. Rather, the focus is on the study of methods for the automation of synthesis of behavioral robotics controllers. In the ideal case, this process would be fully automated, and no a priory knowledge of the elements of behavior would be required to be input into the process. The only

required input from the designer would be a measure of aggregate or overall task completion. After a controller has been evolved, it may be of interest to analyze the details of its acquired behaviors, but it is not necessary to its functioning to do so.

The fidelity of transference of behaviors from simulation to reality is another matter. It is important to demonstrate that differences between simulation and reality are small enough so that behaviors evolved in simulation are expressed in the physical world. In order to achieve this, is important to develop ER simulation test-beds in conjunction with physical robot systems. Although it may be possible to develop a simulation environment without a particular real robot system in mind, it is much more difficult than early researchers may have believed. The difficulties stem mainly from conceptual problems rather than technological ones. The design of unrealistic sensors, the incidental use of un-obtainable environmental information, and other similar issues can lead to the development of untransferable controllers [21][65]. For these reasons, we believe that the feasibility of transference is best demonstrated by example. This will likely not be the case in the future, in light of the rapid development of virtual reality, gaming physics engines and the rise of realistic computer animation for entertainment. In the early years of ER research, it was speculated that transference problems could only be overcome by evolving controllers using only real robots [6]. However, and perhaps surprisingly, this did not turn out to be the case. It is in fact well within the current state of the art of computing, simulation, an autonomous robot technology to build finely couple simulation and

hardware platforms. The real problems lie in the most fundamental elements of the process of automatic construction of intelligence.

In the research reported on in this dissertation, the behaviors of the knowledge-based controller were known, and could be observed in the real world and in simulation. Based on qualitative comparisons, the behaviors of the hand-coded controller were observed to be expressed similarly in simulation and reality. As is the case with the evolved controllers, the hand-coded controller could be run on stimulated robot agents, and the physical robots without the need for any modifications. Hence, its qualitative predictability in both simulated and real environments does support the assertion that behaviors do transfer from simulation to reality in this robot research platform.

That said, transference was not perfect. Because of the iterative looping nature of the fundamental EvBot control structure, both the evolved and hand-code controllers re-compensate for slightly mis-calibrated sensors and actuators. These slight mis-calibrations were observed to lead to transference problems in one situation. If a robot controller produces a precisely evolved arch command close to a right-angled corner or wall edge, the robot can move in one time step into a position where the robot is stuck, but in which the robot camera does not see the object the robot is stuck on. It is important to note that this situation is modeled in the simulated world, and that controllers must evolve to avoid it. Evolved controllers were usually successful in avoiding out-of-view corners, but collisions with corners where more frequent in

the physical maze environment. The problem in transference arises when slight differences in the physical results of evolved arch commands cause the robot to hit an out-of-view object, rather than just barely missing it (as it would do in simulation). After such an occurrence, controllers receive no sensor information that would allow them to re-compensate (since they don't see the object). In other situations observed in the real world, evolved controllers were able to avoid objects in ways that were qualitatively similar to that observed in simulation.

6.5 Chapter Summary

This chapter presented quantitative and comparative measurements of evolved controller performance. Evolved controllers were tested in tournaments of games against a hand-coded knowledge-based controller. Testing tournaments were performed in environments of different complexity to provide a better measure of controller performance.

Each of the four evolved populations discussed in the previous chapter was tested. It was found that the fittest controller from population 4 at generation 450 was able to compete best against the knowledge-based controller and could beat it in a small majority of the games played to a win in the most challenging testing environment. Population 4 was evolved under environmental-incremental conditions. Population 1, however, which was evolved entirely in a single very challenging world performed

almost as well as population 4, so no real advantage can be awarded to incremental evolution methods.

The salient result is that populations of ANN controllers can be evolved using the bimodal fitness function of Chapter 4 to perform competitively with this particular knowledge-based controller. The performance of evolved controllers can be measured with respect to a fixed controller of know abilities. These measurements can be used to compare controllers to each other, and to demonstrate incremental improvement over the course of evolution. The knowledge-based controller was not used in any part of the evolution of the ANN controllers. During post evolution, the knowledge-based controller was effectively a novel opponent to the evolved ANN controllers.

CHAPTER 7. CONCLUSIONS AND FUTURE WORK

This chapter contains a brief summary (Section 7.1) of the research presented in the previous chapters. Section 7.2 provides a discussion of important unanswered questions relevant to ER. Section 7.3 proposes related future research. In that section four lines of work that build on the research presented herein are outlined.

7.1 Summary of Main Results

This research has generated several contributions to the field of evolutionary robotics. These results will be reviewed briefly in the following paragraphs.

Very large arbitrarily complex neural controllers were evolved for group searching behaviors in robots. In particular, populations of neural controllers were evolved to play the competitive team game *Capture the Flag*. The networks used are much larger than those reported on in other ER research in the literature. Use of such networks is significant for two reasons: First, evolution of very large networks addresses issues related to the scalability of ER methods to complex tasks in which the details needed to specify ideal network architectures are unknown. With a few exceptions, previous work has been restricted to very small neural networks, generally using 10 or less hidden neurons. Second, large networks are likely to be

required to accommodate complex and numerous sensor inputs or complex sensor fusion elements.

This research is, to the knowledge of the author, the first ER work to evolved behavioral robotics controllers that depend exclusively on processed color vision for the sensing of their environment. The work made use of video images that were processed into 150 or more individual network inputs. The coupling of simulated and real world sensor systems was accomplished by selecting an intermediate level of processing that was both amenable to simulation, and could be implemented efficiently on real video images.

Evaluation of performance fitness during evolution was based on competition between individuals within each current generation of an evolving population. Similar types of selection have been used in other areas of evolutionary computing and co-competitive evolution has been applied in ER to evolve predator and prey mobile robot controllers [30][59][27]. The application of relative competitive selection was used in this research to drive the evolution of mobile robot controllers to play an interactive competitive team game, *Capture the Flag*. A bimodal fitness selection function was formulated to accommodate sub-minimally competent initial controller populations early in training (the Bootstrap Problem), but to select controllers based only on aggregate success-failure later in evolution.

Finally, evolved controllers were evaluated through the use of extensive tournaments with one another and also with a knowledge-based controller of known abilities. Metrics for absolute and comparative controller performance were developed using consistent sets of random game position initializations, and environments of increasing complexity. Such metrics are necessary for evaluating the performance of behavioral robotics controllers because sensor to motor mappings that produce a given behavior are generally not know for non-trivial behaviors. Additionally, relative and competitive aggregate training fitness functions produce a relative ranking, but give little information about absolute controller performance quality. The best evolved game-playing controllers in this research were able to win a modest majority of games in tournaments of 240 games against the knowledge-base controller.

7.2 The State of the Art of ER and Unresolved Issues

There are questions of fundamental concern to the further development of ER methods that remain unanswered. This section addresses several of these unresolved issues.

7.2.1 Artificial Evolution is Unnatural

Probably the most important question facing ER, and related fields is this: Is it possible to use artificial evolution to evolve arbitrarily complex intelligent systems in the general case? State of the art ER methods have not by any stretch of the imagination lead to the fully automated construction of complex behavioral robotics

controllers. As was noted in Chapters 1 and 2 of this dissertation, most work in the field has dealt with very basic proof-of-concept experiments. The research in this dissertation extends such basic work, but is still focused on a particular application within a class of tasks that can be formulated into competitive games.

The fundamental problem is that almost all artificial evolution methods differ from natural evolution in one crucial and fundamental aspect: artificial evolution methods are formulated to solve specific complex problems, while natural evolution solves only one simple problem, namely continued propagation of self-replicating structures. It has been argued that natural evolution has found numerous incredible and intricate solutions to *specific* physical and computational problems. This is not the case. Although living systems do indeed solve very complicated problems, this is purely a by-product of natural evolution's single selection process. Natural evolution does not seek to find any particular solutions to any particular problems beyond the propagation of self-replicating structures. The resulting complexities found in nature have not been the result of specific directed synthesis or optimization processes.

To solve a specific problem using simulated natural evolution, it is necessary to find a representation of the problem that is equivalent to survival. That in itself is fundamentally unnatural. Even worse, for a complex problem, designers may be required to know exactly how to solve the problem in order to formulate it in terms of survival in an artificial evolution environment.

There are only a few examples of the application of artificial evolution resulting in the evolution of complex novel intelligence. Those cases, though, exploit unusual features to formulate selection metrics or environments that allow survival to become synonymous with continued improvement of performance of a specific task. The best know example of these is the Checker playing neural network of Chellapilla and Fogel [68]. That work evolved Checkers-playing neural networks using only aggregate competitive selection. Using only aggregate win/lose selection, randomly initialize networks were evolved in an all-in-one (non-incremental) evolution session into networks that could beat human experts (although not masters). The work, however, exploited a feature of Checkers that is very rarely found in complex systems: the game will almost always be played to completion with a winner, no matter the skill level of the players. This is very important, because it allowed randomly initialized networks of very low competence to compete against each other and to be ranked in a meaningful way.

It is clear that certain classes of problems can be formulated so that truly novel (problem specific) intelligences can be evolved to solve them. This is likely the case even for certain problems that humans do not know how to solve well. But the question of generalization remains unanswered. The complexity of evolved systems will likely need to be increased by many-fold in order to really address the question of general performance-feedback driven evolution of intelligence.

7.2.2 Maintenance of Memory

Another pressing problem facing ER is that of maintenance of learned behaviors. For most of the proof-of-concept work that has been done so far, this issue did not arise because the few sub-behaviors required to generate the behaviors in those studies were continually maintained every generation.

In a fully evolvable system, behaviors that aren't tested or reinforced each generation run the risk of being corrupted. With black box systems, such as arbitrarily connected and evolvable neural networks, overcoming this problem is non-trivial. Various methods such as selective attenuation of mutation rates, and the selective fixing of sections of networks deemed to be well trained have been suggested. These though, can lead to stagnation of the leaning process. Often, networks evolve to *not evolve* after a point, and generate results no better than those which leave un-reinforced behaviors vulnerable.

It has also been suggested that something other than a black-box architecture be evolved. Then, perhaps methods can be developed to identify which elements of the evolved controllers represent useful behaviors. Unfortunately, in order to maintain the potential utility of ER to generate novel complex controllers automatically, any methods of controller analysis would have to be integrated into the fitness testing and propagation phases of evolution. This brings us back to the problem of fitness function specification, but now this appears in the context of determining which portions of an evolved controller should be insulated from mutational corruption.

Integration of fitness evaluation over several or many trials is one viable way to increase the retention of acquired behaviors. The application of this integration is surprisingly non-trivial. In the work involving the evolution of checkers playing programs [68], fitnesses were integrated over five trials for each individual before ranking and selection. Other levels of integration were not reported on. Of course, computation times increase linearly with integration level. It is not at all clear that benefit increases linearly with integration level, though.

In the research presented in this dissertation, no fitness integration was applied. Only a single tournament per generation was played before ranking and propagation (for the results presented in Chapters 5 and 6). However in preliminary work, integration levels were tested at 5 and 10 tournaments per generation (data not shown), but these resulted in no increase of rate of behavior acquisition. This was not perused in-depth in this research, but it is speculated that less than one beneficial mutation was found per generation, and that an individual receiving a beneficial mutation was usually selected for propagation, even if only one tournament was played. Additionally, it was noted during the selection of the "best" individuals (for post-evolution testing), that diversity of behaviors remained quite high even in the fully evolved populations of controllers. There, scores from 30 tournaments were averaged. For every game initialization used during selection of the "best" controllers, there was always at least one individual in a given population that could win the game. In some cases, though, the controller that could win one particular game was unable to win most other

games. This maintenance of diversity may significantly complicate methods of performance integration.

One method suggested here to elevate the problem of maintenance of behaviors in black-box controllers would be to include elements into the controller that would identify which portions of the controller were active during a given evaluation test. Then, the mutations associated with currently functioning elements could be identified and retained, while others, affecting inactive areas of the controllers could be discarded, thereby preventing corruption of any un-displayed behaviors. This is suggested as an area of possible future research in Section 7.3.4.

7.2.3 Training Plateaus

Plateauing during training or evolution is the natural result of reinforcement learning methods that evaluate fitness with static functional fitness functions. The fitness function will eventually be maximized (or error minimized). In contrast, in some cases, relative fitness selection method methods such as co-competitive and competitive selection have the potential to generate increasing fitness indefinitely by creating an ever-increasing level of environmental difficulty (due to the increasing levels of competitor competence). Even so, relative competitive methods appear to plateau at competent, but perhaps sub-optimal levels. This was seen in the results presented in Chapters 5 and 6. There, the best-evolved controller out of all the evolutions was able to play competitively with a hand-coded knowledge-based controller. It is very likely that the hand-coded controller is not the best possible controller. Further evolution, though, did not result in further improvement.

Controllers from the 450th generation were just at competent as controllers from the 650th generation.

In the long run, noise in fitness selection is likely to be a limiting factor. Although it may be the case that one individual in the population is better than others, the fitness ranking method might be un-able to resolve the differences. In turn, noise in selection is affected by some of the issues discussed in previous sections of this chapter, such as maintenance of beneficial behaviors.

Controller architecture and evolutionary algorithm settings are also likely to play an important role in the process of plateauing. In the work presented in this dissertation, some significant gains in evolved controller abilities were made by adjusting and manipulating particular elements of the algorithm. If true competitive selection can be achieved, then noise in selection becomes the major limiting factor. Basic evolutionary computing theory might play an important role in algorithm and parameter selection. However, theory related to the dynamics of real-valued, variable dimension evolving systems is not well developed. Most evolutionary computing theory, such as schema theory [3], relies on simplifying assumptions that virtually preclude its application to complex problems. Additionally, in the controller architecture search spaces used in ER, the fraction of controller configurations that have detectable fitness approaches zero. This does not mean that there are no fit solutions, rather, there are likely uncountable infinitely many fit solutions, but the set of fit solutions makes up an infinitesimally small proportion of the set of all possible

controllers (also uncountably infinite). Because of this, many of the metrics used in other areas of EC are in-applicable to ER or to the evolution of continuous intelligent systems. These issue have been discussed to a degree in the ER literature[81][47][82][92][93], but rigorous theory has not been developed [81].

7.2.4 Unconventional Methods

Some of the issues raised in the previous sub-sections may be ameliorated by application of methods that would be considered unconventional by current engineering standards.

For example, mechanisms could be included into artificial evolutionary simulations, which would allow a human to arbitrarily alter the course of an otherwise automatic process. An artificial evolution process in which a human acts as the soul selective factor is referred to as “breeder training” and has been studied in [83][84][94]. Partial-breeder training might serve as an alternative to incremental fitness functions, and also to side-step sub-optimal training plateaus. An automatic evolutionary process, driven by a competitive metric could be allowed to progress with occasional monitoring from a human. The human could at times choose to alter the course of evolution by selecting individuals bases on their observed behaviors. In any generation in which a human selection was made, selected controllers would be given the highest fitness rating and propagated to the next generation. Such a course of training has generally been considered of limited value because the process relies on arbitrary whim and is irreproducible. Even so, once generated, neural controllers, including those generated by arbitrary means, can be duplicated and implemented on

many robots, even if it is not possible to exactly replicate the process that derived the original network. Systems that would allow a network to be easily transferred from one environment or population to another to allow the application of multiple training methods on a single network would also fall into this class of hybrid training methods.

An additional unconventional method of automated controller design might involve returning to ER's roots in artificial life [2]. A proposed method involves evolving complex robot controllers in very complicated environments using a simple survival performance metric that requires robots to acquire "energy" to replicate. Some such work has been conducted in ER and is referred to as metabolism based selection [12][50]. If an environment is complex enough and robots can interact with one another, complex behaviors have the potential to evolve over many generations. As is the case with natural evolution, the particular behaviors may be difficult to predict. The designer would then attempt to select interesting evolved behaviors, and to build a library of behavioral modules that could be incorporated into other controllers, or used as pre-trained initial controller populations for further evolution.

7.3 Future Research

The research presented in this dissertation explored questions related to the extension of ER methods to the evolution of general and complex robot behaviors. In a broader sense, the work addressed issues related to the automatic synthesis of embodied

machine intelligence. Compared to other robotic and machine intelligence fields, ER is in its early stages and the opportunities for new research are many. Several topics for future and related research are outlined in the remainder of this chapter.

7.3.1 Application of Advanced Computing to ER

Although an extensive amount of code was written to develop the software components used in this research, no special effort was made to optimize simulation code or finely tune elements of controller portability. Incorporation of sophisticated software techniques (such as those used in video gaming engines) to develop very fast simulation environments and to include advanced sensor fusion elements would be very beneficial. An increase in simulation speeds of between 100 and 10,000 times would allow comparative experiments to be preformed that are not feasible using the current platform. Comparative experiments are used in reinforcement learning and other iterative machine learning systems to determine the effects and ranges of parameter and environmental settings. Some work along these lines has been done in ER [41] but such work used very simple simulation environments, simple simulated sensors and small controller structures.

Although not discussed in detail in the text, the ER simulation and neural computing environment used here generates and records numerous metrics and data over the course of an evolution. These include data recording trends in network size, the ages of neurons within networks, mutations rates and magnitudes, selection rates, and population turnover points (takeover), among other data. Much faster simulation and evolution speeds may allow these trend data to be correlated with evolution outcomes

and to be formulated into diagnostic metrics. Such diagnostic metrics would be of great value to the field of ER. Currently, ER methods are evaluated almost exclusively by demonstration, i.e. if a method produces a functioning controller, it is good.

Another area in which fast computing and new hardware technologies could be applied to ER is in three-dimensional simulation. In this work, as in almost all ER work, simulated and real robots were given a two-dimensional environment view. New sensors and high-quality simulations could be used to extend this research into three dimensions. We propose implementing a LADAR-based [85] sensing system in conjunction with color overlaid from a proximal video camera to replace the existing two-dimensional real and simulated range sensors used by the EvBots. This would allow robot controllers to receive an unprecedented level and resolution of spatial and object-type information from their environments. This sensor information would be suitable to be fed directly into complex neural controllers.

Additionally the next generation of robots, the EvBotII [77] can be equipped with directional sound sensor arrays and have been designed to accommodate other sensor modules simultaneously. This will allow ER experiments involving sensor fusion to be conducted.

7.3.2 Integrating Human and Machine Learning

Another area of great interest is that of incorporating human behavior into evolving systems, not through the use of human designed and biased functional fitness

functions, but by direct human-robot interaction. Such methods could be used both to jumpstart the evolutionary process, and to distil human kinetic skills into mobile robots.

Proposed work in the CRIM involves using humans to control robots remotely in the real world and in simulated environments using a simple game interface. During this process, robot sensor readings and the resulting human-generated robot movement commands will be recorded. These sensor readings and motor commands will be formulated into training data sets similar to those used in [69] and used to train neural controllers.

In addition, a natural extension of this work would be to compete evolved controllers against human-controlled robots for the purposes of evaluating real-world evolved controller fitness. Such work would duplicate the comparative fitness metrics discussed in Chapter 6, but here a human would take the place of the knowledge-based controller. In such a scenario, one team of EvBots would be configured with evolved neural controllers. The second team would be driven remotely by humans. The human competitors would be physically separated from the robots and would make control decisions based on processed video images of the same format used by the evolved controllers. The EvBot research platform is currently being upgraded to accommodate such experiments.

7.3.3 Application of the Bimodal Training Metric to Multiple Tasks

One of the motivations behind defining a fitness function with multiple modes was to make it applicable to a general class of behaviors. The initial mode of the function selects for the ability for move in a given environment. It is the second mode that selects for performance of a given high-level task or behavior. This second mode though, is purely aggregate, so it is only necessary to specify a single condition for success/failure to modify the bimodal function to select for a new complex behavior. It would be of interest to show experimentally that the metric does generalize to several tasks when only the single task-specific high-level success/failure binary element of the function is changed.

One specific task that the metric might be applicable to is grouping (or flocking). A limited amount of research has been done on robot flocking behaviors in ER [61][63]. These have used complex hand-formulated functional fitness functions for selection. Flocking behavior can be formulated into a competitive game as follows: many robots of two teams (of different colors) can be initially placed at random locations throughout an environment. Robots can then be allowed to perambulate about the environment. The game would be won when all of the robots from one of the teams had gathered together within a predefined radius.

Group foraging is another possible task. This would involve robots collecting small objects (often, such objects are referred to as “food” in the ER and artificial life literature). Initially, food objects would be scattered randomly about an environment.

Robots would start in a group and move through their environment attempting to come in contact with as many food objects as possible. Again, this behavior can be formulated into a competitive game with a binary win/lose end condition. For instance, if two groups of robots are involved, an end condition could be defined as the point at which one or the other group has consumed 50% or more of the food objects.

Finally, using the next generation EvBotII architecture, behaviors involving homing on sound sources in unknown terrain can be studied. The *Capture the Flag* goal objects can be modified to emit a sound signal so that robots can evolve directed homing behaviors in complex environments. The training fitness function could be applied without modification. Such experiments, would allow the effects of fundamentally different sensor systems on the evolution of behavior to be studied.

7.3.4 Maintenance of Learned Behaviors in Evolution

The final proposed area of research related to this work is focused on retention and maintenance of complex behaviors. As noted earlier, complex behaviors may be difficult to achieve or maintain during the evolution of neural controllers because sub-behaviors are subject to stochastic degradation if they are not reinforced every generation. For many complex behaviors, sub-behaviors and simple responses may only occasionally be expressed. It would be ideal if a sub-behavior or reaction once found, was retained without mutation until it was expressed again.

It is possible to formulate many intrusive and problem-specific hand-designed methods to improve training and retention of sub-behaviors in specific well-studied cases. A human observer can identify some particular sub-behavior, and then formulate training conditions so that the behavior is likely to be retained. However, truly automatic methods that do not rely on detailed human understanding of specific applications have not been well studied, at least as applied to ER.

Simple integration of multiple fitness evaluation sessions during each generation has been used in almost all research in which reinforcement learning has been applied to synthesize machine intelligence. Such integration does increase the likelihood of a wider spread of behaviors or reactions being expressed. However, and perhaps counter-intuitively, low levels of such integration may not always result in more efficient selection. As discussed in Section 7.2, this might reflect an interaction with a maintenance of diversity of strategies in a population. It is likely that massive levels of integration, (on the order of 100 tests per individual per generation) would produce improved maintenance of sub-behaviors, but this would be very computationally expensive. Such repeated testing is also very computationally wasteful because commonly expressed behaviors would be tested far more that is necessary to determine relative fitness. Again, it is tempting to suggest methods that actively identify useful and common sub-behaviors at the functional level, and to explicitly reinforce them, but that would represent a reduction of the automated synthesis process to one of glorified optimization.

There is another more fundamental reason for insulating un-tested or unused elements of the controller structure from degradation by mutation. There is likely an upper limit on the complexity of behavior that can be maintained in neural network structures that are mutated at a high rate. However, mutation levels must be kept high enough to keep networks in exploration of their search space throughout most of their training. For behavioral evolution, exploitation (sometimes called fine-tuning) plays a very small role, if any, in the primary synthesis process. This is one of the fundamental factors that distinguishes evolutionary synthesis processes, from optimization and example driven training methods.

To address these issues, we propose research in which evolving neural networks are coupled to monitoring software structures designed to identify which portions of the networks are active during a given evolutionary testing session. Since behaviors are not likely to be localized in sub-regions within neural networks, this must be done at the individual neuron level. Even at the single-neuron level, it is probably not possible to fully de-integrate particular behaviors. Even so, neurons actively participating in the expression of a behavior will produce varying levels of output over the course of a testing session. A metric can be formulated that will distinguish active neurons from inactive ones. During training, this metric could be applied to determine which mutations are accepted, and which are discarded in the best performing networks in an evolving population.

7.4 Chapter summary

This Chapter concludes the dissertation. A summery of the main research results was presented. In addition, current issues and unanswered questions relevant to the future development of the field of ER were discussed. Finally, several suggestions for further research were presented.

REFERENCES

[1] R.C. Arkin, *Behavior-Based Robotics*, The MIT Press, Cambridge, MA, 1998.

[2] C. Adami, *Introduction to Artificial Life*, Springer-Verlag, New York, 1998.

[3] J.J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan, 1975.

[4] W.G. Walter, *The Living Brain*, W. H. Norton, New York, 1953.

[5] V. Braitenberg, *Vehicles: Experiments in Synthetic Psychology*, MIT Press, Cambridge, MA,1984.

[6] R.A. Brooks, "Elephants Don't Play Chess," *Robotics and Autonomous Systems*, vol. 6, pp. 3-15, 1990.

[7] J.R. Koza, "Evolution of subsumption using genetic programming," in F.J. Varela, P. Bourgine Eds., *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pp. 110-119, The MIT Press, Cambridge, MA, 1992.

[8] R.A Brooks, "Intelligence Without Representation", *Artificial Intelligence Journal*, vol. 47, pp. 139–159, 1991.

[9] R.A. Brooks. "Artificial life and real robots," in, *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, F.J. Varela, P. Bourgine, Eds, MIT Press/Bradford Books, Cambridge, MA, 1992, pp. 3-10.

[10] S. Nolfi, D. Floreano, O. Miglino, F. Mondada, "How to Evolve Autonomous Robots: Different Approaches in Evolutionary Robotics," in R.A. Brooks, P. Maes Eds., *Proceedings of the IV International Workshop on Artificial Life*, Cambridge, MA, MIT Press, 1994.

[11] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in F. Moran, A. Moreno, J. Merelo, P. Chacon, Eds., *Advances in Artificial Life: Proc. 3rd European Conference on Artificial Life, Lecture Notes in Artificial Intelligence 929*, Springer-Verlag, 1995, pages 704-720.

[12] O. Michel, "An Artificial Life Approach for the Synthesis of Autonomous Agents," in *Proceedings of the European Conference on Artificial Evolution*, Springer-Verlag, 1995, pp. 220-231.

[13] H.H. Lund, O. Miglino, "From Simulated to Real Robots," in *Proceedings of IEEE International Conference on Evolutionary Computation*, 1996, pp. 362-365.

[14] W. Lee, "Evolving Autonomous Robot: From Controller to Morphology," *IEICE Trans. Inf. & Syst.*, vol E83-D, no. 2, pp. 200-210, 2000.

[15] F. Gruau, "Automatic definition of modular neural networks," *Adaptive Behavior*, vol. 2, pp.151-183, 1995.

[16] D. Filliat, J. Kodjabachian, J.A. Meyer, "Incremental evolution of neural controllers for navigation in a 6 1egged robot," in Sugisaka and Tanaka, Eds., *Proc. Fourth International Symposium on Artificial Life and Robotics*, Oita Univ. Press, 1999.

[17] N. Jakobi, "Running Across the Reality Gap: Octopod Locomotion Evolved in a Minimal Simulation," in P. Husbands, J.A. Meyer, Eds., *Evolutionary Robotics: First European Workshop, EvoRobot98*, Springer-Verlag, 1998, pp. 39-58.

[18] A. Lewis, A.H. Fagg, G.A. Bekey, "Genetic Algorithms for Gait Synthesis in a Hexapod Robot," *Recent trends in mobile robots*, Y.F. Zheng, Ed, World Scientific, 1993.

[19] S. Nolfi, D. Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*, The MIT Press, Cambridge Massachusetts, 2000.

[20] D. Cliff, P Husbands, I. Harvey, "General Visual Robot Controller Networks via Artificial Evolution," in D. Casasent, Ed., *Proceedings of the Society of Photo-optical Instrumentation Engineers Conference 1993 (SPIE93), Session on Intelligent Robots and Computer Vision XII: Algorithms and Techniques*, SPIE Conference vol. 2055, 1993, pp. 271-282.

[21] S. Nolfi, J.L. Elman, D. Parisi, "Learning and evolution in neural networks," *Adaptive Behavior*, vol. 3, no. 1, pp. 5-28, 1994.

[22] D. Floreano, F. Mondada , "Evolution of homing navigation in a real mobile robot," *IEEE Transactions on Systems, Man, Cybernetics Part B: Cybernetics*, vol. 26 no. 3, pp. 396-407, 1996.

[23] H.H. Lund, J. Hallman, "Evolving Sufficient Robot Controllers," in *Proceedings of the IEEE International Conference on Evolutionary Computation*, 1997, pp. 495-499.

[24] J. Kodjabachian, J.A. Meyer, "Evolution and development of neural networks controlling locomotion, gradient-following, and obstacle avoidance in artificial insects," *IEEE Transaction on Neural Networks*, vol. 9, no. 5, pp. 796-812, Sept. 1998.

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

[26] G.S. Hornby, S. Takamura, J. Yokono, O. Hanagata, M. Fujita, J. Pollack, "Evolution of Controllers from a High-Level Simulator to a High DOF Robot," in J. Miller, Ed., *Evolvable Systems: from biology to hardware; proceedings of the third international conference (ICES 2000)*, *Lecture Notes in Computer Science*, vol. 1801, Springer, 2000, pp. 80-89.

[27] D. Cliff and G. F. Miller, “Tracking the Red Queen: Measurements of adaptive progress in co-evolutionary simulations,” in F. Moran, A. Moreno, J. J. Merelo, P. Cachon Eds., *Advances in Artificial Life: Proceedings of the Third European Conference on Artificial Life (ECAL95)*. Lecture Notes in Artificial Intelligence 929, Springer-Verlag, 1995, pp.200-218,

[28] D. Cliff and G. F. Miller ``Co-Evolution of Pursuit and Evasion II: Simulation Methods and Results,", in P. Maes, M. Mataric, J.-A. Meyer, J. Pollack, W. Wilson Eds., *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB96)*. MIT Press Bradford Books, pp.506-515, 1996.

[29] I. Harvey, P. Husbands, D. Cliff, A. Thompson, N. Jakobi, “Evolutionary robotics: the Sussex approach,” *Robotics and Autonomous Systems*, vol. 20, no. 2-4, pp. 205-224, 1997.

[30] S. Nolfi, D. Floreano, “Co-evolving predator and prey robots: Do 'arms races' arise in artificial evolution?” *Artificial Life*, vol. 4, no. 4, pp. 311-335, 1998.

[31] Buason & Ziemke (in press). Competitive Co-Evolution of Predator and Prey Sensory-Motor Systems. To appear in the proceedings of the EvoRob 2003 workshop. Springer.

[32] D. Floreano, S. Nolfi, F. Mondada, “Competitive Co-Evolutionary Robotics: From Theory to Practice,” in R. Pfeifer, Ed., *From Animals to Animats 5. Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior (SAB'1998)*, Cambridge, MA, MIT Press, 1998.

[33] A.L. Nelson, E. Grant, T.C. Henderson, “Competitive relative performance evaluation of neural controllers for competitive game playing with teams of real mobile robots,” *Measuring the Performance and Intelligence of Systems: Proceedings of the 2002 PerMIS Workshop*, NIST Special Publication 990, Gaithersburg MD, Aug. 13-15, 2002, pp. 43-50.

[34] A.L. Nelson, E. Grant, T.C. Henderson, “Evolution of neural controllers for competitive game playing with teams of mobile robots,” *Journal of Robotics and Autonomous Systems*, 2003, submitted for publication.

[35] E.D.V. Simoes, K.R. Dimond, “Embedding a distributed evolutionary system into a population of autonomous mobile robots,” in *2001 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 2, 2001, pp.1069-1074, 2001.

[36] J. Pollack, H. Lipson, P. Funes, S. Ficici, G. Hornby, “Coevolutionary Robotics,” in J.R. Koza, A. Stoica, D. Keymeulen, J. Lohn, Eds., *The First NASA/DoD Workshop on Evolvable Hardware*, 1999, pp. 208-216.

[37] A. Ishiguro, S. Tokura, T. Kondo, Y. Uchikawa, “Reduction of the Gap between Simulated and Real Environments in Evolutionary Robotics: A Dynamically-Rearranging Neural Network Approach,” in *The 1999 IEEE International Conference on Systems, Man, and Cybernetics: Conference Proceedings*, Vol. 3, 1999, pp. 239-244.

[38] K. Kawai, A. Ishiguro, P. Eggenberger, "Incremental Evolution Of Neurocontrollers with a Diffusion-Reaction Mechanism of Neuromodulators," in *International Conference on Intelligent Robots and Systems 2001 Proceedings*, vol. 4, Maui, HI, Oct. 29- Nov. 3, 2001, IEEE/RSJ, pp. 2384-2391.

[39] W. Lee, J. Hallam, H. Lund, "Applying Genetic Programming to Evolve Behavior Primitives and Arbitrators for Mobile Robots," in *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, 1997, pp. 495-499.

[40] S. Nolfi, "Evolving non-trivial behaviors on real robots," *Robotics and Autonomous Systems*, vol. 22, no. 3-4 pp. 187-198, 1997.

[41] T. Ziemke, "Remembering how to behave: Recurrent neural networks for adaptive robot behavior," in Medsker and Jain Eds., *Recurrent Neural Networks: Design and Applications*, Boca Raton, CRC Press, 1999.

[42] D. Floreano, and J. Urzelai, "Evolutionary robots with on-line self-organization and behavioral fitness," *Neural Networks*, vol. 13, no. 4-5, pp. 431-443, June 2000.

[43] E. Tuci, M. Quinn, I. Harvey, "Evolving Fixed-Weight Networks for Learning Robots," in *Proceedings of the 2002 Congress on Evolutionary Computing*, Honolulu HI, vol 2, 2002, pp 1970-1975.

[44] I. Ashiru C.A.Czarnecki, "Evolving Communicating Controllers for Multiple Mobile Robot Systems," in *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, vol. 4, 1998, pp. 3498-3503.

[45] M. Quinn, "Evolving cooperative homogeneous multi-robot teams," in *Proceedings of the IEEE / RSJ International Conference on Intelligent Robots and Systems (IROS 2000)*, vol.3, Takamatsu Japan, 2000, pp. 1798-1803.

[46] G. Baldassarre, S. Nolfi, D. Parisi, "Evolving Mobile Robots Able to Display Collective Behaviours," in C. K. Hemelrijk, E. Bonabeau, Eds., *Proceedings of the International Workshop on Self-Organisation and Evolution of Social Behaviour*, Monte Verità, Ascona, Switzerland, Sept. 8-13, 2002, pp. 11-22.

[47] T.M.C. Smith, P. Husbands, M. O'Shea, "Neutral Networks in an Evolutionary Robotics Search Space," *Congress on Evolutionary Computation (CEC2001*), IEEE Press, 2001, pp 136-145.

[48] M. Mataria, D. Cliff, "Challenges in evolving controllers for physical robots," *Robotics and Autonomous Systems*, vol. 19, no. 1, pp. 67-83, Nov. 1996.

[49] L.A. Meeden, D. Kumar, "Trends in Evolutionary Robotics," in *Soft Computing for Intelligent Robotic Systems*, L.C. Jain, T. Fukuda, Eds., Physica-Verlag, New York, NY, 1998, pp. 215-233.

[50] R.A. Watson, S.G. Ficici, J.B. Pollack, "Embodied Evolution: Distributing an Evolutionary Algorithm in a Population of Robots," *Robotics and Autonomous Systems*, vol. 39, no. 1, pp 1-18, Apr. 2002.

[51] T. Gomi A. Griffith, "Evolutionary Robotics – An Overview," in *Proceedings of IEEE International Conference on Evolutionary Computation*, 1996, pp. 40-49.

[52] I. Harvey, P. Husbands, D. Cliff, "Seeing the light: artificial evolution, real vision," in D. Cliff, P. Husbands, J.-A. Meyer, S. Wilson Eds., *From Animals to Animates 3, Proc. of 3rd Intl. Conf. on Simulation of Adaptive Behavior, SAB94*, MIT Press/Bradford Books, Boston, MA, 1994, pp. 392-401.

[53] F. Southley, F. Karray, "Approaching Evolutionary Robotics Through Population-Based Incremental Learning," in *Proceedings of the 1999 IEEE Conference on Systems, Man, and Cybernetics*, vol. 2, 1999, pp. 710-715.

[54] S. Luke, C. Hohn, J. Farris, G. Jackson, J. Hendler, "Co-evolving soccer softbot team coordination with genetic programming," in *Proceedings of the First International Workshop on RoboCup*, Nagoya, Japan, Aug. 1997, pp. 115-118.

[55] W. Lee, "Evolving Complex Robot Behaviors" *Information Sciences*, vol. 121, no. 1-2, pp. 1-25, 1999.

[56] D. Floreano, J. Urzelai, "Evolutionary Robots: The Next Generation," in *The 7th International Symposium on Evolutionary Robotics (ER2000): From Intelligent Robots to Artificial Life*, Gomi T. Ed., pp. 231-266, AAI Books, 2000.

[57] S. Nolfi, D. Marocco, "Evolving robots able to visually discriminate between objects with different sizes," *International Journal of Robotics and Automation*, vol.17, no. 4, pp.163-170, 2002.

[58] D. Marocco, D. Floreano, "Active Vision and Feature Selection in Evolutionary Behavioral Systems," in J. Hallam, D. Floreano, G. Hayes, J, Meyer, Eds., *From Animals to Animats 7*, Cambridge, MA, MIT Press, 2002.

[59] E.H. Østergaard H.H. Lund, "Co-Evolving Complex Robot Behavior", in *Proceedings of ICES'03, The 5th International Conference on Evolvable Systems: From Biology to Hardware*, Trondheim, Norway, Mar. 17 – 20, 2003.

[60] A. Bonarini, C. Bonacina, M. Matteucci, "An approach to the design of reinforcement functions in real world agent-based applications," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 31, no. 3, pp. 288-301, 2001.

[61] M. Quinn, "Evolving communication without dedicated communication channels," in J. Kelemen, P. Sosik, Eds., *Advances in Artificial Life: Sixth European Conference on Artificial Life (ECAL 2001)*, Prague, Czech Republic, Sept. 2001, Springer, pp. 357-366.

[62] J.A. Driscoll, R.A. Peters, II, "A development environment for evolutionary robotics," in *2000 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 5, 2000, pp. 3841-3845.

[63] M. Quinn, L. Smith, G. Mayley, P. Husbands, "Evolving team behaviour for real robots," in *EPSRC/BBSRC International Workshop on Biologically-*

Inspired Robotics: The Legacy of W. Grey Walter (WGW '02), Aug. 14-16, 2002, HP Bristol Labs, U.K.

[64] F. Pasemann, U. Steinmetz, M. Hülse, B. Lara, "Evolving brain structures for robot control," in *IWANN'01 Proceedings LNCS, 2085*, vol. II, Granada, Spain, June 13-15, 2001, Springer-Verlag, pp. 410-417.

[65] F. Gomez, R. Miikkulainen, "Incremental Evolution of Complex General Behavior," *Adaptive Behavior*, vol. 5, pp. 317-342, 1997.

[66] H. Nakamura, A. Ishiguro, Y. Uchilkawa, "Evolutionary construction of behavior arbitration mechanisms based on dynamically-rearranging neural networks," in *Proceedings of the 2000 Congress on Evolutionary Computation*, vol.1, IEEE, 2000, pp. 158-165.

[67] O. Miglino, D. Denaro, G. Tascini, D. Parisi, "Detour Behavior in Evolving Robots: Are Internal Representations Necessary?" in *Proceedings of First European Workshop on Evolutionary Robotics*, Springer-Verlag, 1998, pp. 59-70.

[68] K. Chellapilla, D.B. Fogel, "Evolving an Expert Checkers Playing Program Without Using Human Expertise," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 422-428, 2001.

[69] A.L. Nelson, E. Grant, G. Lee, "Using Genetic Algorithms to Capture Behavioral Traits Exhibited by Knowledge Based Robot Agents," in *Proceedings of the ISCA 15th International Conference: Computer Applications in Industry and Engineering (CAINE-2002)*, San Diego, CA, Nov. 7-9, 2002, pp. 92-97, ISBN: 1-880843-45-5.

[70] W. Lee, S. Lia, "An Example-Based Approach for Evolving Robot Controllers," in *Proceedings of the 1999 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 5, 1999, pp. 618-623.

[71] A.J. Ijspeert, "Synthetic Approaches to Neurobiology: Review and Case Study in the Control of Anguiliform Locomotion," in *Proceedings of the Fifth European Conference on Artificial Life, ECAL99*, Springer-Verlag, 1999, pp. 195-204.

[72] A.L. Nelson, E. Grant, J.M. Galeotti, S. Rhody, "Maze Exploration Behaviors Using an Integrated Evolutionary Robotics Environment," *Journal of Robotics and Autonomous Systems*, 2003, to be published.

[73] J.C. Bongard, "Evolved Sensor Fusion and Dissociation in an Embodied Agent," in *Proceedings of the EPSRC/BBSRC International Workshop Biologically-Inspired Robotics: The Legacy of W. Grey Walter*, 2002, pp. 102-109.

[74] K. Swingler, *Applying Neural Networks: A Practical Guide*, Morgan Kaufman Publishers Inc, San Francisco, CA, 1996, ISBN 0-12-679170-8.

[75] I. Cloete, J. M. Zurada Eds., *Knowledge-Based Neurocomputing*, The MIT Press, Cambridge Ma., London, England, ISBN 0-262-03274-0, 2000

[76] J. Galeotti, S. Rhody, A.L. Nelson, E. Grant, G. Lee, “EvBots – The Design and Construction Of A Mobile Robot Colony for Conducting Evolutionary Robotic Experiments,” in *Proceedings of the ISCA 15th International Conference: Computer Applications in Industry and Engineering (CAINE-2002)*, San Diego, CA, Nov. 7-9, 2002, pp. 86-91, ISBN: 1-880843-45-5.

[77] J.M. Galeotti, "The EvBot: A Small Autonomous Mobile Robot for the Study of Evolutionary Algorithms in Distributed Robotics", Master's thesis, North Carolina State University, 2002.

[78] L.S. Mattos, “The EvBot II: An Enhanced Evolutionary Robotics Platform Equipped with Integrated Sensing for Control,” Master's thesis, North Carolina State University, 2003.

[79] P.H.M. Spronck, I.G. Sprinkhuizen-Kuyper, E.O. Postma, “Evolutionary Learning of a Neural Robot Controller,” *International Conference on Computational Intelligence for Modelling in Control and Automation – (CIMCA'2001)*, 2001, pp. 511-519, ISBN: 0-858-89847-0.

[80] S. Nolfi, “Evolutionary Robotics: Exploiting the Full Power of Self-Organization,” *Connection Science*, vol. 10, pp. 167-183, 1998.

[81] S.G. Ficici, J.B. Pollack, “Game Theory and the Simple Coevolutionary Algorithm: Some Preliminary Results on Fitness Sharing,” in *GECCO 2001 Workshop on Coevolution: Turning Adaptative Algorithms upon Themselves*, 2001.

[82] T.M.C Smith, P. Husbands, M. O'Shea, “Not Measuring Evolvability: Initial Investigation of an Evolutionary Robotics Search Space,” in *Congress on Evolutionary Computation (CEC2001)*, IEEE Press, 2001, pp. 9-16.

[83] H.H. Lund, O. Miglino, “Evolving and Breeding Robots,” in *Proceedings of First European Workshop on Evolutionary Robotics*, Paris, France, Springer-Verlag, 1998.

[84] H.H. Lund, O. Miglino, L. Pagliarini, A. Billard, A. Ijspeert, “Evolutionary Robotics-A Children’s Game,” in *Evolutionary Computation Proceedings, 1998 IEEE World Congress on Computational Intelligence*, 1998, pp. 154-158.

[85] M.W. Scott, “Range Imaging Laser Radar,” U.S. Patent 4,935,616, June 19, 1990.

[86] R. Salomon, L. Lichtenstieger, “Exploring Different Coding Schemes for the Evolution of an Artificial Insect Eye,” in *Proceedings of the first IEEE Symposium on Combinations of Evolutionary Computing and Neural Networks*, 2000, pp. 10-16.

[87] O. Miglino, H.H. Lund, S. Nolfi, “Evolving mobile robots in simulated and real environments,” *Artificial Life*, vol. 2, no. 4, pp. 417-434, 1995.

[88] V.V. Hafner, R. Salomon “Evolving Neural Controllers for Visual Navigation,” in *Proceedings of the Fourth Congress on Evolutionary Computation (CEC-*

2002), IEEE World Congress on Computational Intelligence, Honolulu, 2002. (ISBN: 0-7803-7282-4)

[89] A. Lubberts, R. Miikkulainen, "Co-Evolving a Go-Playing Neural Network," in *Coevolution: Turning Algorithms upon Themselves, Birds-of-a-Feather Workshop, Genetic and Evolutionary Computation Conference (GECCO-2001)*, San Francisco, CA, 2001.

[90] T. Ziemke, "On 'Parts' and 'Wholes' of Adaptive Behavior: Functional Modularity and Diachronic Structure in Recurrent Neural Robot Controllers," in *From animals to animats 6 - Proceedings of the Fifth International Conference on the Simulation of Adaptive Behavior (SAB 2000)*, Paris, Sept. 2000, MIT Press, Cambridge, MA.

[91] L. Rade, B. Westergren, *Mathematics Handbook for Scientists and Engineers*, Birkhauser, Sweden, 1995, pp. 471-476.

[92] A. Bucci, J.B. Pollack, "Order-theoretic Analysis of Coevolution Problems: Coevolutionary Statics," in *2002 Genetic and Evolutionary Computation Conference Workshop: Understanding Coevolution*, 2002.

[93] T.M.C Smith, P. Husbands, P. Layzell, M. O'Shea, "Fitness Landscapes and Evolvablility," *Evolutionary Computation*, vol. 10, no. 1, pp. 1-34, 2002.

[94] F. Kaplan, P. Oudeyer, E. Kubinyi, A. Miklosi, "Robotic clicker training," *Robotics and Autonomous Systems*, vol. 38, no. 3-4, pp. 197-206, 2002.