Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Robotics and Autonomous Systems 46 (2004) 135–150

Robotics and Autonomous Systems

www.elsevier.com/locate/robot

Evolution of neural controllers for competitive game playing with teams of mobile robots

A.L. Nelson [a,*], E. Grant [a], T.C. Henderson [b]

[a] *Department of Electrical and Computer Engineering, Center for Robotics and Intelligent Machines, North Carolina State University, Raleigh, NC 27695-7911, USA*
[b] *School of Computing, University of Utah, Salt Lake City, UT 84112, USA*

Received 22 April 2003; received in revised form 1 November 2003; accepted 12 January 2004

Abstract

In this work, we describe the evolutionary training of artificial neural network controllers for competitive team game playing behaviors by teams of real mobile robots. This research emphasized the development of methods to automate the production of behavioral robot controllers. We seek methods that do not require a human designer to define specific intermediate behaviors for a complex robot task. The work made use of a real mobile robot colony (EVolutionary roBOTs) and a closely coupled computer-based simulated training environment. The acquisition of behavior in an evolutionary robotics system was demonstrated using a robotic version of the game *Capture the Flag*. In this game, played by two teams of competing robots, each team tries to defend its own goal while trying to 'attack' another goal defended by the other team. Robot neural controllers relied entirely on processed video data for sensing of their environment. Robot controllers were evolved in a simulated environment using evolutionary training algorithms. In the evolutionary process, each generation consisted of a competitive tournament of games played between the controllers in an evolving population. Robot controllers were selected based on whether they won or lost games in the course of a tournament. Following a tournament, the neural controllers were ranked competitively according to how many games they won and the population was propagated using a mutation and replacement strategy. After several hundred generations, the best performing controllers were transferred to teams of real mobile robots, where they exhibited behaviors similar to those seen in simulation including basic navigation, the ability to distinguish between different types of objects, and goal tending behaviors.

Keywords: Evolutionary robotics; Robot colonies; Mobile robots; Evolutionary neural computing; Behavioral robotics

1. Introduction

The fundamental goal of evolutionary robotics (ER) is to apply evolutionary computing methods to automate the production of complex behavioral robotic controllers. The study of behavioral robotic controllers that produce complex robot–environment interactions has been an area of particular interest in recent years. Many proof-of-concept experiments in the field of evolutionary robotics have been carried out in the last decade. Although much of this work was done using computer-based simulations only [1–6], some work has used real robots. Two examples are the evolution of walking behaviors in hexapod and octopod robots

* Corresponding author. Present address: Computer Science and Engineering, University of South Florida, 4202 E. Fowler Ave. ENB 342, Tampa, FL 33620-5399, USA. Tel.: +1-813-546-4515; fax: +1-813-974-5456.
E-mail address: aanelson@csee.usf.edu7 (A.L. Nelson).

0921-8890/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.robot.2004.01.001

[7,8], and the evolution of simple behavioral controllers for small mobile robots in closed environments [9,10]. These include the development of phototaxis behaviors [11,12] and of object avoidance and navigation [10,13] in small robots using differential steering.

The work described in this paper attempts to move evolutionary robotics research beyond this nascent stage. Experimental results are generated with a new evolutionary robotics research testbed that includes a colony of mobile robots, a coupled computer-based simulation environment, and evolutionary neural network controllers. The evolved neural controllers rely on processed video images rather than sonar or IR detectors. During evolution, controllers were selected based on their relative abilities to compete against each other in tournaments of robotic games. The experiments presented in this work show that it is possible to evolve moderately complex mobile robot controllers in simulation using tournament selection methods and to transfer these controllers to real robots.

The field of evolutionary robotics has been reviewed in recent publications [13–17]. Several important issues raised in this literature include: (1) the application of ER methods to more sophisticated problems; (2) methods of performance evaluation; (3) embodied evolution in real robots versus evolution in simulation; (4) the coupling of simulation to reality. In this work we will focus on the first two issues.

1.1. Evolutionary versus knowledge-based autonomous robot control: Why pursue ER?

There is a plethora of recent work in which particular elements of controller design have been automated (automatic engineering). ER as a field is distinguished from these methods in that the entire controller structure is developed using evolutionary methods. This is a process of primary synthesis of a behavioral controller rather than the optimization of an existing structure.

Many researchers in the field of ER have stated that ER is a new and potentially powerful method of mobile robot controller development. Although we agree that ER has great potential, it must be pointed out that ER has not as yet produced results that are competitive with modern knowledge-based autonomous robot controllers. In fact, we believe that ER research will follow a long and an arduous path before the field sees a significant amount of real world application. There are several factors that make ER very potent in the long term. These include: (1) the ability to automate controller design; (2) the ability to produce controllers for uncharacterized behavioral domains. Because of the latter, it may be possible to use ER methods to develop robot controllers for systems in which insufficient information exists to formulate a traditional knowledge-based controller.

1.2. The future of evolutionary robotics: fitness evaluation in evolving populations of robot controllers

One of 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 robot behaviors. In turn, a key issue related to the successful evolution of complex behaviors is the specification of a training fitness function or objective function.

One common method used to address the problem of evolution of more complex behaviors is incremental evolution. In incremental evolution, fitness functions select for simple behaviors in the early stages of evolution and for more advanced behaviors as the evolution proceeds [5,7,13]. The main criticism of incremental evolution and related methods has been that they require the designer to decide which behaviors are basic to the final complex behavior sought. This limits the potential for application to uncharacterized behavioral domains. Direct evaluation by humans (sometimes termed ‘breeder selection’) has been used in some ER work [13,19,20]. These methods limit the automation aspect that is central to ER because they require the constant attention of a human designer during training.

Fitness function definitions must contain some task-specific information from the designer, i.e., the designer must state what the task is. However, for complex behaviors, simply stating the task in a formalized way is not generally sufficient to select for the ability to perform that task. Most fitness functions used in current ER research contain information pertaining both to the task outcome and how to perform the task. In almost every case seen in the literature, some form of hand designed task-specific absolute fitness function is used to evaluate and select robot controllers during the course of evolution. For very

simple behaviors, especially those involving only one or a few sub-behaviors, stating the task becomes equivalent to stating *how to accomplish* the task. This is a subtle point but it has profound implications for the long-term application of current ER methods. This simple duality case has allowed researchers to investigate proof-of-concept ER behavioral problems, but has not provided for the viable extension of these methods to more complex domains. The success of most of the results reported in the ER literature relies heavily on the simplicity of the tasks investigated. In this work, the claim is made that specification of training fitness functions in the simple case is dissimilar to the specification of fitness functions for the complex case to the degree that most current ER results are not extensible to non-trivial problems.

1.3. Using competitive tournaments to evaluate the relative fitnesses of robot controllers

The partial or complete automation of fitness function (metric) specification would profoundly change the field of evolutionary robotics. In this work, we investigate a form of reinforcement learning that makes use of competitive tournaments of games played by individuals in a population of neural controllers.

Many games requiring high levels of skill can be scored in a tournament using relatively simple and deterministic metrics (measures). Such methods have yielded impressive results in the evolution of Checkers playing neural networks [21] and of Go playing neural networks [22] in the field of computer science. In cases where at least one team or player of an evolving population achieves a win in a tournament, metric complexity can be reduced further to best number of games won in a tournament. The research described in this paper applies competitive tournament selection methods to the complex domain of behavioral robotics.

In the broader context of evolutionary computation, various competitive and co-competitive evolution strategies have been investigated [30,31]. However, at the time of this research, very few instances of relative competitive fitness selection have been applied to the generation of monolithic behavioral controllers for physically embodied agents. In ER there are several example of co-evolution of two controller species [13,32,33]. These generally involve a predator population co-evolving with a prey controller population. In those cases, the controller selection is co-competitive in that the fitness of predators may affect the fitness evaluations of prey controllers and vise versa. However, in those cases, the fitness metrics for the individual species have been absolute rather than relative. ER differs significantly from other evolutionary computing endeavors because the evolution is focused on primary controller synthesis based on behavior. In almost every case, mappings between the desired behavior, and sensor input and actuator outputs are unknown. Hence, training in ER cannot make use of typical error metrics based on I/O training data sets. Controllers must be evolved to mediate a complex sensor–actuator–environment feedback loop based only on features of expressed behavior. The appropriate input–output mappings are not known, and in fact could remain uncharacterized even after successful evolution.

2. A competitive tournament based multi-robot evolutionary robotics research environment

This section describes the evolutionary robotics research platform used in this research. The main components of this testbed are: (1) an evolutionary artificial neural network application, (2) a colony of physical autonomous mobile robots and environment, (3) a vision-based range-finding sensor emulation system, (4) a simulation and evolutionary training environment. We will focus in detail on the neural network and genetic algorithm formulations. The hardware and the vision-based sensor systems have been described in [23] and will be only briefly described here.

2.1. The evolutionary neural network architecture

Neural networks are by far the most commonly used controller structures in ER. This is mainly due to their flexibility and their close association with the research field of evolutionary computing. In the early days of ER research, initial work was done using a variety of evolvable controller architectures. These include evolvable state transition structures, evolvable hardware [13], and genetic programming [29]. Genetic programming is still used in a small proportion of ER research [18]. However, even though neural networks have come to dominate the field of ER, little if any

work has been done exploring the relative suitability of different architectures. We use neural networks in this work because they afford a real-valued variable dimension search space that is well suited to stochastic search methods.

Because the dynamics of behavioral robotics tasks are not well characterized, we believe it is important to allow a neuro-evolution process to use a very broad network morphology search space. Much of the ER work to date used very simple network topologies and restricted weight values [10,12,24,25]. Such restrictions limit the scalability of the methods studied. Other researchers have used more complex networks [8,13,26] and we pursue this path. 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 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.

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 structure **N** with one formatted field per neuron. 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 and progressing into the past with successive rows. The maximum level of time delay supported by a network is specified by a scalar integer, δ. 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. Also, the formulation supports efficient addition or removal of neurons and connections without disruption of the network's overall connectivity relationships.

Neuron activation functions take the form

$$f_n(u) = f_n(\mathbf{w}_n, \mathbf{i}(t, \tau(n))), \tag{1}$$

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

$$u = \sum_{m=1}^{N+1} w_m i_m, \tag{2}$$

where $N + 1$ is the width of **W** and of **I**. These activation function types include sigmoid, linear and step functions. Note that bias inputs are accounted for 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 Euclidean distance between **w** and **i** in n-space.

Network inputs are considered to be linear neurons with all zero connecting weights except for a single self-connection with a unit weight. 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

$$\mathbf{I}(t+1) = \mathit{Network}(\mathbf{I}(t), \mathbf{N}, \mathbf{W}), \tag{3}$$

and

$$\mathbf{o} \subset \mathbf{i}(t+1, 1), \tag{4}$$

where **o** is a vector of values from specified output neurons and is a subset of $\mathbf{i}(t+1, 1)$, the first row of the new $\mathbf{I}(t+1)$.

Fig. 1 shows the graphical representation of several members of a population of heterogeneous networks.

2.2. The EvBot platform and environment

Our laboratory has recently developed a new, computationally powerful colony of small mobile robots named EVolutionary roBOTs (EvBots). Each robot in the colony is fully autonomous and capable of performing all control computing and data management on board. The robots use a PC-104 X86 compatible 133 MHz computer architecture and run a custom distribution of the Linux operating system. The full

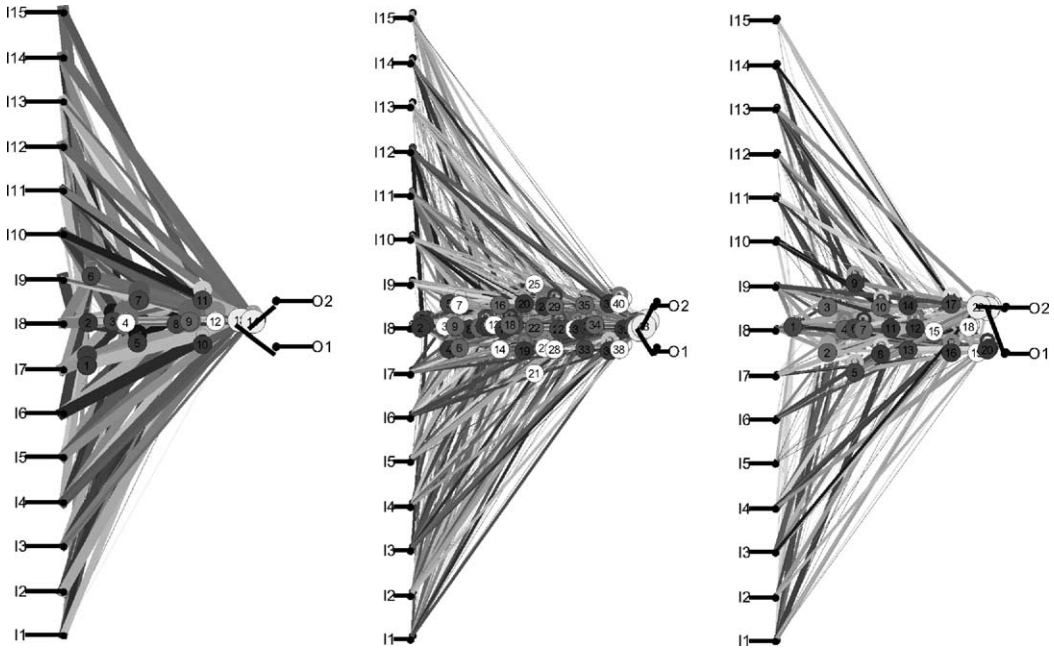


Fig. 1. Several example networks from a heterogeneous population of neural networks initialized for an evolutionary training.

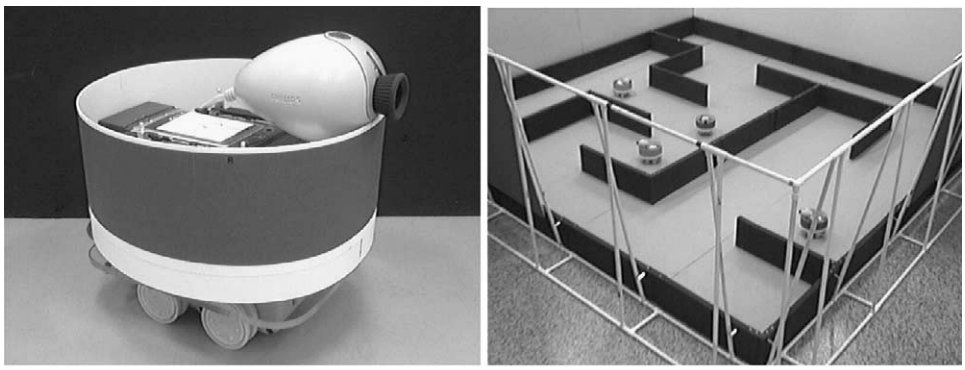
hardware and computer architecture of these robots is described in [23,27].

A physical reconfigurable maze environment was constructed for the mobile robot colony. Robots and other objects in the environment were fitted with colored skirts to facilitate color vision sensing of the environment. A fully assembled EvBot and the physical environment are shown in Fig. 2.

2.3. Video range-finding emulation sensors

Each robot is fitted with a small video camera. Images captured from the video cameras are processed into range data before being feed into the neural controllers.

The vision systems on the robots use fixed geometric properties within the physical maze environment to



(a) (b)

Fig. 2. Photographs of a fully assembled EvBot (a) and the physical robot maze environment containing several robots (b).

calculate the ranges and angles of walls and objects. This is done by identifying the pixel colors and regions and then calculating the vertical sum of pixels in the regions. For an object of known fixed size, a simple formula can be used to calculate the distance of the object that produced the colored region in the original video image based on the calculated pixel sum. The vision system can detect five object (material) types. These are walls, red robots, green robots, red goal objects and green goal objects. For each detectable object type, range and angle values are reported over a spread of 48° centered on the forward direction of the robot body frame of reference. A vector of range values is produced for each object type. Angular data is implicitly encoded in the sequential left-to-right order of range values reported in each range data vector.

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.

2.4. The coupled simulation environment

Evolution of the neural controllers is performed in a simulated environment coupled to the real robot–environment. Robot agents, sensors, and robot–environment and robot–robot interactions are modeled. Evolved controllers can be transferred directly to the real robots without alteration of the controller portion of the code.

Fig. 3 shows the two views of the simulation environment. In Fig. 3 of panel (a) the black lines represent walls, the smaller circles represent the robots, and the larger circles represent the stationary goals. Panel (b) shows a graphical representation of simulated range sensor data received by the robot agent in the lower left corner of the environment.

3. The competitive tournament evolutionary algorithm

In this section we present the details of the genetic algorithm and training fitness functions used to evolve the game playing controllers discussed in this research.

3.1. Performance function specification in evolutionary robotics

In ER, fitness for selection is most often measured using hand-formulated parametized absolute real-valued functions to be maximized during training. It is arguable that for many complex behaviors, the knowledge required to specify an adequate absolute training fitness function is equivalent to that would be required to design a rule/knowledge-based controller to perform these same behaviors.

Tournament ranking evaluation partially eliminates the need to specify a fully domain-specific fitness function. As long as the problem can be formulated into a game that is either won or lost, other details about the game need not be included in the fitness

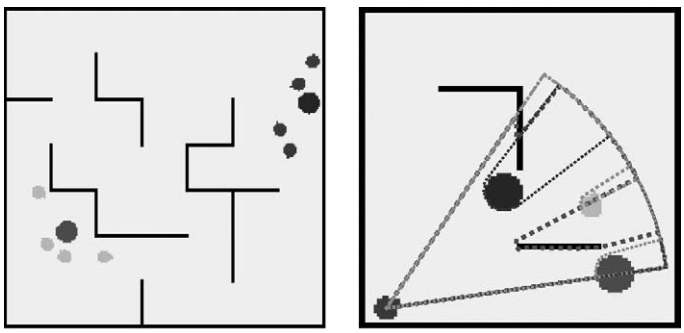


Fig. 3. Graphical representations of the simulation environment. In (a) robots are shown clustered around their respective goal objects. Panel (b) shows a graphical representation of simulated range sensor data received by the robot agent in the lower left corner of the environment.

function definition. Agents in an evolving population that receive higher relative competitive rankings in a tournament of games will be propagated preferentially over agents receiving lower rankings.

Below, a general formulation of the tournament competitive fitness selection function used in this work is given. This fitness function is designed to be useful for team games that can be formulated to produce a win–lose outcome. These would include games like soccer and robot tag. Many useful real world behaviors such as mine sweeping and group searching behaviors in unknown terrain can be also formulated into scorable team games.

The training fitness function can be reduced to two overall parts: (1) select for controllers that win more competitive games, (2) identify and select against pathological controller morphologies. This function generates an intra-population competitive ranking similar to that used in the evolutionary neural computing work in [21] rather than the more common co-competitive absolute fitness rankings used in [33]. This was done to take advantage of competition to drive a single population toward continual improvement.

Each generation in the evolutionary process consisted of a tournament of games played between the robot teams using neural controllers in the evolving population. Robot controller selection was based on whether controllers won or lost games in the course of a tournament. Controllers that won more games were deemed to be fitter. Following a tournament, the fittest neural controllers were selected. Mutated versions (offspring) of these fittest controllers then replaced the least fit members of the population to make the next generation's population.

Because we wanted to rely on tournament selection as much as possible, only pathological cases that were known to lead to early and catastrophic stagnation of the evolutionary process were actively selected against. The fitness function selected against two pathological controller behaviors. The first behavior was the production of constant reverse wheel speeds in one or both wheels throughout the course of a game. The second pathological behavior was that of becoming stuck and remaining stuck for the duration of a game. For purposes of formalization, we will indicate these two cases as Boolean functions.

A population **P** of evolving robot controllers consists of a fixed number P of neural networks. At each generation, a tournament of competitive games is played to evaluate each controller's relative fitness within the population. At the beginning of a tournament, a set of game starting positions for robot teams and goals is quasi-randomly generated. These beginning positions are then used for every game in that tournament. Each controller in the population **P** plays against every other controller in **P** (complete tournament). For every pair of controllers in the population, two games are played. In the first game the first controller is used in the first team of robots and the second in the second team of robots. In the second game, the controllers are switched. The scoring of the controller population members makes use of the relative scores of each such pair of games. This eliminates any advantage one team of robots may have incurred due to the environment and to initial conditions.

A generalized form of the tournament performance evaluation function can be written as

$$F(p) = w + d + n, \tag{5}$$

where w, d and n are the functions evaluating the contributions of games won, games played to a draw and expression of pathological behavior respectively during a tournament. $F(p)$ gives the fitness of the pth controller of the population **P**.

The relative fitness of the robot controllers playing in one game is dependent on the outcome of a reciprocal paired game in which the starting positions of the controllers are reversed. We will denote these paired games as g and g'. Using these paired games we break the game wins into three classes. In class 1, a particular controller wins both games g and g'. Games of class 2 are those in which one controller wins one of the games but plays the other to a draw. In class 3, one controller wins one game but loses the other.

The points awarded to the pth controller for number of wins in the tournament (the w sub-function of Eq. (5)) can be broken down as follows:

$$w = aG_1 + bG_2 + cG_3, \tag{6}$$

where G_1, G_2, and G_3 denote numbers of games won of each of the three classes and a, b, and c are the scalar weighting factors. The values of a, b, and c are generally set so that

$$a > b > c > 0,$$

since a controller that can win from both of the starting positions of g and g' is better than one that can only win one of the two games.

Points are also awarded to a particular controller depending on the number paired of games played to a 'better' draw. These are defined as games one of the teams is closer to winning the game in both of the games g and g'. The d sub-function of Eq. (5) becomes

$$d = \delta D_1, \tag{7}$$

where D_1 denotes number of games played to a better draw and δ is a scalar weighting factor. δ is set to be much less than a, b, or c so that results related to numbers of wins dominate the tournament selection process. The sub-function d from $F(p)$ in Eq. (5) is included because it is possible to have tournaments with very few wins, especially early in training.

The n sub-function of Eq. (5) selecting against pathological behaviors can be expanded as

$$n = \alpha B_1 + \beta B_2, \tag{8}$$

where B_1 and B_2 are the Boolean functions denoting the presence (1) or lack (0) of expression of each of the pathological behaviors in the current tournament (these were defined above as continual backward motion and becoming stuck, respectively). α and β are the scalar weighting factors. The values of α and β are generally set to be large negative values in relation to a, b, and c so there is a heavy selective pressure against these behaviors even if they result in wins.

3.2. Neural network genetic representation and mutation

As noted in the section describing the neural network, all connectivity and weighting information required to specify a particular network is stored in a matrix of real numbers and a vector structure of neuron properties. These two objects are acted upon by the genetic algorithm directly and can be thought of as the genetic material of a network. There is no encoding (such as a binary encoding), direct or otherwise beyond this level. The genome $\mathbf{C}$ is then specified by the two-dimensional matrix or real numbers

$$\mathbf{C} = [\mathbf{W} : \mathbf{N}'], \tag{9}$$

where $\mathbf{W}$ is the weight/connection matrix, and $\mathbf{N}'$ is a two column matrix extracted from the formatted structure $\mathbf{N}$ specifying the type and associated time delay of each neuron (two integers for each neuron).

During evolution, networks are mutated in three ways. First, the non-zero real-valued elements in the weight matrix can be perturbed. Second, connections can be added or removed by changing zero elements to non-zero values and vice versa. Finally, neuron units can be added or removed by inserting or deleting paired rows and columns of $\mathbf{C}$.

Mutation of a network selected for inclusion in the next generation population can be formalized by the compound relation

$$\mathbf{C}' = M_s(M_c(M_w(\mathbf{C}))), \tag{10}$$

where $\mathbf{C}$ is the chromosome of the parent network and $\mathbf{C}'$ the resulting mutated offspring network chromosome. M_w, M_c, and M_s are the 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. Each of the three mutation operators takes a chromosome as an input and produces an altered chromosome as output. Their general form is

$$M(\mathbf{C}) = \begin{Bmatrix} mutate(\mathbf{C}) & \text{if } R < m\_rate \\ \mathbf{C} & \text{if } R \geq m\_rate \end{Bmatrix}, \tag{11}$$

where $mutate(\cdot)$ is a function that performs the appropriate alteration on $\mathbf{C}$, R is a number from a uniform random distribution between 0 and 1, and m_rate is the probability threshold associated with that type of mutation. R is updated with a new random value each time a mutation operation is performed. Mutation is the only genetic operator used. Crossover was not employed. There has been some work that indicated that crossover is not beneficial for evolutionary neural computing applications.

In this research, the probabilities for invocation for the genetic operators M_w, M_c, and M_s were set to 0.9, 0.5, and 0.5 per generation, respectively. In the case that the mutation operator was invoked, the mutation rate was initialized to 0.1 (10% of the weights in the network on average would be randomly selected and mutated). The mutation magnitude was initialized to 2 (mutated weights would be perturbed random values between -2 and 2). New connection weights associated with newly inserted neurons or connections were initialized to values between -0.5 and 0.5.

3.3. The evolutionary training algorithm

Populations of fixed size P were evolved using an evaluation, mutation, and replacement scheme. After a tournament of games (one generation), controller population members p were scored relative to each other using the performance metric $F(p)$ defined in Eq. (5) of Section 3.1. The population $\mathbf{P}$ is always ordered from fittest to least fit before propagation to the next generation.

The next generation population $\mathbf{P}\_next$ is constructed from the union of the following three sets derived from the current (parent) population:

$$\mathbf{P}\_next = \{p_1 \cdots p_m\} \cup \{p'_1 \cdots p'_m\} \cup \times \{p_{m+1} \cdots p_{P-2m}\}, \quad (12)$$

where $p_m \in \mathbf{P}$ is the mth individual of the current (parent) population $\mathbf{P}$, p'_m is a mutated version of p_m, and P is the fixed population size. The result of Eq. (12) is that m of the fittest controllers are transferred un-changed to the next generation. This same fraction of the controller population is mutated to produce offspring using Eq. (10) and added to the next generation. The remainder of the next generation population is made up of the fittest remaining members of the current population. Values for m and P are selected by the user and reflect a trade-off between evolutionary speed and chaos during training. For example, an $m/P = 1/3$ ratio gives a replacement rate of 33.3%. The evolutionary algorithm described in Eq. (12) is a form of greedy mutation-only $(\mu + \lambda) - \mathrm{ES}$ with incomplete replacement [28].

Although this algorithm is technically a greedy one, the game environment initialization for each tournament affects the outcome of the games to such a degree that the fittest member of the population could be eliminated. This adds a high degree of probabilistic selection to the algorithm.

4. Results

In this section, we present initial results and tests of one population of robot controllers evolved to play robot *Capture the Flag*. In this game, there are two teams of robots and two goal objects. All robots on team #1 and one of the goal objects are of one color (red). The other team members and their goal object are of another color (green). In the game, robots of one team must try to come within a certain distance of the other team's goal object while protecting their own. The robot which first comes within one robot body diameter's distance of an opponent's goal wins the game for its team. The best evolved controllers were able to play and win competitive games both in the simulated environment and in the physical environment using the real robots.

4.1. Experimental setup

We will focus on an evolved controller that displays two identifiable sub-behaviors: wall avoidance and selective avoidance of one's own goal. The controller was evolved in a population of size $P = 6$ for 366 generations. The population replacement rate was set to 50% per generation. Several populations were evolved with these parameter settings and found by subjective observation to be able to perform competitively. However, due to the high level of computation and physical setup time involved, the experiments detailed below were only conducted on one of the evolved populations.

The parameters relating to the performance metric $F(p)$ of Eq. (5) used in this training evolution are given in Table 1. The resulting overall tournament score for a particular controller is the sum of all points awarded for each game during the tournament as discussed in Section 3.1. Although a range of parameter settings were tested, insufficient data exists to perform a statistical sensitivity analysis on the parameter settings. In fact, a common feature of many genetic search spaces that they are computationally intractable with respect to a direct search and individual searches (evolutions) are too time consuming to allow for the statistically optimization of search parameters.

Table 1
Fitness function parameter settings used during evolution

Parameter	Game case description	Value (points awarded)
a	Win–win	20
b	Win–draw	15
c	Win–lose	10
δ	Best draw	2
α	Backward motion	−10
β	Stuck	−2

In all cases, robot games played with the real robots in their physical maze environment were recorded by collecting a sequence of video images from a camera mounted directly above the maze. These video sequences were then processed to extract the movements of the robots during the course of the game(s).

4.2. Behavior of evolved robot controllers in simulation and in reality

In order to demonstrate the functionality of the evolved robot controllers, sets of games were played both in simulation and then using real robots in the real maze environment. Using the fittest evolved neural controller from the population, two sets of games were played: one set in simulation, and the other in the real environment using real robots. Each team consisted of two robots.

Sets of initial game positions were automatically generated based on the first nine random seed states of the MATLAB random number generator. These initial positions were used in the simulated games and then this same set of nine initial positions was used in the games played with the real robots in their physical maze environment. The maze environment was configured similarly in both the simulated and real worlds. A game was allowed to continue for 80 time steps before being terminated. Games in which all robot agents on both teams became stuck were also halted. In these cases, the game was given to the team that was closest to its opponent's goal at the termination of the game.

Fig. 4 shows the results of nine consecutive simulated games where each robot agent was controlled by the fittest member of the evolved population of controllers. The final positions of the robots are shown, and the path taken by each robot is indicated by a curve. The darker lines indicate the paths followed by the team 1 (red) robots while the lighter lines indicate the paths followed by the team 2 (green) robots.

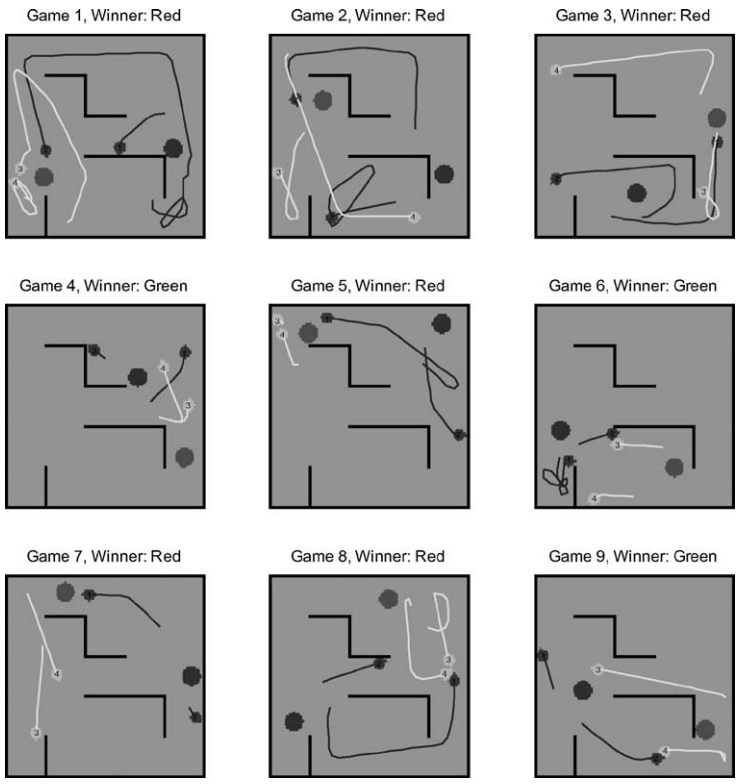


Fig. 4. Nine simulated games played with robot agents using the single best evolved controller from a population. Team 1 robot paths and goal object are indicated by a dark line and circle, while team 2 paths are indicated by light lines.

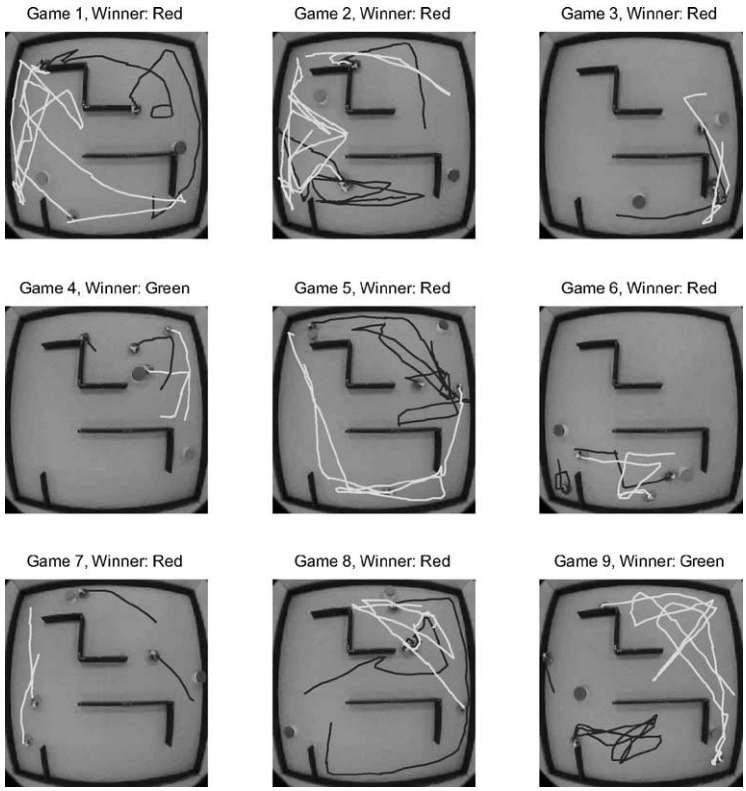


Fig. 5. Nine games played with real robots in the physical environment. The initial conditions of the games match those of the simulated games. Team 1 robot paths and goal object are indicated by a dark line and circle, while team 2 paths are indicated by light lines.

The larger circles represent the stationary goals (flag). Robots start each game close to their own goal.

Fig. 5 shows the results of the nine consecutive games when played with real robots in the physical environment. The initial positions of the robots and goals of the physical games corresponded to those used in the simulated games. The same best performing evolved neural controller as was used in the simulated game set (from the particular evolved controller population) was transferred to each of the real robots. These robots are fully autonomous: all sensor data, image processing, and neural net computing is done onboard each of the robots. The data were generated by collecting sequences of images using a camera mounted over the physical environment. The image sequences were processed to extract each robot path. The paths were then superimposed on the final image of each game to produce a graphic similar to that presented for the simulated games.

The results of the simulated and real game sets are presented in tabular form in Tables 2 and 3.

Even though the outcome of a game played in simulation is likely to have the same outcome in reality (88.8% similarity), it is clear from the data presented in Figs. 5 and 6 that the exact behavior and particular

Table 2
Simulated game results for each of the random game initializations

Game initial state	Game duration (time steps)	Winner
1	59	Team 1 (red)
2	25	Team 1 (red)
3	21	Team 1 (red)
4	7	Team 2 (green)
5	21	Team 1 (red)
6	33	Team 2 (green)
7	9	Team 1 (red)
8	33	Team 1 (red)
9	14	Team 2 (green)

Table 3
Real robot team game results for each of the random game initializations

Game initial state	Game duration (time steps)	Winner
1	80	Team 1 (red)
2	80	Team 1 (red)
3	12	Team 1 (red)
4	10	Team 2 (green)
5	45	Team 1 (red)
6	11	Team 1 (red)
7	8	Team 1 (red)
8	48	Team 1 (red)
9	70	Team 2 (green)

paths followed by robots in simulation generally deviate from those obtained in the real world. This may reflect the possible unequal transference of subtle abilities with regard to robot performance. However, the same basic evolved abilities are observed in simulated and real games. Comparison of the exact paths followed in simulation and in reality is not considered to be the best metric of transference quality. The reactions of the trained neural controllers are highly non-linear. Deviations resulting from slight variations in starting positions or sensor inputs compound with each time step so that even a small difference will result in divergence in a relatively short time.

4.3. Measuring the performance of evolved controllers

In this section we will present experiments aimed at measuring both the quality and the character of the evolved controller used in the previous section's experiments.

The evolved neural controller was tested in a set of real robot games against hand coded rule set knowledge-based controllers designed to play robot *Capture the Flag*.

Two knowledge-based controllers were developed. The first was designed to be a difficult opponent to beat and made use of both temporal and spatial information to avoid walls, extract itself from corners, avoid team mates, block opponents and home in the opponent's goal. In addition, this controller could determine if it was stuck on an object outside its sensor field of view and extract itself. The second controller was designed to be a poor player and produced random wheel speed commands at each time step that have no relationship to its sensor inputs. This poor-performing controller was included in these tests to provide a minimum performance baseline reference. In the absence of any viable competition, a controller producing random speed commands may eventually happen upon their opponent's goal (flag) by chance. It is important to show that the evolved controllers perform better than

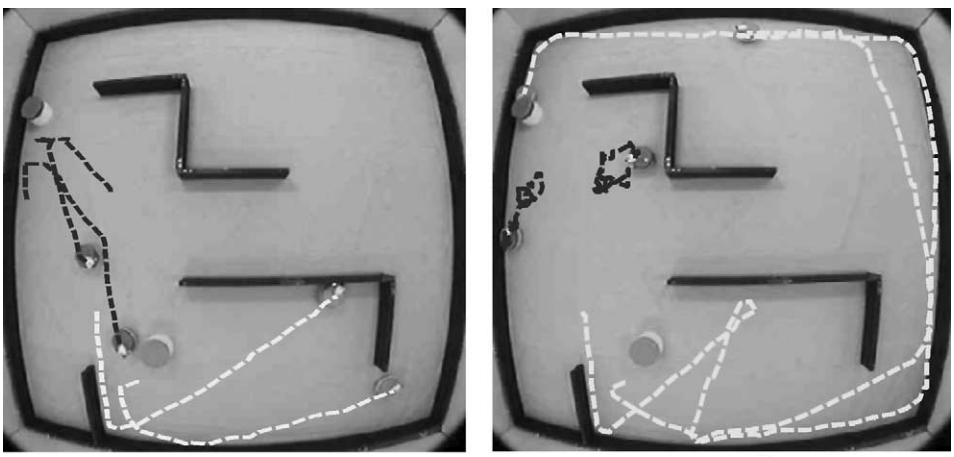


Fig. 6. Example games played between trained neural network controllers (lighter dotted lines) and rule-based controllers (dark dotted lines). In (a) good rule-based robots beat neural network controllers while in (b) neural controllers eventually beat random controllers starting from similar initial conditions.

Table 4
Results of the set of 10 games played between the evolved neural network controller and the hand coded rule-based (good) controller

Game	Random initialization	Team 1	Team 2	Winner	Time steps
1	1	Neural	Good rule	Neural	14
2	1	Good rule	Neural	Good rule	20
3	2	Neural	Good rule	Neural	16
4	2	Good rule	Neural	Good rule	39
5	3	Neural	Good rule	Good rule	36
6	3	Good rule	Neural	Good rule	28
7	4	Neural	Good rule	Good rule	114
8	4	Good rule	Neural	Neural	31
9	5	Neural	Good rule	Good rule	13
10	5	Good rule	Neural	Good rule	27

random chance. These test reference controllers will be referred to as the 'good rule-base' and 'random' controllers, respectively. In both cases, sensor inputs and motor output ranges were restricted to those allowed in the evolved neural network controllers.

The evolved neural controller competed in a series of real games against both the good rule-base and the random controllers. This was done to rank the quality of the evolved controller on a continuum including a good controller and a very poor controller.

Ten games were played between the evolved neural network and the good rule-based controller. Game initial positions may give one or the other of the competing teams an advantage. For this reason, a comparative set of games must contain two games for each starting configuration used. In the first, each team is in a particular initial position, and in the second, the two team's starting positions are swapped. For these games, five initial game positions were generated. Since two games are played for each staring position, the total is 10 games.

A similar set of 10 real games was played between the evolved neural network and the random (poor) controller. Again, the same set of five game initializations was used to conduct a set of 10 paired reciprocal games.

Tables 4 and 5 give the results of the games involving the good rule-base and the random controller, respectively.

Summarizing these results, we find that the neural network controllers won 3 out of 10 games against the good rule-based controller, or 30%. On the other hand the good rule-base won 7 out of 10 or 70% of its games. All of the games between the neural network and the rule-based controller were played to completion. The neural network controllers won 8 out of 10 against the random controller, or 80%. The random controller was not able to win any games. In this case

Table 5
Results of the set of 10 games played between the evolved neural network controller and the hand coded random (poor) controller

Game	Random initialization	Team 1	Team 2	Winner	Time steps
11	1	Neural	Random	Neural	16
12	1	Random	Neural	Neural	67
13	2	Neural	Random	Neural	17
14	2	Random	Neural	None	Incomplete
15	3	Neural	Random	Neural	81
16	3	Random	Neural	Neural	38
17	4	Neural	Random	Neural	26
18	4	Random	Neural	Neural	30
19	5	Neural	Random	None	Incomplete
20	5	Random	Neural	Neural	18

two of the games were not completed because all the robots became stuck before the end of the game.

Fig. 6 shows the two example game results from the above tables; these are games 2 and 12. The robots are shown in their final end-game positions. The dotted lines indicate the courses of the robots during each game. In the first game (Fig. 6(a)), neural network controllers (green, lighter dotted lines) compete against good rule-based controllers (red, dark dotted lines). In the second game (Fig. 6(a)), neural network controllers (green) compete against poor random controllers (red). In the first game, the rule-based robots reach the green goal before the neural network based controllers can find the red goal. On the other hand, in the second game, the poorer random controllers are not able to make progress toward the green goal and the neural network based controllers eventually find the red goal and win the game.

These results imply that the functional quality of the evolved controller is somewhat less than that of the hand coded rule base. This is compared to the base line negligible abilities of the random controller. The evolved controller was able to beat the random controller in every game played to completion. It should be noted that identical or equally matched controllers would receive the same number of wins when competing against one another in a set of reciprocal games. For example, the rule-based controller would receive 5 out of 10 wins on average when played against a copy of itself, or 50%. Also, the rule based controller wins against the random controller 100% of the time (data not shown).

As noted in Section 1, evolved behavioral robotics control systems do not yet rival well designed sophisticated knowledge-based controllers. Nonetheless, these results indicate that in this case, an evolved controller can beat a hand coded controller a fraction of the time.

5. Conclusions and future research

In this paper, a new evolutionary robotics environment was described and initial experimental results were presented. This research testbed was developed to address several current issues in the field of ER. These include questions about the extension of current ER methods to more complex problems. In particular, the feasibility of using evolutionary computing methods to evolve teams of mobile robots to play competitive games was investigated. A tournament training performance evaluation function was implemented. This fitness function was used to evolve controllers for teams of robots to play a benchmark competitive game, *Capture the Flag*. The fitness function was not based on game specific factors and could be used on other multi-robot tasks that can be formulated into competitive games. The use of competitive performance evaluation allows for the improvement of behavior without the need for an absolute performance measure.

This work will be extended by applying the competitive relative performance metric to other related mobile robot behaviors and by investigating the related training dynamics. We will investigate the possibility of improving training measures without adding more task-specific information. Alterations of the training metric could include the weighting of some tournaments more highly than others. Tournaments in which some controllers win many more than others carry more fitness information than tournaments in which all controllers win nearly the same number of games. It may also be of interest to investigate the effects of game initialization on controller evolution. This work used random game initializations for each tournament. Another approach would be to select several game starting configurations and use only these. This method would run the risk of controllers learning environment specific behaviors that would not generalize well but could reduce the negative effects of poor game initializations that result in equal relative scores for all controllers and thus generate no selective pressure.

References

[1] F. Gomez, R. Miikkulainen, Incremental evolution of complex general behavior, Adaptive Behavior 5 (1997) 317–342.
[2] J. Xiao, Z. Mickalewicz, L. Zhang, K. Trojanowski, Adaptive evolutionary planner/navigator for mobile robots, IEEE Transactions on Evolutionary Computing 1 (1) (2000) 18–28.
[3] 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, 2001, pp. 357–366.
[4] 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.
[5] 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 9 (5) (1998) 796–812.
[6] M. Potter, L.A. Meeden, A. Schultz, Heterogeneity in the coevolved behaviors of mobile robots: the emergence of specialists, in: Proceedings of the 17th International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 2001.
[7] D. Filliat, J. Kodjabachian, J.A. Meyer, Incremental evolution of neural controllers for navigation in a 6 1egged robot, in: Sugisaka, Tanaka (Eds.), Proceedings of the Fourth International Symposium on Artificial Life and Robotics, Oita University Press, 1999.
[8] 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, 1998, pp. 39–58.
[9] D. Floreano, S. Nolfi, F. Mondada, Competitive co-evolutionary robotics: from theory to practice, in: R. Pfeifer, From Animals to Animats 5, Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior (SAB'1998), MIT, Cambridge, MA, 1998.
[10] D. Floreano, F. Mondada, Evolution of homing navigation in a real mobile robot, IEEE Transactions on Systems Man, and Cybernetics. Part B: Cybernetics 26 (3) (1996) 396–407.
[11] N. Jakobi, P. Husbands, 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, Proceedings of the Third European Conference on Artificial Life, Lecture Notes in Artificial Intelligence, vol. 929, Springer, 1995, pp. 704–720.
[12] R.A. Watson, S.G. Ficici, J.B. Pollack, Embodied evolution: distributing an evolutionary algorithm in a population of robots, Robotics and Autonomous Systems 39 (1) (2002) 1–18.
[13] I. Harvey, P. Husbands, D. Cliff, A. Thompson, N. Jakobi, Evolutionary robotics: the Sussex approach, Robotics and Autonomous Systems 20 (2–4) (1997) 205–224.
[14] M. Mataria, D. Cliff, Challenges in evolving controllers for physical robots, Robotics and Autonomous Systems 19 (1) (1996) 67–83.
[15] S. Nolfi, D. Floreano, Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-organizing Machines, MIT, Cambridge, MA, 2000.
[16] H. Lund, J. Hallman, Evolving sufficient robot controllers, in: Proceedings of the IEEE International Conference on Evolutionary Computation, 1997, pp. 495–499.
[17] D. Floreano, J. Urzelai, Evolutionary robots: the next generation, in: Proceedings of the Seventh International Symposium on Evolutionary Robotics (ER2000), From Intelligent Robots to Artificial Life, AAI Books, 2000, pp. 231–266.
[18] W. Lee, Evolving complex robot behaviors, Information Sciences 121 (1–2) (1999) 1–25.
[19] H.H. Lund, O. Miglino, L. Pagliarini, A. Billard, A. Ijspeert, Evolutionary robotics—a children's game, evolutionary computation, in: Proceedings of the IEEE World Congress on Computational Intelligence, 1998, pp. 154–158.
[20] F. Kaplan, P. Oudeyer, E. Kubinyi, A. Miklosi, Robotic clicker training, Robotics and Autonomous Systems 38 (3–4) (2000) 197–206.
[21] K. Chellapilla, D.B. Fogel, Evolving an expert checkers playing program without using human expertise, IEEE Transactions on Evolutionary Computation 5 (4) (2001) 422–428.
[22] 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, 2001.
[23] J. Galeotti, The EvBot: a small autonomous mobile robot for the study of evolutionary algorithms in distributed robotics, MS Thesis, North Carolina State University, 2002.
[24] F. Southley, F. Karray, Approaching evolutionary robotics through population-based incremental learning, in: Proceedings of the IEEE Conference on Systems, Man, and Cybernetics, vol. 2, 1999, pp. 710–715.
[25] S. Nolfi, Evolving non-trivial behaviors on real robots, Robotics and Autonomous Systems 22 (3–4) (1997) 187–198.
[26] F. Gruau, Automatic definition of modular neural networks, Adaptive Behavior 2 (1995) 151–183.
[27] J. Galeotti, S. Rhody, A. 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 on Computer Applications in Industry and Engineering (CAINE-2002), San Diego, CA, November 7–9, 2002, pp. 86–91.
[28] I. Ashiru, C.A. Czarnecki, Evolving communicating controllers for multiple mobile robot systems, in: Proceedings of the IEEE International Conference on Robotics and Automation, vol. 4, 1998, pp. 3498–3503.
[29] 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, MIT, Cambridge, MA, 1992, pp. 110–119.
[30] D.B. Fogel, Evolutionary Computation, Toward a New Philosophy of Machine Intelligence, 2nd Ed., IEEE Press, Piscataway, NJ, 2000.
[31] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, Cambridge, MA, 1998.
[32] S. Nolfi, D. Floreano, Co-evolving predator and prey robots: Do 'arms races' arise in artificial evolution? Artificial Life 4 (4) (1998) 311–335.
[33] D. Cliff, 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, vol. 929, Springer, 1995, pp. 200–218.

A.L. Nelson was born in Laramie, Wyoming, in 1967. He received his B.S. degree from the Evergreen State College in Olympia, Washington in 1990. He received his M.S. degree in Electrical Engineering from North Carolina State University in 2000 and his Ph.D. in Electrical Engineering at the Center for Robotics and Intelligent Machines at North Carolina State University in 2003. Currently, his is a Visiting Scholar in the Department of Computer Science and Engineering at the University of South Florida.

His main research interests are in the fields of autonomous distributed robotic control with a particular focus on cooperative multi-robot colonies and automatic controller synthesis. Currently, his research is focused on the evolution of artificial neural networks and fuzzy rule sets for the control teams of autonomous mobile robots. His long-term goal is the development of fully autonomous environmentally situated intelligent robots.

E. Grant is the Director of the Center for Robotics and Intelligent Machines (CRIM) at North Carolina State University and Associate Professor of Electrical and Computer Engineering. He earned a Bachelor of Science (Hons) in Mechanical Engineering from Dundee College of Technology (now University of Abertay Dundee) in 1969, Master of Engineering in Mechanical Engineering (Fluid Power Control) from the University of Sheffield in 1972, and a Ph.D. in Computer Science from the University of Strathclyde in 2000. Dr Grant is a Chartered Engineer (C.Eng.), a Fellow of the Institution of Mechanical Engineers (F.I.Mech.E.), and a Senior Member of the Institute of Electrical and Electronics Engineers (S.M.I.E.E.E.). He was the Founding Chairman of the United Kingdom and Republic of Ireland Chapter of the I.E.E.E. Robotics and Automation Society. He is an Associate Editor of the International Journal of Robotics and Autonomous Systems. His research interests are in the areas of knowledge-based control of robotic systems and dynamic systems, evolutionary robotic systems, machine learning with applications to medical image diagnosis, e-textiles for conformal surfaces for robots, and search and rescue robots.

T.C. Henderson received his B.S. degree in Math from Louisiana State University in 1973 and Ph.D. in Computer Science from the University of Texas at Austin in 1979. He is currently the Professor of Computer Science in the School of Computing at the University of Utah. He has been at Utah since 1982, and was an Visiting Professor at DLR in Germany in 1980, and at INRIA in France in 1980 and 1987, and at the University of Karlsruhe in Germany in 2003. Professor Henderson is the author of Discrete Relaxation Techniques (University of Oxford Press), and Editor of Traditional and Non-Traditional Robotic Sensors (Springer-Verlag). His research interests include autonomous agents, robotics and computer vision, and his ultimate goal is to help realize functional androids. He has produced over 200 scholarly publications, and has been principal investigator on over $8M in research funding. Prof. Henderson is a Fellow of the IEEE, and received the Governor's Medal for Science and Technology in 2000. He enjoys good dinners with friends, reading, playing basketball and hiking.