A. L. Nelson, E. Grant, G. Lee, "Developing Evolutionary Neural Controllers for Teams of Mobile Robots Playing a Complex Game," *IEEE International Conference on Information Reuse and Integration, (IRI 2003)*, Las Vegas NV, Oct 27-29, 2003, pp. 212 – 218.

Developing Evolutionary Neural Controllers for Teams of Mobile Robots Playing a Complex Game

Andrew L. Nelson, Edward Grant, and Gordon Lee

***Abstract*—This research develops methods of automating the production of behavioral robotics controllers. Population-based artificial evolution was employed to train neural network-based controllers to play a robotic version of the team game *Capture the Flag*. The robot agents used processed video data for sensing their environment. To accommodate the 35 to 150 sensor inputs required, large neural networks of arbitrary connectivity and structure were evolved. An intra-population competitive genetic algorithm was used and selection at each generation was based on whether the different controllers won or lost games over the course of a tournament. This paper focuses on the evolutionary neural controller architecture. Evolved controllers were tested in a series of competitive games and transferred to real robots for physical verification.**

Index Terms—Evolutionary robotics, Robot colonies, Neural networks, Evolutionary neural computing, Behavioral robotics

I. Introduction

THE fundamental goal of evolutionary robotics (ER) is to apply evolutionary computing methods to automate the production of complex behavioral robotic controllers. Many proof-of-concept experiments reported on in the literature used computer-based simulations only [1]-[3]. Examples of ER applied to real robots include the evolution of walking behaviors in hexapod and octopod robots [4][5], and the evolution of simple behavioral controllers for small mobile robots in closed environments [6][7]. These include the development of phototaxis behaviors [8][9] and of object avoidance and navigation in small robots using differential steering [7][10].

The work described in this paper attempts to move ER research beyond the nascent proof-of-concept stage. The experiments presented in this paper show that it is possible to evolve moderately complex mobile robot controllers using competitive tournament-selection methods.

The field of evolutionary robotics has been reviewed in recent publications [10]-[13]. Important issues raised in this literature include 1) the application of ER methods to more sophisticated problems; 2) methods of performance and fitness evaluation; 3) embodied evolution in real robots vs. evolution in simulation; and 4) the coupling of simulation to reality. In this work we will focus mainly on the first issue.

The evolution of robot controllers requires formulation of a fitness selection function. Most commonly, a task specific fitness selection function is formulated by hand and by trial and error. For complex behaviors, this can require in-depth knowledge of the dynamics of the behavior to be evolved. One method used to address the problem of evolution of more complex behaviors is incremental evolution [3][4][10][14]. Direct evaluation by humans has also been used in some ER work [10][15][16]. However, all of these methods limit the automation aspect that is central to ER.

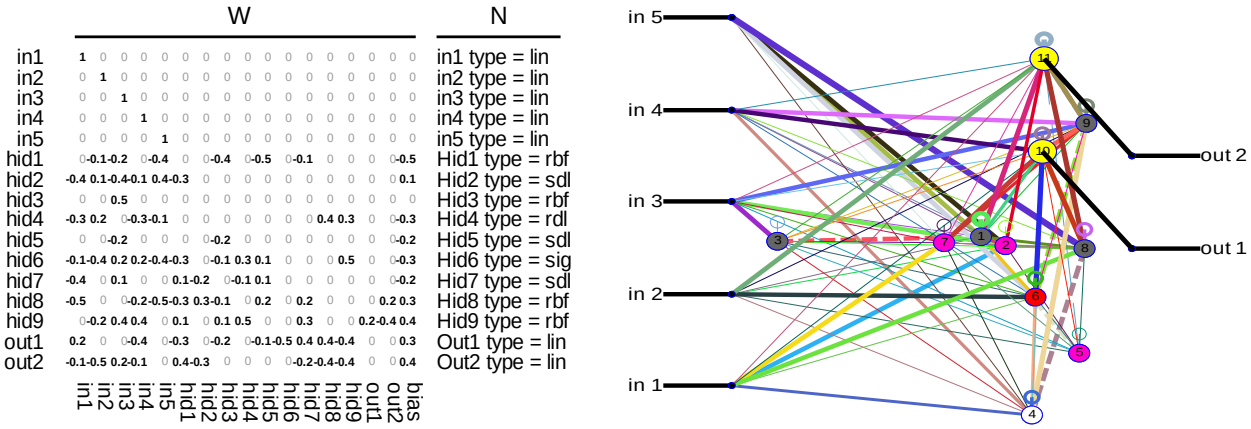
Many games requiring high levels of skill can be scored in a tournament using relatively simple and deterministic metrics (measures), e.g., Checkers-Playing neural networks [17] and Go-Playing neural networks [18]. In cases where at least one team or player of an evolving population achieves a win in a tournament, metric complexity can be reduced further to best number of games won in a tournament.

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 goal. The robot which first comes within a range of its opponent's goal wins the game for its team. The game is played in maze worlds of varying configurations.

An advanced evolutionary robotics research testbed was used in this research. The components of this testbed are: (1) an evolutionary artificial neural network application; (2) a colony of robots that use vision-based range finding sensor systems; and (3) a simulation and evolutionary training environment. We focus in detail on the neural network and genetic algorithm formulations. The physical robot systems are described in [19] and [20].

A. L. Nelson and E. Grant are with the Center for Robotics and Intelligent Machines, Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC 27695-7911, alnelson@ieee.org, egrant@ncsu.edu.

G. Lee is with the Department of Electrical and Computer Engineering, San Diego State University, 5500 Campanile Drive, San Diego, CA 92182, glee@kahuna.sdsu.edu.



(a) (b)

Fig. 1. An example neural network encoding. Panel (a) shows the weight and connectivity matrix W, and the neuron type vector N. Panel (b) shows the corresponding network graphical representation with inputs on the left and outputs on the right.

II. The Evolutionary Neural Network

Much of the ER work to date used very simple network topologies and restricted weight values [7][9][21][22]. Such restrictions limit the scalability of the methods studied. Other researchers have used more complex networks [5][10][23] and we pursue this path. We have developed a generalized class of network structures. These networks contain: (1) feed forward and feedback connections, (2) mixed types of neurons, and (3) variable integer time delays on neuron inputs. Neuron activation function types include sigmoid, linear, step-threshold, and Gaussian radial basis functions.

The connectivity and weighting relationships are contained in a single two-dimensional matrix **W**. Information specifying neuron types are stored in a vector structure **N**, with one formatted field per neuron. Fig. 1 shows an example network encoding. **W** and **N** are shown in panel (a) and the resulting network graphical representation is shown in panel (b). In the graphical representation, only nonzero weights are shown (as weighted lines). Neuron location is a function of connectivity. Note that the example network of Fig. 1 is much smaller than the typical network evolved in this work. The example network is included to illustrate network representation.

This network representation is designed to facilitate the evolution of populations of variable-size and arbitrarily connected networks. In particular, neurons can be added or removed without altering the connectivity relationships of other neurons in the network by inserting (or deleting) the appropriate row and column of **W**, and row of **N**.

Current and past network inputs and neuron functional levels (outputs) are stored in an ordered matrix, **I**. The maximum level of time delay is a scalar integer, δ. Neuron activation functions take the form:

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

where $n \in \{1..N\}$, $\mathbf{w}_n$ is the nth row of the weight matrix **W**, $\mathbf{i}(t, \tau(n))$ is the τth row ($1 \le \tau \le \delta$) 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 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 sum of the weight inputs (dot product),

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

For the radial basis activation functions, u is the Euclidian distance between **w** and **i** in n-space.

Network inputs are considered to be linear neurons while function outputs can be selected and read from the matrix **I** after a network updating cycle. The network input-output relationship is:

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

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 sub-set of ***i***, 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). **I**(t) is given in expanded form below.

$$\mathbf{I}(t+1) = \begin{bmatrix} [i_1, \ldots i_L, f_{L+1}(\mathbf{w}_{L+1}, \mathbf{i}'(t+1, \tau(L+1)), \ldots f_N(\mathbf{w}_N, \mathbf{i}'(t+1, \tau(N))] \\ \mathbf{i}(t,1) \\ \mathbf{i}(t,2) \\ \vdots \\ \mathbf{i}(t, \delta-1) \end{bmatrix} \tag{5}$$

The functional *Network* of Equation (3) calculates the outputs of each neuron specified in **N** in order, placing resulting values in successive elements of **I**.

A fully evolved controller network is shown in Fig. 2. The network uses 150 inputs to accommodate processed video sensor information and produces two drive wheel command outputs that control the robot's differential-steering wheel motors. The details of the neural evolution process are discussed in section IV.

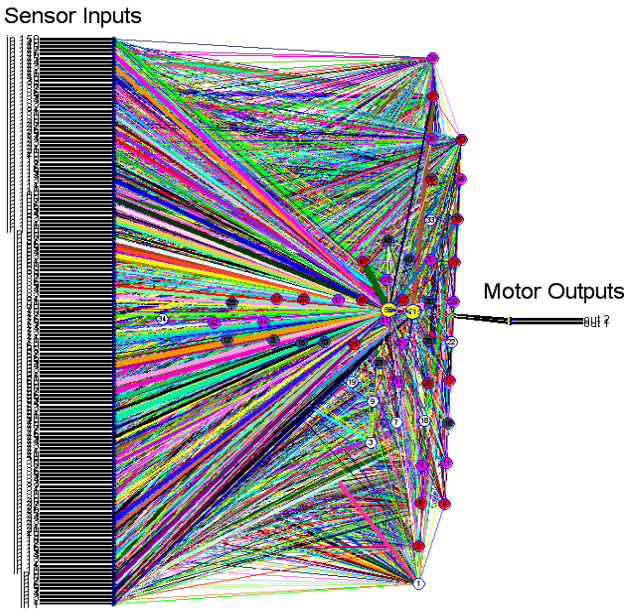


Fig. 2. A fully evolved controller neural network.

III. The Real Robots and the Simulation Environment

Evolution of the neural controllers is performed in a simulated environment, one that is coupled to a real robot environment for testing and verification. The physical robots and the simulation environment have been described in [19][20] and [24], respectively and will be only briefly discussed in this section.

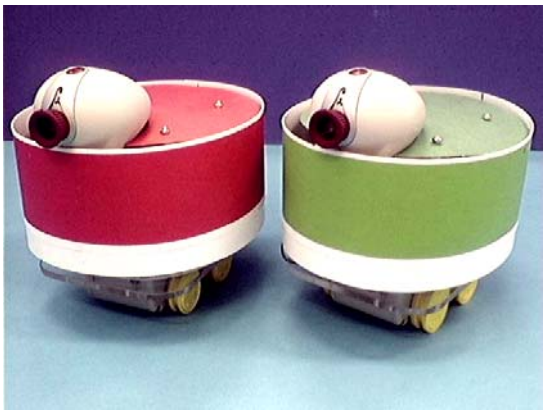
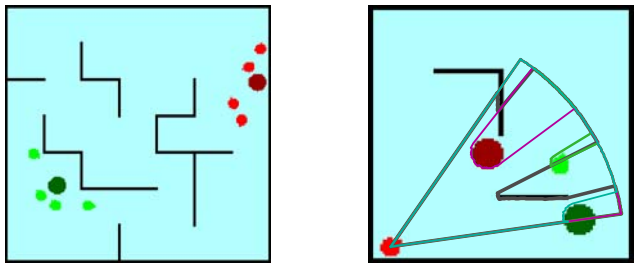


Fig. 3. EvBot robots fitted with colored shells.

The physical robots used in this work are the EvBots [19][20]. These robots use vision base range-finding sensors for detection of their environment [24]. The robots are fully autonomous and are capable of performing all vision processing and control computation on board. Evolved neural networks are transferred to the real robots for testing. Fig. 3 shows a photograph of two EvBots. Each robot is fitted with a colored shell. The shells are used in the *Capture the Flag* game behavior and serve to differentiate robots on different teams.

In the simulation environment, robot agents, sensors, robot-environment and robot-robot interactions are modeled. Simulated sensors extract range data from the environment and format them into the same format reported by real video range sensors used on the physical robots [25].

Fig. 4 shows two screen captures form the simulation environment.



(a) (b)

Fig. 4. Graphical representation of the simulation environment. In (a) the robots are shown clustered around their respective goal objects. In (b) the simulated range sensor data received by the robot in the bottom left-hand corner is displayed as the pie-piece shaped graphic superimposed on the environment.

IV. Fitness Selection and the Evolutionary Algorithm

In this section the fitness selection metric and evolutionary algorithm used to evolve populations of neural network-based robot controllers is discussed.

The process of controller evolution consists of a repeating cycle of several steps. This cycle is roughly analogous to a generation in a natural evolutionary process. During the cycle, individual neural controllers in a larger population of neural controllers ($p \in \mathbf{P}$) perform a task or engage in a performance period. For the game-playing behavior, this performance period consists of a tournament of competitive games involving all the members of the population. Following this, each neural controller's performance is evaluated based on a fitness selection metric $F(p)$. In the final step of the cycle, a genetic algorithm (GA) is applied. The GA uses information from the fitness selection function to select and propagate the fittest individuals in the current population to the next generation population. During propagation, controller networks are altered slightly using

stochastic genetic operators
(mutation) to produce offspring that make up the next generation of controllers. This cycle is repeated for many generations to produce populations of functional robot controllers. This process can be considered as a form of machine learning or neural network training.

A. Fitness selection function specification

Fitness for individual controllers is based on their performance in competition in tournaments of games. During each generation, a single tournament of games was played. A bimodal training fitness selection function was used. The selection function has 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 (winning or losing games). Additionally, this selection metric was applied in a relative competitive form in which controllers in an evolving population compete against one another to complete their task - to win the competitive game.

Fitness $F(p)$ of an individual p in an evolving population **P** ($p \in \mathbf{P}$) takes the general form:

$$F(p) = F_{\text{mode\_1}}(p) \oplus F_{\text{mode\_2}}(p) \tag{6}$$

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 $F_{\text{mode\_1}}$. $F_{\text{mode\_1}}$ is formulated to return negative values and returns 0 when maximized or if $F_{\text{mode\_2}}$ is active. $F_{\text{mode\_2}}$ in contrast returns positive values based on number of game-wins, if any.

The first mode of the fitness function selects for minimal competence to successfully complete the task (however poorly) in a detectable fraction of the trials, and in a finite amount of time. In essence, the mode selects for the ability to travel a distance D through the competition environment. The general form of mode 1 is as follows:

$$F_{\text{mode\_1}} = F_{dist} - s - m \tag{7}$$

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:

$$F_{dist} = \begin{Bmatrix} -\alpha * (D-d) & \text{if } d < D \\ 0 & \text{otherwize} \end{Bmatrix} \tag{8}$$

D is defined as half the length of the training environment's greatest dimension and α is a constant of proportionality. In Equation (7), 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.

The second mode of the fitness function $F_{\text{mode\_2}}$ is classified as aggregate because it produces fitness based only on success or failure of the controllers to complete the task at hand (competitive team game playing). The formulation of the success/failure mode ($F_{\text{mode\_2}}$) of the fitness function is determined by the competitive nature of the training algorithm and the behavioral task. Here, 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). Note that the opponent was selected at random from the population at the beginning of each tournament. The possible outcomes of these games incurred different levels of fitness and are summarized in Tab. 1 below.

Table 1. Fitness points awarded by the aggregate success/failure mode $F_{\text{mode\_2}}$, for pairs of reciprocal games during a generational tournament.

Game Pair Outcomes	Fitness Points Awarded
win-win	**3**
win-draw	**1**
win-lose	**.5**
no win	0 ($F_{\text{mode\_1}}$ dominates)

Note that in cases where no win occurs $F_{\text{mode\_1}}$ is used to determine a negative fitness value.

B. The GA and neural network mutation

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 Equation (6). The population **P** is always ordered from fittest to least fit before propagation to the next generation.

A very simple selection and replacement genetic algorithm (GA) was used. Offspring are generated using mutation only. During the propagation phase of the GA, the fittest 50% of the population produce offspring that replace 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. Ancestor-elitism was used to include the best individual from two generations passed (slightly reduces the percent of the wins per generation required for pure win/lose based selection).

Network weights, neurons and connectivity are mutated directly. The genome **C** is specified by the two dimensional matrix of real numbers

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

where **W** is the weight/connection matrix, and **N**' is a column vector extracted from the formatted structure **N** (see Fig. 1). Mutation of a network selected for inclusion in the next generation population is formalized by the compound relation

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

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.

V. Results

In this section, we present results and tests of a population of neural network-based robot controllers evolved to play the game *Capture the Flag*. Training data collected over the course of evolution of a population of controllers are presented. Examples of game-playing behaviors of fully evolved controllers are presented. In addition, evolved controller performance is measured by placing the best member of the evolved population in competition against a knowledge-base controller of well-defined abilities.

A. Population evolution

Fig. 5 shows evolutionary training data collected over the course of the evolution of a population of controllers. The population was kept at a constant size of 20 individuals. The evolution process was performed in a single maze environment (of the configuration shown in Fig. 6). The top panel of Fig. 5 shows fitness selection values generated by equation (6) for the population and plotted over the course of 500 training generations. The best, mean, and poorest controller fitness values for the evolving population are plotted. The data indicate that the highest relative selection value (3, due to winning two games in a generation) is not achieved by any controller before the 50th generation. Early in training, no controller in the evolving population is able to win any game. During this early phase of evolution the first mode (F_{mode_1}, equation (6)) of the bimodal fitness selection function dominates selection. The lower panel of Fig. 5 shows the number of total wins achieve by the population as a whole at each generation. This is a purely passive metric and has no effect on selection whatsoever. It is included here to give a measure of overall population fitness. After the 300th generation the number of wins per generation is sufficient so that the second purely competitive mode of the fitness function dominates the selection process. The dashed horizontal line on the lower panel of Fig. 5 indicates the wins-per-generation threshold at which selection become completely based on win/lose information.

B. Evolved controller behavior

Fig. 6 shows the results of controllers from the evolve population competing against one another in a simulated environment. Fig. 7 shows the evolved controllers competing in the physical environment using teams of real robots. In both cases, all robots used evolved neural network controllers. The paths taken by the robots in both figures are indicated by light (green) and dark (red) curves. In simulation, and in the physical world, robots display qualitatively similar general behaviors.

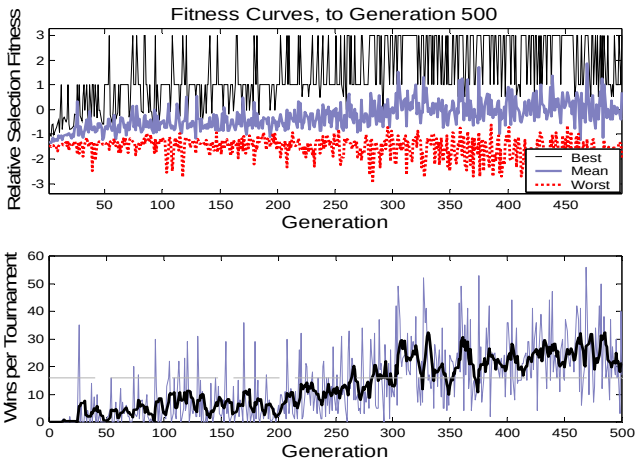


Fig. 5. Training and evolution data collected over the course of evolution of a population of controllers. The top panel shows fitness selection function values. The lower panel shows a purely passive fitness metric: the number of total games won by the population during each generation.

The controller simulation and physical environments are coupled so that controllers can be transferred directly from simulation to physical robots without and alterations. Hence, evolved controllers can be tested in the real world, in simulation, or in a combination of both.

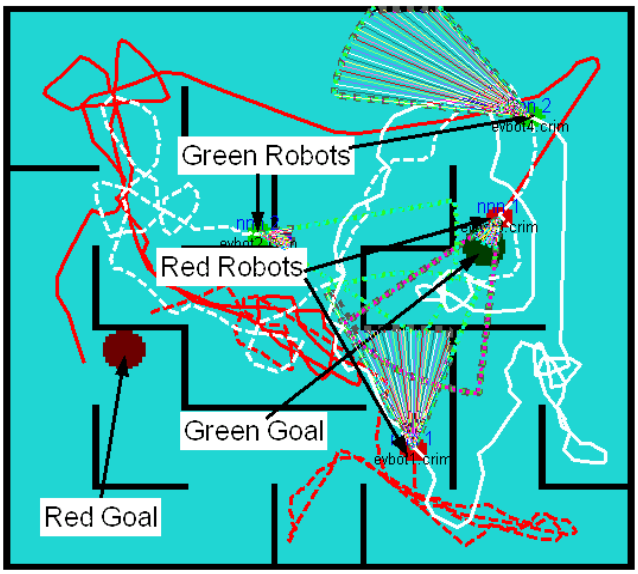


Fig. 6. Simulated robot agents competing in a simulated maze world. The light and dark curves indicate the paths taken by the

robots during the course of the game. The game was won be the red (dark) team.

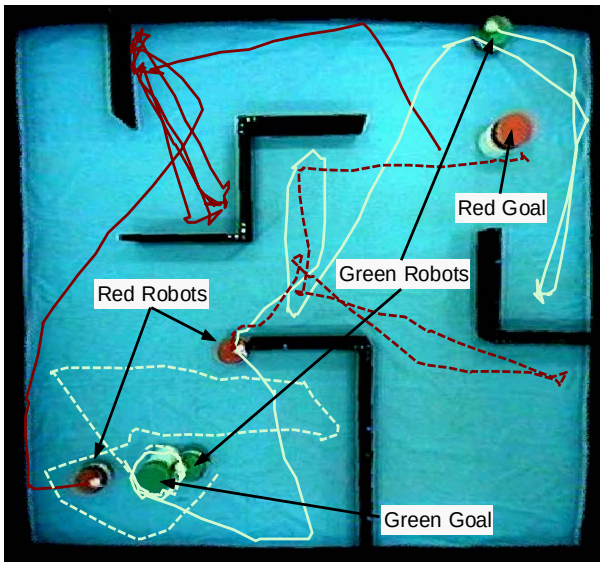


Fig. 7. Real robots competing in a physical environment. The light and dark curves indicate the paths taken by the robots during the course of the game. The game was won be the red (dark) team.

C. Evaluation of evolve controllers performance

The game sequences of Figs. 6 and 7 demonstrate that controllers have evolved competent game-playing behaviors. However, because a relative competitive fitness selection metric was used to drive the evolutionary process, absolute fitness is not known. To address this, fully evolved controllers were compared to knowledge-based controllers of known abilities. An extensive tournament of 240 games was conducted in an environment similar to the one shown in Fig. 6. Each game during the tournament was initialized with a new randomly generated set of starting position for robots and goals. Fig. 8 shows the results of this tournament. The best-evolved neural controller won 108 games, the knowledge-based controller won 103 games, and 29 games were played to a draw.

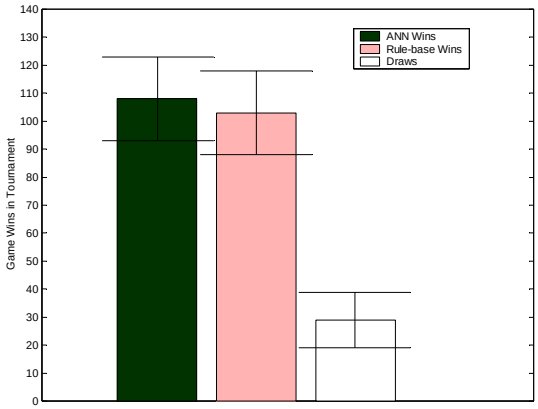


Fig. 8. Bar graphs displaying evolved controller and knowledge-based controller competition data collected during a tournament of 240 randomly initialized games. Data are shown with 95% confidence intervals.

In marked contrast to the knowledge-based controllers, the neural network-based controllers displayed complex trajectories that were extremely difficult to predict exactly. Although it may be possible to qualitatively analyze the evolved controller behaviors to a degree, such analysis is not at all necessary to the evolutionary process. Human knowledge was required to formulate the minimal competence mode of the fitness function ($F_{\text{mode\_1}}$, equation (6)), but this only selected for minimal navigation abilities. The high-level game-playing behaviors were evolved based mainly on competitive game playing during evolution.

The main focus of this paper has been on the evolutionary neural network architecture and evolutionary process. Further studies related to the characterization of evolved controllers using this platform are reported on in [24][25].

VI. Conclusion

In this paper, an evolutionary robotics neural network controller architecture and training environment were described. The networks evolved in this work are very large in comparison to other ER work. Networks accommodate between 35 and 150 processed video sensor inputs. Robot controllers were evolved to play the game *Capture the Flag* and were tested in a set of simulated games and in a physical environment with real robots. Evolved neural network based controllers were able to play competitively against a knowledge-based controller of well-defined abilities..

References

[1] F. Gomez, R. Miikkulainen, Incremental Evolution of Complex General Behavior, Adaptive Behavior, Vol. 5, pp. 317-342, 1997.

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

[3] J. Kodjabachian and 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) (September 1998).

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

[5] N. Jakobi, Running Across the Reality Gap: Octopod Locomotion Evolved in a Minimal Simulation, Proceedings of the First European Workshop on Evolutionary Robotics: EvoRobot'98, 1998.

[6] D. Floreano and S. Nolfi. Adaptive behavior in competing co-evolving species. Mantra technical report, LAMI, Swiss Federal Institute of Technology, Lausanne, 1997. Also submitted to ECAL97.

[7] D. Floreano and F. Mondada , Evolution of homing navigation in a real mobile robot. IEEE Transactions on Systems, Man, Cybernetics Part B: Cybernetics Vol. 26 No. 3, (1996), pp. 396-407.

[8] 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, and P. Chacon, editors, Advances in Artificial Life: Proc. 3rd European Conference on Artificial Life, pages 704--720. Springer-Verlag, Lecture Notes in Artificial Intelligence 929, 1995.

[9] 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, Volume 39, April 2002.

[10] I. Harvey, P. Husbands, D. Cliff, A. Thompson and N. Jakobi, Evolutionary robotics: the Sussex approach, Robotics and Autonomous Systems, (20)2-4 (1997) pp. 205-224.

[11] M. Mataria and D. Cliff, Challenges in evolving controllers for physical robots, Robotics and Autonomous Systems , Volume 19, Issue 1, Pages 67-83, November 1996.

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

[13] Floreano, D. and Urzelai, J. (2000) Evolutionary Robotics: The Next Generation. In T. Gomi (ed.), Evolutionary Robotics III, Ontario (Canada): AAI Books, 231-266.

[14] W. Lee, Evolving Complex Robot Behaviors. Information Sciences 121(1-2): 1-25 (1999).

[15] H. Lund, O. Miglino, L. Pagliarini, A. Billard, A. Ijspeert, Evolutionary Robotics-A Children's Game, Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence, pp. 154-158, 1998.

[16] F. Kaplan, P. Oudeyer, E. Kubinyi and A. Miklosi, Robotic clicker training, Robotics and Autonomous Systems, Volume 38, Issues 3-4, Pages 197-206, 2002.

[17] 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.

[18] 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.

[19] 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.

[20] J. Galeotti, S. Rhody, A. Nelson, E. Grant, and Gordon Lee, EvBots – The Design and Construction Of A Mobile Robot Colony for Conducting Evolutionary Robotic Experiments, Proceedings of the ISCA 15th International Conference: Computer Applications in Industry and Engineering (CAINE-2002), pp. 86-91, San Diego Ca, Nov. 7-9, 2002.

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

[22] S. Nolfi, Evolving non-trivial behaviors on real robots, Robotics and Autonomous Systems, (22) 3-4 (1997) pp. 187-198.

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

[24] 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.

[25] A.L. Nelson, "Competitive Relative Performance and Fitness Selection for Evolutionary Robotics", Doctoral Dissertation, North Carolina State University, 2003.

Corresponding Author:
Andrew Nelson
E-mail: alnelson@ieee.org
Web: http://www.nelsonrobotics.org