Presented at the International Symposium on Robotics
Montreal, Canada, 14–17 May 2000

A Step in the Right Direction
Learning Hexapod Gaits Through Reinforcement

T D Barfoot
Institute for Aerospace Studies
University of Toronto
4925 Dufferin Street
Toronto, ON, Canada, M3H 5T6
tdb@sdr.utias.utoronto.ca

E J P Earon
Institute for Aerospace Studies
University of Toronto
4925 Dufferin Street
Toronto, ON, Canada, M3H 5T6
e.earon@utoronto.ca

G M T D'Eleuterio
Institute for Aerospace Studies
University of Toronto
4925 Dufferin Street
Toronto, ON, Canada, M3H 5T6
gde@utias.utoronto.ca

Abstract

A simple reinforcement learning approach to developing walking gaits for a legged robot is presented. Each leg of the robot is given its own controller in the form of a cellular automaton which serves to arbitrate between a number of fixed basis behaviours. Communication exists between the legs. A multiagent cooperative Q-learning *approach is presented and employed to search for a set of cellular automata whose arbitration results in successful walking gaits on a real hexapod robot.*

1 Introduction

Insects and spiders are examples of relatively simple creatures from nature which are able to successfully operate many legs at once in order to navigate a diversity of terrains. Inspired by these biological marvels, robotics researchers have attempted to mimic insect-like behaviour in legged robots [4, 2]. Typically, however, the control algorithms for these types of robots are quite complicated (e.g., dynamic neural networks), requiring fairly heavy on-line computations to be performed in real time. Here a simpler approach is presented which reduces the needs for such computations by using a coarsely coded control scheme.

As with dynamic neural network approaches, each leg of a robot is given its own controller rather than using a central pattern generator. Coupling between controllers can be sparse (legs communicate with only some other legs) or full (legs communicate with all other legs). In either circumstance the control is still carried out in a local fashion which results in some global behaviour (walking gait) for the robot. There is some neurobiological evidence that this notion of local control holds for some insects [5]. The controllers used here are cellular automata [13, 11, 16, 8] or simple lookup tables. Each leg is given its own cellular automaton (CA) which can be in one of a finite number of states. Based on its state and those to which it is coupled, a *basis behaviour*, or action, is chosen according to the lookup table. The basis behaviour leads the leg to a new state from the old state. Typically all legs are updated synchronously such that the robot's behaviour is represented by a set of locally coupled difference equations (rather than differential equations), which are simple to compute in real time.

Under this framework, design of a control algorithm is reduced to coming up with the cellular automata which produce successful walking gaits. Our approach has been to use a modified version of Q-learning [14] (a form of reinforcement learning) to this end [12]. It should be noted that although the model described here does not explicitly take into account the dynamics (physics) of walking robots, they may be accounted for implicitly if the control algorithms are learned directly on hardware.

2 Cellular Automata

According to neurobiological evidence described by Cruse [5], the behaviour of legs in stick insects is locally coupled as in figure 1 (left). This pattern of ipsalateral and contralateral connections will be adopted for the purposes of discussion although any pattern could be used in general (only some of them would work).

We assume that the output of each leg controller may be discrete. This may be done by way of a set of *basis behaviours* [9, 1]. Rather than specify the actuator positions (or velocities) for all times, we assume that we may select a simple behaviour from a finite predefined palette. This may be considered a postprocessing step which takes a discretized output and converts it to the actuator control. This postprocessing step will not be allowed to change once set. The actual construction of the postprocessing requires careful consideration but is also somewhat arbitrary. Here the basis behaviours will be modules which move the leg from its current *zone* or state (in output space) to one of a finite number of other zones. Figure 2 shows two possible discretizations of a 2-degree-of-freedom output space (corresponding to a simple leg) into 4 or 3 zones. It is important to distinguish between a basis behaviour and a leg's current zone. For our experiments we will use only two basis behaviours ($M = 2$): (stay in current zone) or (move to next zone in counterclockwise fashion). This defines the ideal transition function for the robot which will naturally be modified by noise (unsuccessful control actions). By using basis behaviours, the leg controllers may be entirely discrete.

Once all the postprocessing has been set up, the challenge remains to find an appropriate arbitration scheme which takes in a discrete input state, s, (current leg positions or basis behaviours of self and neighbours) and outputs the appropriate discrete output, a, (one of M basis behaviours) for each leg. There are several candidates for this role but the one affording the most general decision surfaces between input and output is a straightforward lookup-table similar to *cellular automata* (CA)

$$a = \phi(s)$$

This type of lookup-table control in autonomous robots is often called *reactive*. For every possible input sequence the CA scheme stores a discrete output value. In other words, for every possible input sequence there is an output corresponding to one of the basis behaviours. At each time-step, the leg controller looks up the action which corresponds to its current input sequence and carries it out. The size of the lookup-table for a leg which communicates with $K - 1$ other legs will then be M^K such that the approach is therefore usually rendered feasible for only modest numbers for K and M. The number of all possible lookup tables is $M^{(M^K)}$. Again, modest numbers of basis behaviours keep the size of the search space reasonable. For example, with a hexapod robot with coupling as in figure 1 (left) and output discretization as in figure 2 (left) the forward and rear legs will require lookup tables of size 4^3 and the central legs 4^4. If we assume left-right pairs of legs have identical controllers the combined size of the lookup-tables for forward, center, and rear legs will be $4^3 + 4^3 + 4^4 = 384$ and the number of possible table combinations for the entire robot will be 4^{384}. From this point on, the term *CA lookup-table* will refer to the combined set of tables for all legs in the robot (concatenation of individual leg lookup-tables).

3 Example Controller

One well established gait for hexapod walkers is called tripod. The legs are divided into two sets, {Forward Left, Center Right, Rear Left} and {Forward Right, Center Left, Rear Right}. While one set is on the ground, the other is lifted, swung forward, and then lowered. This is repeated, alternating the set to be lifted. If we use the output discretization of figure 2 (left), then a tripod gait could be represented by one set of legs cycling through zones $ABDDCCABDDCC\ldots$ while the other cycled through $DCCABDDCCABD\ldots$ (out of phase with first set).

For this example, a genetic algorithm was employed to look for controllers which eventually got to a tripod gait from as many initial conditions as possible. For more information on this experiment see [6]. The best solution we found in this way was labelled ϕ_{tripod} which got to a tripod gait from 98.4% of all possible initial conditions.

Figure 3 depicts two aspects of ϕ_{tripod}. The left side shows a typical time history or *gait diagram* of ϕ_{tripod} on a particular initial condition. Each column shows the states of the 6 legs at a given time-step. The leftmost column is the initial condition. The right side of figure 3 is an attractor basin portrait of ϕ_{tripod} as described by [17]. Each node in this plot represents an entire 6 state leg configuration of the robot (i.e. one column of the left plot). Lines between nodes represent transitions from one leg configuration to another (as specified by ϕ_{tripod}). Transitions only occur towards the center of the attractor basin. The inner hexagon represents the $ABDDCC\ldots$ tripod gait which is unidirectional (indicated by a clockwise arrow). The purpose of the right plot is to draw a picture of ϕ_{tripod} as a whole and to make a connection with the concept of stability. Beginning from any node on the attractor basin portrait, and following the transitions inward (one per time-step), one will always wind up on the inner hexagon (tripod gait).

The conclusion we may draw from this simple exercise is that this type of controller certainly is able to produce patterns resembling known walking gaits. We could use a genetic algorithm to attempt to evolve walking gaits directly on the hardware [6] but instead we now turn to another approach which we hope will be more expedient, reinforcement learning [12].

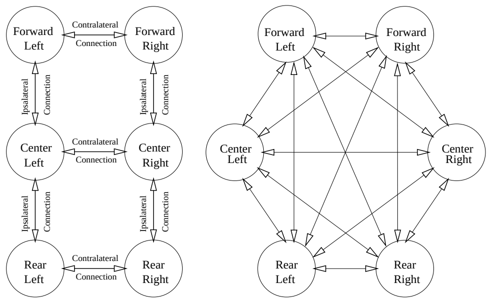

Figure 1: Behavioural coupling between legs. (left) Sparse coupling as in stick insects [5] (right) Full coupling.

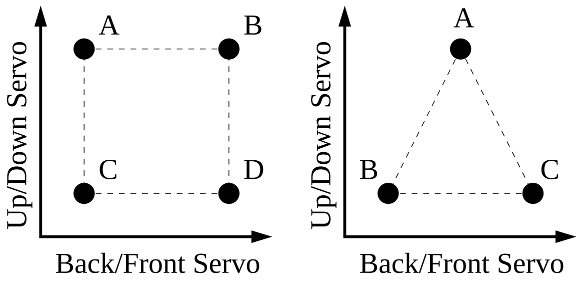

Figure 2: Example discretizations of output space for 2 degree of freedom legs into (left) 4 zones and (right) 3 zones.

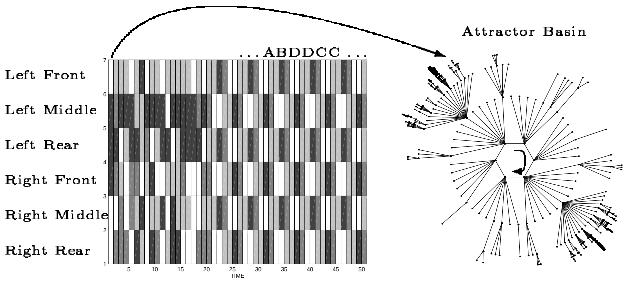

Figure 3: (left) Time history of ϕ_{tripod} on a particular initial condition. Each 6 state column represents an entire leg configuration of the robot; the left-most column is the initial condition. (right) Attractor basin portrait of ϕ_{tripod}. Each node represents an entire 6 state leg configuration of the robot. Lines represent transitions from one leg configuration to another. The inner hexagon represents the $ABDDCC\ldots$ tripod gait.

4 Reinforcement Learning on a Hexapod

Reinforcement learning was originally derived from dynamic programming and is treated well in the literature [12]. We assume that the problem we are trying to solve may be modeled as a *Markov decision process*

$$(\Omega_S, \Omega_A, \delta, r)$$

Here Ω_S is the state space, Ω_A is the action space, $\delta(s,a)$ is the state transition function (probabilistic) where $s \in \Omega_S$ and $a \in \Omega_A$, and $r(s,a)$ is the reward function. We will be attempting to maximize the *discounted cumulative reward function* given as follows

$$R(t) = r(s_0,a_0) + \gamma r(s_1,a_1) + \gamma^2 r(s_2,a_2) + \ldots$$

where $s_i = s(t+i)$ and $a_i = a(t+i)$. The Markov decision process and the discounted cumulative reward function together form a *Markov decision problem* (MDP) which Bellman [3] showed to have an optimal solution. We will be using a variant of reinforcement learning, Q-learning [14], which allows one to learn an optimal decision making policy

$$\pi^*(s) = \arg\max_a Q(s,a)$$

to the MDP. It has been shown to converge under certain conditions [14] but also to work reasonably well when these conditions are only weakly met [9] as is often the case in real robotics applications.

Here s is the state and a is a basis behaviour. The state action matrix, $Q(s,a)$, is a grid of size $|s| = M^K$ by $|a| = M$. The relation between the cellular automata maps discussed above, $\phi(s)$, and the reinforcement learning policy is as follows

$$\phi(s) = \pi^*(s) = \arg\max_a Q(s,a)$$

As in the cellular automata discussion above, we will be using a policy for each leg such that control is still distributed. However, only three of these will be unique as identical policies will be used for pairs of legs joined by contralateral connections. The three policies will be termed Q_f, Q_c, and Q_r for 'forward', 'center', and 'rear', respectively. Coupling between legs will be as in figure 1 (right). This type of setup is generally known as a *multiagent* reinforcement learning problem as we are trying to learn more than one policy simultaneously and the abilities of the policies are interdependent [15]. There are many issues associated with the problem described here including non-Markovian states, partial observability [7], and non-determinism but we will forge ahead in spite of these to see what comes of our attempt.

For rewards we will be using the net distance travelled (positive is forward, negative is backwards) by the hexapod robot. All three policies will be receiving the same reward. The task will be to optimize how fast the robot can walk forwards on the treadmill. Mathematically we will be looking for an optimal solution to the MDP. Note, all three policies will have to work together to produce an overall behaviour which propels the robot forward. Rewards will be assessed at the end of each action taken (each "step" of the robot). Typically in Q-learning, each policy is updated according to

$$Q(s,a) \leftarrow r(s,a) + \gamma \max_{a'} Q(s',a')$$

where $s = s(t)$ is the old state, $s' = s(t+1)$ is the new state, $r(s,a)$ is the reward for starting at s and selecting action a, and $\gamma \in [0,1)$ is the discount constant.

Unfortunately, this algorithm does not suffice for our purposes as we are working with multiagent system where the reward is based on the performance of the entire system (not just one leg). Specifically, the reward depends on the actions of all six legs, not just one of them and as a result, a leg could receive very different values for the reward associated with following an action from a given state, $r(s,a)$ (depending on what the other legs decide). It does not make sense to average the rewards (as in stochastic Q-learning) as this can lead to suboptimal policies. Instead, a memory is introduced which keeps track of the best reward earned by following action a from state s,

$$r_{max}(s,a)$$

which has the same dimensions as $Q(s,a)$. We call the modified learning algorithm, *cooperative Q-learning*, as it promotes cooperation between concurrent learners (who are receiving identical rewards). The cooperative Q-learning algorithm is as follows

$$\begin{array}{rcl} r_{max}(s,a) & \leftarrow & \max\left[r_{max}(s,a), r(s,a)\right] \\ Q(s,a) & \leftarrow & r_{max}(s,a) + \gamma \max_{a'} Q(s',a') \end{array}$$

where $s = s(t)$, $a = a(t)$, $s' = s(t+1)$.

5 Experimental Results

This section briefly describes implementation of the above algorithm on Kafka [10], a 12 degree of freedom, hexapod robot. Figure 4 shows Kafka in action on a treadmill. The robot was mounted on an unmotorized treadmill in order to measure controller performance (for walking in a straight line only). As Kafka walks, the belt on the treadmill causes the rollers to rotate. An odometer reading from the rear roller is fed to Kafka's computer such that distance versus time-step plots may be used to determine the performance of the controller.

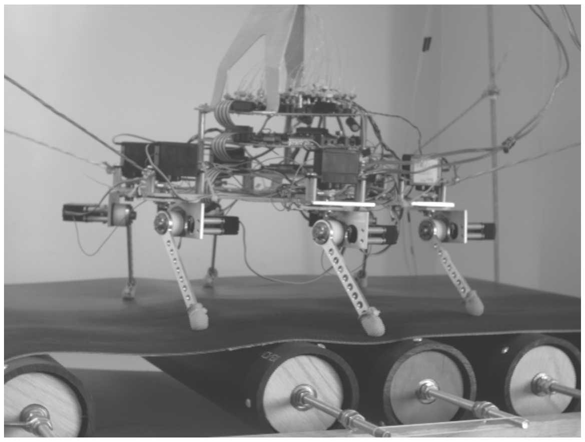

Figure 4: A hexapod robot/treadmill setup designed to evolve walking gaits.

Cooperative Q-learning was tested on Kafka using the leg coupling of figure 1 (right) and the leg discretization of figure 2 (right). To keep the size of the search space small, only two actions were permitted for each leg: (stay in the current state) or (move one state forward in the counter-clockwise cycle, $ABCABCABC\ldots$). Furthermore, the learning occurred in an episodal manner. An episode consisted of starting the robot in the configuration $BCBCBC$ and allowing it 10 time-steps to choose actions. A typical run consisted of 300 episodes. This episodal-style training sped up the experiment by allowing the robot to only learn part of the Q-matrix rather than learning it in its entirety.

The last issue to discuss is the trade-off between exploration of new actions and exploitation of the best known actions. Typically in Q-learning, new actions are explored with some probability, $p_{explore}$, otherwise the best action is chosen according to the Q-matrix. Here we used a linear time-varying $p_{explore}$ as depicted in figure 5 (bottom). This allowed many new actions to be explored initially and fewer later on. This is of course is a heuristic and often needs to be "tuned" in Q-learning experiments.

Figure 5 (top) shows the odometer position over a typical run of 3000 time-steps. Figure 5 (middle) shows the slope of the top figure (as computed by fitting a cubic to the top figure and taking the analytical derivative). The middle figure approximately represents the instantaneous speed of the robot which we

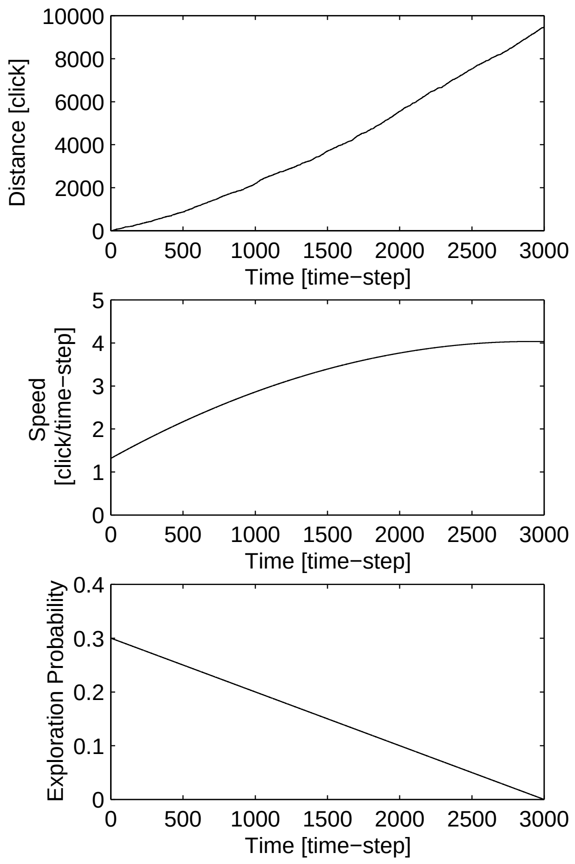

Figure 5: Cooperative Q-learning Performance. (top) Odometer position versus time, (middle) slope of top curve as computed by fitting a cubic to the data, (bottom) exploration probability versus time.

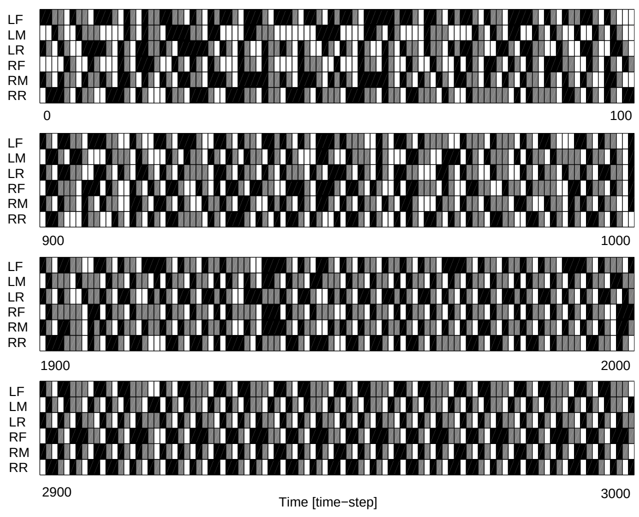

Figure 6: Gait diagrams at various times throughout a 3000 time-step run.

can see improves significantly over time to a level comparable but slightly below that of a tripod gait (around 6 click/time-step). The reason the performance does not quite meet that of a hand-designed tripod gait is most likely because we did not allow the experiment to continue past 3000 time-steps (the exploration probability went to zero here).

Figure 6 shows the time history or gait diagrams of the legs for four time intervals of 100 time-steps each. We can see the pattern becoming more regular at later times. By the end of the experiment, Kafka had learned a gait which was similar in performance and nature to the tripod.

6 Conclusion

Many researchers believe that highly parallel and decentralized methods are the key to endowing artificial systems with intelligence. Decentralized controllers for insect robots offer a great deal of redundancy in that if one controller fails, the robot may still limp along under the power of the remaining functional legs [4]. The cellular automata controller approach outlined here was able to successfully control Kafka, a hexapod robot, and should easily extend to robots with more degrees of freedom. One advantage of using such a coarse controller is that it requires very few real-time computations to be made (compared to dynamic neural network approaches) as each leg is simply looking up its action in a table.

The cooperative Q-learning algorithm was shown to be successful at concurrently learning cellular automata controllers from a single reward source (treadmill odometer). These preliminary results are encouraging but certainly more exhaustive testing is required of this algorithm. As with many such machine learning approaches, this algorithm may have difficulties scaling up to large problems due to the curse of dimensionality.

This work was motivated by evidence that biological insects generate walking patterns by means of decentralized control [5]. We hope that studying such types of control for artificial walkers may in turn tell us something about the natural systems by which they were inspired.

Acknowledgments

Kafka, the hexapod robot featured in this work, was constructed by David McMillen as part of a Master's thesis at the University of Toronto Institute for Aerospace Studies.

We would like to thank the Natural Sciences and Engineering Research Council of Canada and the Canadian Space Agency for supporting this work.

References

[1] T D Barfoot and G M T D'Eleuterio. An evolutionary approach to multiagent heap formation. Congress on Evolutionary Computation, July 6-9 1999.

[2] Randall D. Beer, Hillel J. Chiel, and Leon S. Sterling. A biological perspective on autonomous agent design. *Robotics and Autonomous Systems*, 6:169–186, 1990.

[3] R Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[4] Hillel J. Chiel, Randall D. Beer, Roger D. Quinn, and Kenneth S. Espenschied. Robustness of a distributed neural network controller for locomotion in a hexapod robot. *IEEE Transactions on Robotics and Autonomation*, 8(3):292–303, june 1992.

[5] Holk Cruse. Coordination of leg movement in walking animals. In J. A. Meyer and S. Wilson, editors, *Simulation of Adaptive Behaviour: from Animals to Animats.* MIT Press, 1990.

[6] E J P Earon, T D Barfoot, and G M T D'Eleuterio. From the sea to the sidewalk: The evolution of hexapod walking gaits by a genetic algorithm. Submitted to: Edinburgh, Scotland, 2000. Third International Conference on Evolvable Systems.

[7] Leslie Pack Kaebling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 1998.

[8] Chris G. Langton. Computations at the edge of chaos: Phase transitions and emergent computation. *Physica D*, 42:12–37, 1990.

[9] Maja J. Matarić. Behaviour-based control: Examples from navigation, learning, and group behaviour. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2):232–336, 1997. Special Issue on Software Architectures for Physical Agents, Editors: H. Hexmoor, I. Horswill, D. Kortenkamp.

[10] David Ross McMillen. Kafka: A hexapod robot. Master's thesis, University of Toronto Institute for Aerospace Studies, 1995.

[11] Melanie Mitchell and Stephanie Forrest. Genetic algorithms and artificial life. In Chris G. Langton, editor, *Artificial Life: An Overview.* MIT Press, 1995. SFI Working Paper 93-11-072.

[12] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* A Bradford Book, MIT Press, 1998.

[13] Jon von Neumann. *Theory of Self-Reproducing Automata.* University of Illinois Press, Urbana and London, 1966.

[14] C J C H Watkins. *Learning From Delayed Rewards.* PhD thesis, Cambridge University, Cambridge, England, 1989.

[15] Gerhard Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* MIT Press, Cambridge, Massachusetts, 1999. QA 76.76.I5 M85 1999.

[16] Stephen Wolfram. *Cellular Automata and Complexity: Collected Papers.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.

[17] Andrew Wuensche. The ghost in the machine: Basins of attraction of random boolean networks. In Chris G. Langton, editor, *Artificial Life III: SFI Studies in the Sciences of Complexity, vol. XVII.* Addison-Wesley, 1994.