Competitive relative performance evaluation of neural controllers for competitive game playing with teams of real mobile robots

A. L. Nelson and E. Grant
Center for Robotics and Intelligent Machines
Department of Electrical and Computer
Engineering
North Carolina State University
Raleigh, NC 27695-7911

T. C. Henderson
Department of Computer Science
School of Computing
3190 Merrill Engineering Building
University of Utah
Salt Lake City, Utah 84112

ABSTRACT

In this research, we describe the evolutionary training of artificial neural network controllers for competitive team game playing behaviors by teams of real mobile robots (The EvBots). During training (evolution), performance of controllers was evaluated based on the results of competitive tournaments of games played between robots (controllers) in an evolving population. Competitive tournament fitness evaluation does not require a human designer to define specific intermediate behaviors for a complex robot task. Intermediate behavior selection and evaluation becomes an implicit part of winning or losing games in a tournament. The acquisition of behavior in this 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 controllers were evolved in a simulated environment using evolutionary training algorithms and were then transferred to real robots in a physical environment for validation. Evolutionary robotics makes use of several distinct types or levels of performance evaluation. The work presented here focuses on the competitive relative tournament ranking metric used to drive the evolutionary process. After a population has been evolved, a second metric is needed to evaluate the quality of acquired game-playing skills. We use a post training evaluation method that compares the evolved controllers to hand coded knowledge-based controllers designed to perform the same task. In particular, a very poor controller, and high quality controller give us two points on a continuum that can be used to rank the evolved controller quality.

Keywords: *evolutionary robotics, performance metrics, mobile robot colonies, evolutionary neural computing, behavioral robotics*

1. Introduction

1.1 Evolutionary robotics

Evolutionary robotics (ER) is a relatively recent addition to the field of autonomous robot control research. ER focuses on the automatic design of model-free robot controllers using evolutionary computing methods. Over the course of last decade, proof-of-concept research in the field of ER has been conducted. Much of this work was done using computer-based simulations only [1][2][6]. Examples of ER research conducted with real robots include the evolution of walking behaviors in hexapod and octopod robots [7][8], and the evolution of simple behavioral controllers for small mobile robots[9][10]. The later include the development of phototaxis behaviors [11][12] and of simple object avoidance [10] and navigation [13]. For recent reviews of the field of ER see [14][15][16][13].

1.2 Intelligence performance metrics in ER

An ER application may make use of one or more types of fitness or intelligence metrics. These include 1) measurement of behavior quality for selection during training, 2) measurement of quality of transference from simulated training environments to the real world, and 3) post training evaluation of acquired behaviors. In this paper, we will focus mainly on development and formulation of the first type of metric. In addition, a post training metric will be applied to evaluate the quality of the best member of an evolved robot controller population.

In evolutionary robotics, a training performance fitness function is applied to provide selective pressure to an evolving population of robot controllers. The goal is to develop intelligent behavior with regard to a particular task. The nature and implementation of a machine learning application affects the way in which its intelligence can be measured. Behavioral robotics applications impose tight physical constraints on the expression and evaluation of learned behaviors. In particular, a subtle issue arises regarding the point of view of the intelligent robotic system and the point of view of the external observing human who is trying to evaluate that system's intelligence. These different points of view are known as the proximal (local) and distal (external) viewpoints respectively [3].

In most cases, fitness functions used in ER are formulated by designers from the distal point of view. Designers naturally develop fitness functions based on their own understanding of the desired behavior and system

dynamics. In doing so, they implicitly incorporate information from their own complex distal world model into the fitness evaluation of the evolving agent(s). Since the evolving agent has no such model, the evolved behaviors tend to be very brittle. In the following section we will discuss some of the properties of such distal absolute fitness functions. As an alternative to absolute fitness functions we will present the formulation and experimental testing of an aggregate relative fitness evaluation method for ER.

1.3 Absolute vs. aggregate relative fitness functions in ER

Fitness in ER systems is often measured as an absolute scalar function to be maximized or minimized during training (see [4][5], for examples of this type of fitness function).

Absolute fitness functions used in evolving behavioral robotics applications are problematic for the following reasons: 1) Often, a forced learning plateau arises when the fitness metric is maximized or minimized, 2) Human biases are incorporation into the metric, 3) Each new robotic application requires a new and often difficult-to-formulate metric, and 4) For many complex behaviors, the knowledge required to specify an adequate absolute training fitness function is equivalent to that that would be required to design a rule/knowledge based controller by hand.

In this work we study a relative aggregate fitness selection metric for the evolution of behavioral robotics controllers. In particular, we focus on behaviors that can be formulated into competitive games played between two or more mobile robots. The metric produces a relative ranking in an evolving population of controllers, but does not give an absolute measure of fitness with respect to an external scale. Evaluation of evolving controllers based on their relative abilities to perform a task has the affect of aggregating evaluation of intermediate behaviors into one simple performance measure.

Tournament ranking evaluation partially eliminates the need to generate 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 function definition. Agents in an evolving population that receive higher relative rankings in a tournament of games will be propagated preferentially over agents receiving lower rankings. This reduces the amount of human bias that is incorporated into the performance metric. It also allows metrics to be specified in cases where humans lack adequate information to specify effective absolute fitness factors. This is important

to the long-term scalability of ER methods to uncharacterized domains.

Implementing an aggregate competitive fitness function in the domain of evolving robot controllers is qualitatively different than similar implementations in pure computer science domains. In [17] tournament selection was applied to evolve neural networks to play computer Checkers with impressive results. In that work, the game board configurations were deterministically coded and fed directly into the neural networks. In ER, training environments must maintain an explicit I/O coupling analogous the robot's physical functional environment. This coupling must enforce a realistic proximal view onto the evolving agents. Modeled sensors must report only information that could be produced by the real sensors. Modeled motor actuators must in turn produce an alteration in the robot's frame of reference that appropriately alters the modeled sensor values (i.e. after the robot moves, it sees something new). This forms a controller-actuator-sensor loop with relational dynamics must be the analogous to those experienced by the real robots. The temporal fidelity of this controller-actuator-sensor loop must be maintained.

Another factor that complicates implementation of reinforcement learning of behavior in ER systems is that, many potential controller configurations may not lead to detectable expression of a desired behavior in a finite amount of time. In such cases, the search space must be restricted so that controllers will display detectable differences in performance even when that performance is measured at the aggregate level of win or lose. One way to do this is to formulate the competitive evaluation environment so that most controllers, even very poor ones, will eventually win at least a fraction of the games if their opponents are as poor or poorer than themselves.

2. The evolutionary robotics research testbed

Before presenting a formulation of the relative aggregate fitness function used in this work, we will provide a brief summery of the ER research testbed used here. This will provide a context for the training fitness function formulation. The ER research testbed consists of a physical colony of autonomous mobile robots, and an evolutionary neural network training environment. The real robots use a vision-based range finding sensor emulation system to locate them selves in a physical maze environment. The evolutionary neural network training application uses simulation of the robots and their environment to evolve neural controllers to drive the robots. Controllers are evolved in simulation and then transferred to real robots for testing and verification.

2.1 The EvBot platform and environment

The physical verification and testing of evolved controllers developed in this research was conducted using a colony of small mobile robots named the EvBots (EVolutionary roBOTs)[18]. These robots are computationally powerful, fully autonomous and capable of performing all control, computing and data management on board. The robots move and steer using differential speed control of parallel drive wheels.

A physical maze environment was constructed for the mobile robot colony. Robots and objects in the environment were fitted with colored skirts to aid in vision based sensing of the environment. A fully assembled EvBot and the physical maze environment are shown in panels (a) and (b) of Figure 1 respectively.
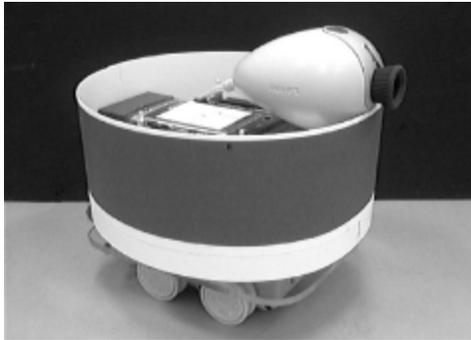
(a)
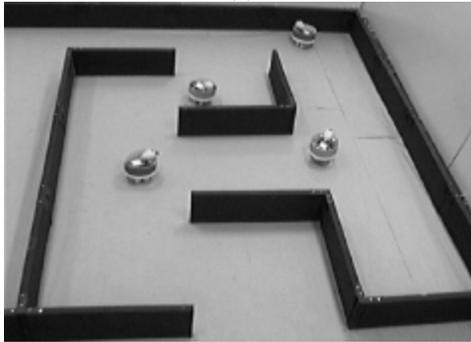
(b)

Figure 1. Photographs of (a) a fully assembled EvBot and (b) the physical robot maze environment containing several robots.

2.2 Video range-finding emulation sensors

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

The vision-based range-finding sensor systems on the robots used fixed geometric properties of the physical maze environment to calculate the ranges and angles of materials. Using color and position, the vision system could detect walls, robots, and goals. The goals are stationary cylinders and were used in the robotic 'Capture the Flag' game.

At each sensor update interval, and for each object type, range and angle values were calculated over the horizontal field of view of the robot's camera. A vector of range values was produced for each object type. Angular data was implicitly encoded in the order of the range values reported in each object range data vector. Object type information was not explicitly given to the robot neural controllers. Controllers were only given these resulting numerical data vectors. All associations relating distances, angles, and object types must be learned by the neural networks.

2.3 The evolutionary neural network architecture

In this research, a generalized evolvable neural network architecture capable of implementing a very broad class of network structures was used. The networks may contain arbitrary feed forward and feedback connections between any of the neurons in the network. Networks contain neurons with heterogeneous activation functions including sigmoidal, linear, step-threshold, and Gaussian radial basis functions. Neurons include a variable time-delay associated their inputs. This give networks the potential to evolve temporal processing abilities.
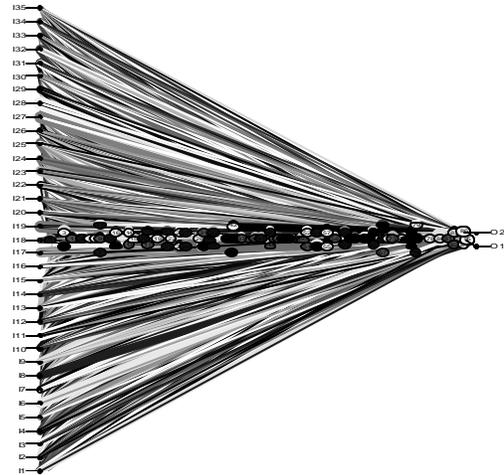
Figure 2. An example robot neural network controller from an evolved population of heterogeneous neural networks. The inputs to the network (left) are supplied by the robot's video range emulation sensors. The outputs of the network (right) are interpreted as wheel motor speed commands.

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 vector structure **N** with one formatted

field per neuron. **W** and **N** form the basis of the genetic encoding for each network.

Figure 2 shows an example of a graphical representation of an evolved neural network.

2.5 Network mutation

The elements in the weight matrix and neuron information structures are acted upon directly by the genetic algorithm. Formally, the genome for a network **C** can be specified by the two dimensional matrix of real numbers

$$\mathbf{C} = [\mathbf{W} : \mathbf{N}'] \tag{1}$$

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

$$\mathbf{C}' = M_s(M_c(M_w(\mathbf{C}))) \tag{2}$$

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.

3. The fitness function and genetic algorithm

3.1 Fitness function formulation

In this section, we will define the fitness function used in this research. It 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 robot games.

The training fitness function is comprised of two over-all parts: 1) select for controllers that win more games, 2) identify and select against pathological controller morphologies.

Only pathological cases that were known to lead to catastrophic stagnation of the evolutionary process were explicitly selected against. Two pathological controller behaviors were actively selected against. The first behavior was the production of constant continuous 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. These cases will be represented by Boolean functions *B1* and *B2* denoting the presence (1) or lack (0) of expression of each of the pathological behaviors, respectively.

A population **P** of evolving robot controllers consists of a fixed number *P* of neural networks. At the beginning of a tournament, a set of game starting positions for robot teams and goals is quasi-randomly generated and used for every game in that tournament (generation). In every training generation, a full tournament of games is played: Each controller in the population **P** plays against every other controller in **P**. After a tournament of games, each controller is given a score that depends on the number and quality of wins it achieved. 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 controller in the second team of robots. In the second game, the controllers are switched. This eliminates any advantage a controller may have incurred due to the random initial conditions used in the games of that tournament.

A generalized form of the fitness function for an individual controller after a tournament has been played can be written as

$$F(p) = w + d + n \tag{3}$$

Where *w*, *d* and *n* are 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 relative fitness of the *p*th controller from 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. Let *G1*, *G2*, and *G3* denote numbers of games won during a tournament of each of the three classes respectively. Then the number of points awarded to the *p*th controller for games won in a tournament is given by:

$$w = a * G1 + b * G2 + c * G3 \tag{4}$$

Where a, b, and c are scalar weighting factors. The values of a, b, and c are generally set so that $a>b>c>0$. This reflects the evaluation that 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 given in the case that both of the games g and g' are played to a draw. If the robot agents of a particular controller are closer to their opponent's goal in both games, that controller is awarded points. The d sub-function of equation (3) becomes

$$d = d*D1 \qquad (5)$$

where $D1$ denotes the number game pairs played to a better draw and d is a scalar weighting factor. d is set to be much less than a, b, or c so that results related to numbers of wins dominate the tournament selection process.

Similarly, The n sub-function of 3 selecting against pathological behaviors can be expanded as

$$n = a*B1 + \beta*B2 \qquad (6)$$

Where $B1$ and $B2$ are 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 permanently stuck, respectively). a and b are scalar weighting factors and are generally set to be large negative values relative to a, b, and c so there is a heavy selective pressure against these behaviors even if they result in wins.

3.2. The evolutionary training algorithm

Populations of fixed size P were evolved using an evaluation, mutation, and replacement scheme. After each tournament of games, controller population members p were scored relative to each other using the performance metric $F(p)$ defined in equation (3). The population $\mathbf{P}$ was then reordered from fittest to least fit before propagation. The next generation population $\mathbf{P}\_next$ was then constructed from the union of the following three sets derived from the current (parent) population:

$$\mathbf{P}\_next = \{p_1 \ldots p_m\} \bigcup \{p'_1 \ldots p'_m\} \bigcup \{p_{m+1} \ldots p_{P-2m}\} \quad (7)$$

Where $p_m \in \mathbf{P}$ is the mth individual of the ordered current (parent) population $\mathbf{P}$, p'_m is a mutated version of p_m, and P is the fixed population size. Equation (7) produces a next generation composed of the following sets: 1) m of the fittest controllers are transferred un-changed to the next generation, 2) m of the fittest members of the controller population are mutated using equation (2) and added to the next generation, and 3) The remainder of the next

generation population is made up of the remaining fittest remaining members of the current population. Although this algorithm is technically a form of greedy mutation-only $(\mu + \lambda)$-ES with incomplete replacement [19], 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 and Discussion

4.1 The game

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 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 of 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.

Evolved controllers were able to play and win games both in simulation and when transferred to real robots in the physical world. The best evolved controllers acquired several distinct and testable abilities. These included avoidance of ones own goal, wall avoidance, goaltending, blocking and chasing robots from the other team, and homing in on an opponent's goal. Evolved controllers generally acquired two or three behaviors and exploited those rather than developing many behaviors for individual situations.

4.2. Experimental setup

We will focus on an evolved controller that displays two identifiable sub-behaviors: wall avoidance and selective avoidance of the robot'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. The parameters relating to the performance metric $F(p)$ of equation (3) used in this training evolution are given in Table 1 below.

Table 1. Fitness function parameter settings used to evolve the controller studied in these experiments.

Parameter	Game case (g, g')	Points awarded
a	win-win	20
b	win-draw	15
c	win-lose	10

d	best draw	2
a	backward	-10
b	stuck	-2

4.3 Performance of evolved controllers in the real world

Here we will present experiments aimed at measuring the quality, and indirectly the intelligence, of the evolved robot controller. These post-evolution evaluation experiments were done in the real world using real robots.

Two knowledge-base 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 on the opponent's goal. The second controller was designed to be a poor player and produced random wheel speed commands at each time step. In both cases, sensor input and motor output ranges were restricted to those allowed in the evolved neural network controller.

The evolved neural controller competed in a series of 20 real games against both the good rule-based 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. All games were recorded by collecting sequences of video images from a camera mounted directly above the maze.

Ten games were played between the evolved neural network and the good rule-base controller. Game initial positions may give one of the teams an advantage. For this reason, the set of games contained two games for each starting configuration used. In the first, each team was in a particular initial position, and in the second, the two team's starting positions were swapped. For these 10 games, five initial game positions were generated based on the random seed states 11 to 15 of the MATLAB random number generator. The games were conducted completely in the real robot maze environment using real robots. No games were allowed to proceed for longer than 200 controller update cycles (time steps). In addition, games were terminated if all robots became permanently stuck.

A similar set of ten real games was also 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 2 and 3 give the results of the games involving the good knowledge based and the random controllers respectively. The evolved neural controller, the good

knowledge based controller and the random controller are denoted as "neural", "rule" and "random" respectively.

Table 2. Results of the set of ten games played between the evolved neural network controller and the hand coded rule-based (good) controller.

Game	Random Init. state	Team1	Team2	Winner
1	1	neural	rule	neural
2	1	rule	neural	rule
3	2	neural	rule	neural
4	2	rule	neural	rule
5	3	neural	rule	rule
6	3	rule	neural	rule
7	4	neural	rule	rule
8	4	rule	neural	neural
9	5	neural	rule	rule
10	5	rule	neural	rule

Table 3. Results of the set of ten games played between the evolved neural network controller and the hand coded random (poor) controller.

Game	Random Init. state	Team1	Team2	Winner
11	1	neural	random	neural
12	1	random	neural	neural
13	2	neural	random	neural
14	2	random	neural	(none)
15	3	neural	random	neural
16	3	random	neural	neural
17	4	neural	random	neural
18	4	random	neural	neural
19	5	neural	random	(none)
20	5	random	neural	neural

Summarizing these results, the neural network controllers won 3 out of 10 games against the good rule based controller, or 30% and the good rule base won 7 out of 10 or 70% of the 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 two of the games were not completed because all the robots became stuck or the game proceeded for more than 200 moves.

Figure 3 shows two example game results from the above tables. These are games 2 and 12 respectively. 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 (Figure 3 (a)), neural network controllers (green, lighter dotted lines) compete against good knowledge-based controllers (red, dark dotted lines). In the second game (Figure 3 (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.
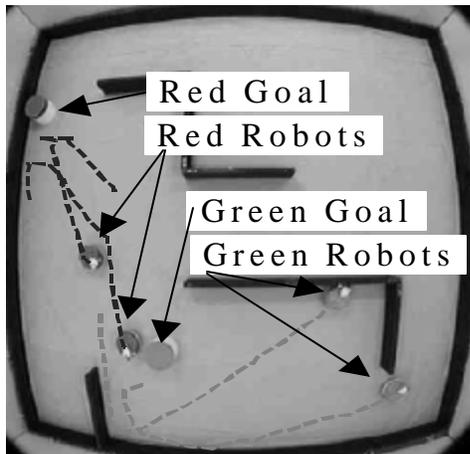
(a)
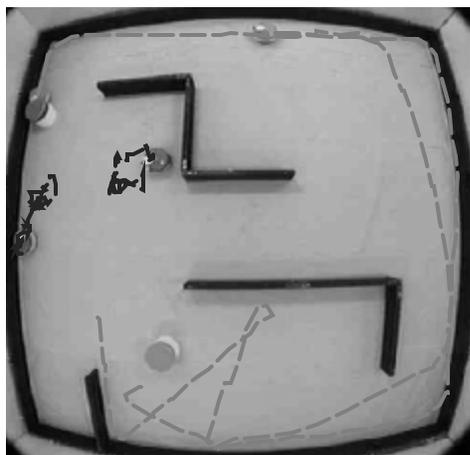
(b)

Figure 3. Examples games played between trained neural network controllers (green robots, lighter dotted lines) and knowledge-based controllers (red robots, 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.

4.4 Discussion

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 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).

Evolved behavioral robotics control systems do not yet rival well designed sophisticated knowledge based controllers. Nonetheless, These results indicate that an evolve controller can beat a hand coded controller a fraction of the time.

The method of post-training controller evaluation does not influence the functionality of the relative tournament fitness function used during evolution of the controllers. This means it is possible to use post training fitness evaluation functions that are inadequate to select for the behavior, but can still measure behavior after it has been evolved. Also, we can evaluate the evolved controllers using human biases without imbedding such biased into the evolved controllers.

5. Conclusions and future research

In this paper a new evolutionary robotics testbed was described. 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 specific features of the game and could be used to evolve behaviors for other multi-robot tasks.

An evolved controller was experimentally tested using real robots in the real world. The evolved controller competed against a sophisticated hand designed knowledge based controller in a tournament and was able to win a fraction of the games.

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. It is also of interest to investigate the affects 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, Vol. 5, pp. 317-342, 1997.

[2] J. Xiao, Z. Mickalewicz, L. Zhang, K. Trojanowski, Adaptive Evolutionary Planner/Navigator for Mobile Robots, IEEE Transactions on Evolutionary Computing, Vol. 1, no. 1 pp. 18-28, 2000.

[3] S. Nolfi, Evolutionary Robotics: Exploiting the Full Power of Self-Organization, Connection Science, Vol. 10, pp. 167-183, 1998.

[4] M. Quinn, 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.

[5] 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)

[6] M. Potter, L. A. Meeden, A. Schultz, Heterogeneity in the coevolved behaviors of mobile robots: The emergence of specialists, Proceedings of the Seventeenth 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 and Tanaka, editors, Proc. of the Fourth International Symposium on Artificial Life and Robotics. Oita Univ. Press, 1999.

[8] 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

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

[10] 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, pp. 396-407, 1996.

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

[12] Richard A. Watson, Sevan G. Ficici, Jordan B. Pollack, Embodied Evolution: Distributing an Evolutionary Algorithm in a Population of Robots, Robotics and Autonomous Systems, Vol. 39 No. 1, pp 1-18, 2002.

[13] I. Harvey, P. Husbands, D. Cliff, A. Thompson and N. Jakobi, Evolutionary robotics: the Sussex approach, Robotics and Autonomous Systems, Vol. 20, No 2-4, pp. 205-224, 1997.

[14] M. Mataria, D. Cliff, Challenges in evolving controllers for physical robots, Robotics and Autonomous Systems , Vol. 19, No. 1, pp. 67-83, 1996.

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

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

[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] John Galeotti, The EvBot A Small Autonomous Mobile Robot for the Study of Evolutionary Algorithms in Distributed Robotics, MS Thesis, North Carolina State University, 2002.

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