Learning to do without synaptic plasticity: evolving adaptive neural networks controllers for autonomous robot

Elio Tuci, Matt Quinn and Inman Harvey
Centre for Computational Neurosciences and Robotics,
University of Sussex, Brighton BN1 9QH, United Kingdom
phone: +44 (0)1273 872945 - fax +44 (0)1273 671 320
eliot, matthewq, inmanh@sussex.ac.uk

Abstract- Recently research in the field of Evolutionary Robotics have begun to investigate the evolution of learning controllers for autonomous robots. Research in this area has achieved some promising results, but research to date has focussed on the evolution of neural networks incorporating synaptic plasticity. There has been little investigation of possible alternatives, although the importance of exploring such alternatives is recognised [8]. This paper describes a first step towards addressing this issue. Using networks with fixed synaptic weights and 'leaky integrator' neurons, we evolve robot controllers capable of learning and thus exploiting regularities occurring within their environment.

1 Introduction

Over the past ten years, an increasing number of researchers have applied artificial evolution approaches to the design of controllers for autonomous robots. The appeal of an evolutionary approach is two-fold. Firstly, and most basically, it offers the possibility of automating a complex design task [7, 9, 4]. Secondly, since artificial evolution needs neither to understand, nor to decompose a problem in order to find a solution, it offers the possibly of exploring regions of the design space that conventional design approaches are often constrained to ignore [5]. This type of approach, known as Evolutionary Robotics (ER), has proven successful in designing robots capable of performing a variety of non-trivial tasks [see 7, 9, 4, for a review of the field]. One of the more recent directions that ER research has taken, has been toward the evolution of robots that are capable of autonomous learning [e.g. 10, 8, 2, 3, 1]. The goal of this research is not the evolution of robots that can be trained by an externally imposed teaching input or reinforcement signal. Instead, the aim is to evolve robots that are capable of autonomously modifying their behaviour through interaction with the environment in which they are situated.

Research in this area has investigated the use of neural network controllers with some form of synaptic plasticity, that is, networks which incorporate mechanisms that change connection weights. These have typically been networks with Hebbian-type update rules, where the properties and type of rules used were placed under evolutionary control [e.g. 3, 1]. Non-Hebbian mechanisms have also been employed; for example, Nolfi and Parisi [10] evolved a control architecture in which a non-plastic network generated a reinforcement signal that served to modify the weights of a second network. The research on autonomous learning robot has generally focused on the evolution of controllers with the ability to modify their behaviour in order to adapt to variation in their operating conditions, that is, changes in the relationship between a robot's sensors and actuators, and its environment. Nolfi and Parisi [10] investigated a task in which a Khepera mini-robot equipped with infra-red (IR) sensors operated in an arena that could have either black walls (low IR reflectivity) or white walls (high IR reflectivity). There was thus a significant variation between the two conditions in the relationship between sensor, activation and wall proximity. Evolved controllers were shown to be able adapt to each of the conditions over a period interaction with their environment. In addition to being a general advantage to an autonomous robot, the ability to adapt to variations in operating conditions is also useful when controllers are evolved in simulation. The conditions to which a robot will encounter in the real world are unlikely to perfectly match those to which it was exposed in simulation. Floreano and Urzelai [3] and Eggenberger et al. [1] have both evolved controllers which have transferred well from simulation to reality by virtue of an ability to adapt to variation in their operating conditions. Floreano and Urzelai [2] have also successfully managed to transfer controllers evolved for the Khepera mini-robot on to another robot with different size and sensory layout.

As noted above, ER has focused on synaptic plasticity as a mechanism by which to achieve learning. However, synaptic plasticity is not the only mechanism that can underlie learning behaviour. For example, Yamauchi and Beer [12] have evolved continuous time recurrent neural networks (CTRNNs) with fixed connection weights that proved capable of learning in response to reinforcement. Nonetheless, alternatives to synaptic plasticity remain largely uninvestigated within ER. A better understanding of the alternatives and their potential strengths and weakness is an important issue for ER research [8]. The work described in this paper is intended as a first step towards addressing this issue.

In what follows, we will demonstrates that it is possible to evolve an integrated dynamic neural network that successfully controls the behaviour of a Khepera mini-robot engaged in a simple learning task. This learning task requires that the robot learn the relationship between the position of a light source and the location of its goal (see section 2 for a more detailed description). The robot must be able to perform the task in an environment in which moving toward the light source will take it toward its goal, but also in environment in which this relationship is inverted. Only though interacting with its environment will the robot be able to learn and thus exploit the particular relationship between goal and light. Rather than employing plastic synapses, we use a neural network controller with fixed connection weights and 'leaky integrator' neurons (i.e. neurons whose activation decays with time), and no reinforcement signal is explicitly hard-wired in to the system. We show that this network architecture is able to produce robots capable of performing the task (section 4).

2 Description of the task

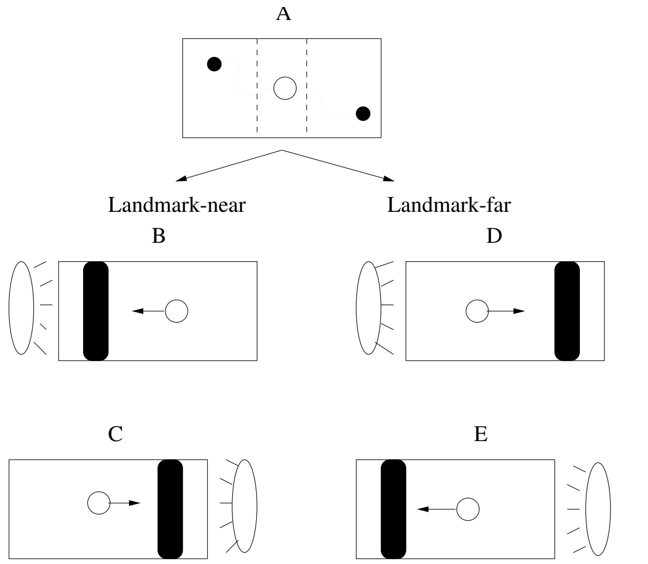

Figure 1: **Depiction of the task**. The small circle represents the robot. The white oval represents the landmark (i.e. a light source) and the black stripe in the arena is the goal. Picture A at the top represents the arena at the beginning of each single trial without landmark and goal. The black points represent two possible starting points. The dotted lines represent two possible routes to the central part of the arena (delimited by dashed lines). Pictures B and C represent the two possible arrangement of the landmark and goal within the landmark-near environment. The pictures D and E represent the two possible arrangement of the landmark and goal within the landmark-far environment. The arrows, in pictures B,C,D,E represent the directions towards which a successful robot should moves.

The task requires the robot to navigate within a rectangular arena. At the start of each trial, a robot is positioned in the left or right side of an empty arena (see the black points in fig 1A). As soon as the robot reaches the central part of the arena (delimited in fig 1A by dashed lines), a goal (represented by a black stripe) is randomly positioned on the white arena's floor. This can be either in front of, or behind the robot, and is not perceivable by its proximity sensors (see fig 1B, 1C, 1D, 1E). The robot's behaviour will be evaluated according to a fitness function that rewards or penalises certain behaviours; this evaluation will be used to determine how many offspring it has within the evolutionary algorithm, but is not in any sense a reinforcement directly available to the robot. The robot is rewarded for leaving its current central position towards the goal until it finds it (see arrows in fig 1B, 1C, 1D, 1E); but it is penalized if it sets off away from the goal. The robot must "decide" which way to go; however, without any clue as to where the goal is, it can do no better than making a random decision.

The robot is equipped with 6 infra red sensors (0 to 5) and 3 ambient light sensors (1, 4, 6). It also has a floor sensor indicated by the central gray circle.

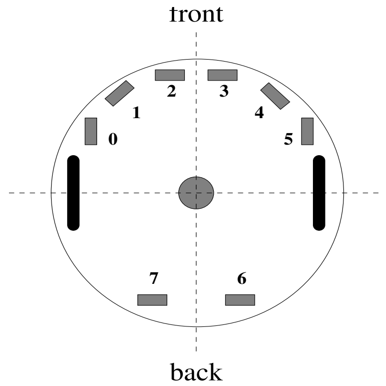

Figure 2: Plan of a Khepera mini-robot showing sensors and wheels.

The robot can use a landmark for guidance. The landmark is a light which lights up as soon as the goal is positioned, and it can be perceived by the robot from anywhere within the arena. The relationship between landmark and goal can vary; in some trials the landmark is close to the goal ('landmark-near environment' fig 1B and 1C), in other trials it is on the opposite side ('landmark-far environment' fig 1D and 1E). Both possibilities can be exploited by the robot to find the right way to the goal. If a robot approaches the landmark within the landmark-near environment, or moves away from the landmark within the landmark-far environment, it will consequently find the goal significantly more often than would be expected in a random walk. However, the current relationship between the landmark and the goal remains unknown to the robot until it has gathered enough experience to be able to disambiguate the environment between landmark-near or landmark-far. Therefore, the task requires a robot to learn, over a series of successive trials, which of these two landmark relationships currently holds true.

The robot undergoes two test sections in the landmark-near and two more in the landmark-far environment. Each time the environment changes, the robot's control system is reset, so that there is no internal state or "memory" remaining from the previous environment. Within each of these four test sections, a trial corresponds to the lapse of time that is given to the robot for reaching the middle of the arena and then finding the goal. The robot has 18 seconds to reach the middle of the arena. Then the landmark and the goal are positioned, and the robot has another 18 seconds to find the goal. Each trial can be terminated earlier either because the first time limit to reach the middle of the arena is exceeded; or because the agent reaches and remains on the goal for 10.0 seconds; or because the robot crashes into the arena wall. At the beginning of the first trial, and for every trial following an unsuccessful one the robot is positioned on either the left or the right part of an rectangular arena close to the short side. For every trial following a successful one, the robot normally keeps the position and orientation with which it ended the previous trial, but there is a small probability that it is replaced randomly. The simulation is deliberately noisy, with noise added to sensors and motors; this is also extended to the environment dimensions. Every time the robot is replaced, the width and length of the arena, and the width of the central area that triggers the appearance of the landmark and the goal, are redefined randomly within certain limits. The robot undergoes 14 trials for each test section. During this time the relationship between landmark and the goal is kept fixed; the relationship remains unknown to the robot unless and until it can 'discover' it through experience. The position of the goal within the arena (i.e. left or right), is randomly determined every single trial and the landmark is subsequently appropriately positioned (depending on whether it is currently landmark-near or landmark-far). Over a full set of trials, all the 4 landmark/goal combinations (fig 1B,fig 1C,fig 1D, fig 1E) occur with equal likelihood.

3 Implementation

3.1 Evaluation function

The robot is rewarded by the evaluation function F_{st} (per test section s, per trial t) for reaching the central area within the arena and then moving towards the goal until it find it (see arrows in fig 1B, 1C, 1D, 1E); but it is penalized either if it fails to reach the central area or if, once on the central area, it set off away from the goal.

$$F_{st} = (abcd)((3.0\frac{(d_f - d_n)}{d_f}) + \frac{p}{p_{max}})$$

with:

$s = [1, ..., 4]$, and $t = [2, ..., 14]$. The robot doesn't get any score during the first trial of each test section, because it is assumed that the robot doesn't know what kind of environment it is situated (i.e. landmark-near or landmark-far environment).

d_f represents the furthest distance that the robot reaches from the goal after the light is on. At the time when the light goes on, d_f is fixed as the distance between the centre of the robot body and the nearest point of the goal. After this, d_f is updated every time step if the new d_f is bigger than the previous one. d_n represents the nearest distance that the robot reaches from the goal after the light is on. At the time when the light goes on, d_n is fixed as equal to d_f, and it is subsequently updated every time step both when the robot gets closer to the goal and when d_f is updated. In the latter case d_n is set up equal to the new d_f.

p represents the number of steps that the robot makes into the goal during its longest period of permanence. $p_{max} = 25$ is the maximum number of steps that the robot is allowed to make into the goal before the trial is ended;

a is a bias term for the type of environment; it is set to 3 in landmark-near environment, and to 1 in landmark-far environment. b, c, d are the penalties. b is set to 0 if the robot fails to reach the central part of the arena, either because time limit is exceeded, or because it crashes into the arena wall; c is set to $\frac{1}{3}$ if the robot leaves the central area towards the opposite side of the arena respect to the goal (i.e unsuccessful behaviour); d is set to $\frac{1}{5}$ if the robot crashes into the arena wall; If the robot doesn't fall in any of these penalties, a, b, c are set to 1.

The fitness of each robot's genotype is its average evaluation score after it has been assessed by evaluating the robot performance in each single trail of each test section.

3.2 Simulated robot

A simple 2-dimensional model of a Khepera's robot-environment interactions within an arena was responsible for generating the base set aspects of the simulation [see 6, for a definition of base set]. The implementation of our simulator, both for the function that updates the position of the robot within the environment and for the function that calculates the values returned by the infra-red sensors and ambient light sensors, closely matches the way in which Jakobi designed his minimal simulation for a Khepera mini-robot within an infinite corridor [see 6, for a detailed description of the simulator]. During the simulation, robot sensor values are extrapolated from a look-up table. Noise is applied to each sensor reading. Our robot sensor ability is limited to its 6 front sensors, three ambient light sensors, positioned 45 degrees left and right and 180 degrees with respect to its face direction (see figure 2). The light is modelled as a bar that illuminates the whole arena with the same luminosity. Each ambient light sensor faces out from a point on the exterior of the robot and detects any light within plus or minus ± 30 degrees from the normal to the boundary. The values returned by the ambient light sensors when impinged by the light, are set up to 1 if they exceed a fixed threshold otherwise they return 0. Our robot has an extra floor sensor, that can be functionally simulated as an ambient light sensor, that returns 0 when the robot is positioned over a white floor and 1 for a black floor. The simulation was updated the equivalent of 5 times a second.

3.3 Neural Network

Robots were controlled by recurrent artificial neural networks, potentially of arbitrary size and connectivity. The thresholds, weights and decay parameters, and the size and connectivity of the network were all genetically determined. The neural network consisted of 10 input nodes, 4 output nodes (motors), and some (potentially variable) number of hidden nodes or "inter neurons", connected together by directional, excitatory and inhibitory weighted links. At any time-step, the output, O_t, a neuron is given by:

$$O_t = \begin{cases} 1 & \text{if } m_t \geq T \\ 0 & \text{if } m_t < T \end{cases}$$

where T is the neuron's threshold. m_t is a function of a neuron's weighted, summed input (s), and the value of m_{t-1} scaled by a temporal decay constant, such that:

$$m_t = \begin{cases} (\gamma_A)m_{t-1} + \sum_{n=0}^{N} w_n i_n & \text{if} \quad O_{t-1} = 0 \\ (\gamma_B)m_{t-1} + \sum_{n=0}^{N} w_n i_n & \text{if} \quad O_{t-1} = 1 \end{cases}$$

where the decay constants γ_A and γ_B are positive real numbers smaller than 1.0. w_n designates the weight of the connection from the n^{th} input (i_n) that scales that input.

Each sensor node outputs a real value (in the range [0.0:1.0]), which is simple linear scaling of the reading taken from its associated sensor. Motor outputs consist of a 'forward' and a 'reverse' node for each motor. The output, M_{out} of each motor nodes is a simple threshold function of its summed weighted inputs:

$$M_{out} = \begin{cases} 1 & \text{if} \sum_{n=0}^{N} w_n i_n > 0 \\ 0 & \text{if} \sum_{n=0}^{N} w_n i_n \leq 0 \end{cases}$$

The final output of the each of the two motors is attained by subtracting its reverse node output from its forward node output. This gives three possible values for each motor output: {-1,0,1}.

3.4 Encoding Scheme

The network is encoded by a topological encoding scheme developed by one of the authors (Quinn); this is designed to allow the size and connectivity of the network to be placed under evolutionary control. The procedure for encoding and decoding is as follows: Each neural network controller is encoded in a 'genotype', this consists of a list of 'genes'. Each gene encodes the parameters for an individual neuron and its associated connections. The form of each gene was { X, T, γ_A, γ_B, L_{in}, L_{out} }, where T, γ_A and γ_B are the (real-valued) threshold and decay parameters, and L_{in} and L_{out} are lists of input and of output connections respectively. X is an integer which serves an 'identity tag'. This is assigned to a gene upon its creation (genes are created either in the first GA generation when a population is created, or when a new gene is added to a genotype as a result of a macro-mutation). The identity tag (i.d.) assigned to a new gene is initially unique – no other gene will have the same i.d. at the time it is first assigned. The i.d. is not subject to mutation, but it *is* inherited. Hence if two genes (on two different genotypes) have the same i.d. it will be because both genes have a common ancestor. Sensor and motor nodes also have unique identity tags. The identity tags of the neurons, and the sensor and motor nodes, are used to determine network connectivity; the encoded connections specify a source or destination by reference to these tags. Each element in the connection lists, L_{in} and L_{out}, takes the form, $\{Z, w\}$, where w is the connection weight, and Z specifies an identity tag of the node (sensor, motor, or neuron) to which it should connect. In decoding the network, the source (or destination) of each connection in the list is determined by matching the tag it specifies with the i.d. of a neuron in the network, or that of a sensor or motor node. A connection may specify the tag of the neuron from which it originates, resulting in a recurrent connection being made. Note that Z is inherited and is subject to mutation (see below).

Genotypes are created to form an initial evolutionary population in the following manner. Each genotype was of length 3 genes and each gene was assigned a unique identity tag. Next, each gene was given between 0 and 7 input connections, and between 1 and 7 output connections. The number of connections was chosen at uniform random for each of the connection lists belonging to each gene. (Output lists were non-empty to ensure that each neuron was at least potentially functional.) Each connection element in each gene was then randomly assigned a tag, such that it was equally likely to connect to any neuron specified by the genotype or to any sensor node, or any motor node. Finally the real-valued parameters, i.e., all thresholds, decay parameters and connection weights were initialised to random values: weights and thresholds in the range [-5:5] and decay parameters in the range [0:1].

3.5 Evolutionary Algorithm and Genetic Operators

We used a simple generational genetic algorithm (GA), with a population size of 200. At each new generation, the highest scoring individuals ('the elite') were retained unchanged. The remainder of the new population was generated by fitness-proportional selection from the 150 best individuals of the old population. 60% of new genotypes were produced by recombination, and mutation operators were applied to all genotypes except the elite.

Mutation consisted of both macro-mutation (i.e. structural changes) and micro-mutation (i.e. perturbation of real-valued parameters). Micro-mutation entailed that a random Gaussian offset was applied to each real-valued parameter encoded in the genotype with a small probability, such that the expected number of micro-mutations per genotype is 0.75. The mean of the Gaussian is 0, and its s.d is 0.5 of that parameter's initialisation range (specified above). The threshold and weight parameters were unbounded, but the decay constants were restricted to their initialisation range of [0:1]. In cases where the mutated value of a bounded parameter fell outside a bound, its value was set at uniform random to a value between the bound and its pre-mutated value.

Three types of macro-mutation are employed. Firstly, gene could be added or deleted from the genotype. In addition mutation, a new gene is created and added to the genotype. Gene creation follows the same procedure as described above, except that the maximum number of connections per connection list is restricted to two in order to minimize the disruptiveness of the addition operation. With a deletion operation, one gene is selected at uniform random and removed from the genotype (deletion is not carried out if it would result in an empty genotype). Gene addition occurs with a probability of 0.005 and gene deletion with a probability of 0.02 per genotype. The second type of macro-mutation involves the addition and deletion

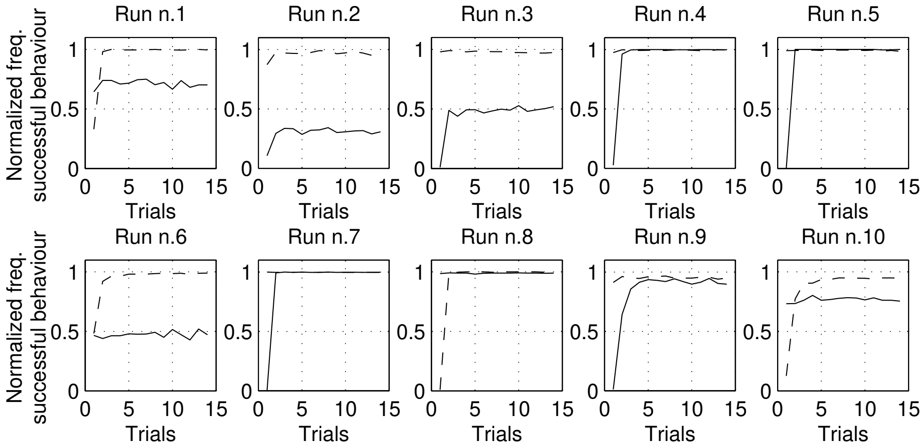

Figure 3: Performance of the best control system of the final generation of each of the 10 runs, evaluated over 500 test sessions in each of two environment types. Normalized frequencies of successful behaviour per trial are shown for for each type of environment on the same axis. Dashed lines indicate the performance recorded in the landmark-near environment. Continuous lines indicate the performance recorded in the landmark-far environment.

of connections. With addition, a new connection is created (as set out above), it has an equal probability of being an input or an output, and is added to the appropriate connection list of a randomly chosen gene. In the case of deletion, a gene is selected at uniform random from the genotype, and one of its two connection lists is chosen with equal probability, if the list is not empty, a connection element is selected (again at uniform random) and deleted. Connection addition and deletion both occur with an independent probability of 0.05 per genotype. Finally, with a probability of 0.05, the genotype is subject to a reconnection mutation: one gene is selected at uniform random, and one of its connection list is then randomly selected and a connection element chosen (again at uniform random). This connection is then reconnected, by randomly assigning it a tag such that it is equally likely to connect to any neuron specified by the genotype, any sensor node or any motor node.

The recombination operator utilises gene's i.d. tags, rather than their position on the genotype, in order to maintain the structural integrity of the resulting genotypes. The i.d. tag is used by the recombination operator to pair genes with a common ancestor, and thereby help to ensure that genes with similar phenotypic functionality crossed. Crossover takes two parent genotypes and produces two offspring, as follows. All genes that can be, are paired by identity number. Each pairing is then crossed with a probability of 0.5. The remaining un-paired genes are not crossed.

4 Results

Ten evolutionary simulations, each using a different random seed, were run for 5000 generations. We examined the best individual of the final generation from each of these runs in order to establish whether they had evolved an appropriate learning strategy. Recall that to perform its task successfully the robot make use of the light source in order to navigate towards its goal. Thus, over the course of a test session, the robot must learn—and come to exploit—the relationship between the light source and the goal that exists in the particular environment in which it is placed. In order to test their ability to do this, each of these final generation controllers was subjected to 500 test sessions in each of the two types of environment (i.e. landmark-near and landmark-far). As before each session comprised 14 consecutive trials, and controllers were reset before starting a new session. During these evaluations we recorded the number of times the controller successfully navigated to the target (i.e. found the target directly, without first going to the wrong end of the arena).

The results for the best controller of each of the ten runs can be seen in figure 3, which shows the normalized frequency of successful behaviour for each consecutive trial during a session under each environmental condition. It is clear that in five of the runs (i.e. 4, 5, 7, 8 and 9) robots quickly come to successfully use the light to navigate toward the goal, irrespective of whether they are in an environment which requires moving towards or away from the light source. Four of these controllers ,runs 4, 5, 7 and 9, employ a default strategy of moving toward the light, and hence are always successful in the landmark-near environment. Consequently they also initially very unsuccessful in the landmark-far environment. Nevertheless, as can be seen from figure 3, when these controllers are placed in the land-mark far environment they are capable of adapting their behaviour over the course of very few trials and subsequently come to behave very differently with respect to the light. Each of these controllers, when placed in the landmark-far environment, initially navigates toward the light, fails to encounter the target and subsequently reaches the wrong end of the arena. At this point it turns around and proceeds to back toward the correct end of the arena where it ultimately encounters the target stripe. The fact that in subsequent trials the robot moves away from the light rather than towards it, demonstrates that the robot has learn from some aspect, or aspects, of its earlier experience of the current environment. More prosaically, these controllers learn from their mistakes. The same can be said of the controller from run 8, which differs only in the respect that it adopts the opposite initial strategy (i.e. navigating away from the light) which it subsequently modifies when placed in landmark-far environment.

It should be noted that the controllers from the other five runs (i.e. 1, 2, 3, 6 and 10), whilst performing the task with varying degrees of success, all modify their behaviour over the course of the initial trails; in each case this results in improved performance. For example, the controller from run 6 initially has a 50% success rate in both types of environment. In the landmark-far environment its success rate remains constant. However, if placed in the landmark-near environment its success rate will climb from 50% to 100% within the course a few trials. Similarly, with the controller from run 1, success in a landmark-near environment increases rapidly from around 25% up to 100% whilst success in the landmark-far remains consistently around 75%.

5 Discussion and Conclusion

The evolution of controllers capable of autonomous learning is a relatively recent research direction for the field of evolutionary robotics. Nonetheless, research in this area has achieved some promising results [e.g. 10, 8, 2, 3, 1]. However, research to date has focussed on the evolution of neural networks incorporating synaptic plasticity. There has been little investigation of possible alternatives, although the importance of exploring such alternatives is recognised [8]. The work described in this paper has been a first step towards addressing this issue. It is intended as a sort of existence proof, a demonstration that that neural networks with fixed connection weights and temporal dynamics can be evolved to produce learning behaviour. Using networks with fixed synaptic weights and 'leaky integrator' neurons, we have evolved robot controllers capable of adapting their behaviour in order to exploit regularities occurring within their environment. Whilst similar networks have been evolved to perform learning with external reinforcement [12, 11], we believe this is the first example of such a network being evolved to learn in the absence of teaching input.

Demonstrating the existence of a viable alternative to synaptic plasticity is a useful first step toward a better general understanding of how evolution may be used to construct learning controllers. However, in isolation, the experiments described give us little indication of the relative merits of the approach when compared to approaches incorporating synaptic plasticity. Our next step is to investigate the evolution of controllers with synaptic plasticity on the task described in this paper, in order to begin to investigate the relative merits of the two approaches.

Bibliography

[1] Eggenberger, P., Ishiguro, A., Tokura, S., Kondo, T., and Uchikawa, Y. (1999). Toward seamless transfer from simulated to real worlds: A dynamically-rearranging neural network approach. In Watt, J. and Demiris, J., editors, *Proc. 8th European Workshop on Learning Robots*, Lausanne, Switzerland. Springer.

[2] Floreano, D. and Urzelai, J. (2000). Evolutionary Robots: The Next Generation. In *Proc. 7th Intl. Symposium on Evolutionary Robotics*.

[3] Floreano, D. and Urzelai, J. (2001). Evolution of plastic control networks. *Autonomous Robots*. in press.

[4] Harvey, I., Husband, P., Thompson, A., and Jakobi, N. (1997). Evolutionary Robotics: the Sussex approach. *Robotics and Autonomous Systems*, 20:205–224.

[5] Harvey, I., Husbands, P., and Cliff, D. (1992). Issues in evolutionary robotics. In Meyer, J.-A., Roitblat, H., and Wilson, S., editors, *Proc. 2nd Intl. Conf. on Simulation of Adaptive Behavior*, pages 364–373. MIT Press/Bradford Books.

[6] Jakobi, N. (1997). Evolutionary robotics and the radical envelope of noise hypothesis. *Adaptive Behavior*, 6:325–368.

[7] Meyer, J.-A., Husbands, P., and Harvey, I. (1998). Evolutionary Robotics: A survey of Applications and Problems. In Husbands, P. and Meyer, J.-A., editors, *Proc. 1st European Workshop on Evolutionary Robotics*. Springer.

[8] Nolfi, S. and Floreano, D. (1999). Learning and evolution. *Autunomous Robots*, 7(1):89–113.

[9] Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics*. MA: MIT Press/Bradford Books.

[10] Nolfi, S. and Parisi, D. (1997). Learning to adapt to changing environments in evolving neural networks. *Adaptive Behavior*, 5(1):75–98.

[11] Yamauchi, B. and Beer, R. (1994a). Integrating Reactive, Sequential, and Learning Behavior Using Dynamical Neural Network. In Cliff, D., Husbands, P., Meyer, J.-A., and Wilson, S. W., editors, *Proc. 3rd Intl. Conf. on Simulation of Adaptive Behavior*.

[12] Yamauchi, B. M. and Beer, R. D. (1994b). Sequential Behavior and Learning in Evolved Dynamical Neural Networks. *Adaptive Behavior*, 2(3):219–246.