An Architecture for Behavior-Based Reinforcement Learning

G.D. Konidaris G.M. Hayes

Institute of Perception, Action and Behaviour
School of Informatics, University of Edinburgh
James Clerk Maxwell Building, King's Buildings
Mayfield Road, Edinburgh EH9 3JZ
Scotland, UK

Abstract

This paper introduces an integration of reinforcement learning and behavior-based control designed to produce real-time learning in situated agents. The model layers a distributed and asynchronous reinforcement learning algorithm over a learned topological map and standard behavioral substrate to create a reinforcement learning complex. The topological map creates a small and task-relevant state space that aims to make learning feasible, while the distributed and asynchronous aspects of the architecture make it compatible with behavior-based design principles.

We present the design, implementation and results of an experiment that requires a mobile robot to perform puck foraging in three artificial arenas using the new model, random decision making, and layered standard reinforcement learning. The results show that our model is able to learn rapidly on a real robot in a real environment, learning and adapting to change more quickly than both alternatives. We show that the robot is able to make the best choices it can given its drives and experiences using only local decisions and therefore displays planning behavior without the use of classical planning techniques.

Keywords: Artificial Intelligence, Robotics, Reinforcement Learning, Layered Learning

1 Introduction

Any credible theory of intelligence must explain the wide spectrum of learning behavior displayed by insects, animals and humans. While some aspects of an autonomous agent can be evolved or directly engineered, other elements of behavior require learning because they involve knowledge that can only be gained by the agent itself, or that may change in unpredictable ways over its lifetime. Although behavior-based robotics has had some success as a basis for the development of intelligent, autonomous robots, the way in which learning fits into the behavior-based framework is not yet well understood.

Reinforcement learning is well suited to the kinds of problems faced by the current generation of behavior-based robots – Sutton (1990) has even argued that the problem facing an autonomous agent *is* the reinforcement learning problem. Reinforcement learning provides goal-directed learning without requiring an external teacher, handles environments that are not deterministic and rewards that require multiple steps to obtain, and has a well-developed theoretical framework (Sutton & Barto, 1998). Because of this, several researchers have included reinforcement learning in their robots. However, these have either involved using reinforcement learning over the robot's sensor space, and have thus suffered from scaling problems (e.g., Mahadevan and Connell (1992)) or have not involved real robots at all (e.g., Sutton (1990)).

This paper introduces a reinforcement learning architecture that is designed specifically for use in situated agents, motivated by the behavior-based emphasis on layered competencies and distributed control. It represents a full integration of behavior-based control and reinforcement learning, rather than a simple combination of the two methodologies, and is novel for three reasons. First, it layers reinforcement learning over a learned topological map, rather than using the robot's sensory space directly as the reinforcement learning state space. This leads to a small, task-relevant state space supported by the behavioral substrate already present on the robot, and adheres to the behavior-based emphasis on layered competence. Second, the model is distributed, potentially allowing it to take advantage of parallel hardware and layer over a distributed topological map and control system. Finally, learning is asynchronous, in that it performs reinforcement learning updates *all the time, in parallel at each state* instead of only after state transitions. This takes advantage of the fact that for situated agents, updates can be performed very much faster (especially when they are done in parallel) than state transitions can, because transitions require mechanical effort and time.

We present an experiment aimed at determining whether or not the model is feasible and effective, in which a mobile robot in an artificial arena is required to learn to find a food puck, explore, and return home guided by internal drives expressed as reinforcement functions. We then outline the development of Dangerous Beans, a mobile robot capable of performing the experimental task using an implementation of the architecture presented here, as well as random decision making and layered Q-learning. The results obtained demonstrate that the new model is capable of rapid learning, resulting in behavioral benefits on a real robot in real time and outperforming both alternatives. We further show that the reinforcement learning complex converges *between decisions*, and thus the actions taken by the robot are the best it can given the experience that it has. We thus argue that Dangerous Beans displays planning behavior.

2 Background

Behavior-based robotics and reinforcement learning are both well developed fields with rich bodies of literature documenting a wide range of research. The following sections briefly cover the related literature in both fields, and outline the concept of layered learning.

2.1 Behavior-Based Robotics and Learning

Behavior-based robotics emphasises the construction of complete, functional agents that must exist in the real world. Agents that exist within and interact with such complex environments in real time are known as *situated agents*, and must confront the issues of real time control and the complexity of the world directly – they must behave in the real world in real time.

One of the consequences of this change in emphasis has been the development of a different set of research concerns than those traditionally considered important in artificial intelligence. Behavior-based robotics emphasises the use of distributed, parallel and primarily reactive control processes, the emergence of complex behavior through the interaction of these processes with each other and the environment, cheap computation, and the construction of agents through the layered addition of complete and functional behavioral levels (Brooks, 1991a). The last point facilitates the incremental construction of mobile robots and explicitly seeks to mimic the evolutionary development of behavioral complexity.

Although behavior-based robotics has produced working robots for a variety of interesting problems, it has had difficulty developing systems that display a level of intelligence beyond that of insects. One of the possible reasons for this is that there has been no thorough investigation into the integration of learning into behavior-based systems. Behavior-based learning models would be required to be autonomous, distributed, layered on top of an existing behavioral substrate, and capable of learning in real time. Brooks (1991b) has argued that the traditional approach to machine learning has produced very few learning models that are applicable to the problems faced by situated agents.

Few major behavior-based systems have included learning that is distributed, layered on top of a behavioral substrate, and sufficiently responsive to be considered fully behavior-based. One early example involved the learning of activation conditions for a set of behaviors that were required to coordinate to produce emergent walking behavior on Ghengis, a six-legged robot (Maes & Brooks, 1990). Although this algorithm produced impressive results, the extent to which it can be generalised is unclear.

Another early and important instance of distributed learning in a behavior-based robot is given in Matarić and Brooks (1990), and is of particular relevance to this paper. Matarić and Brooks (1990) detail the development of Toto, a robot that was capable of wandering around an office environment and learning a distributed topological map of it inspired by the role of "place cells" in the rat hippocampus. This map was made up of independent behaviors, each of which became active and attempted to suppress the others when the robot was near the landmark it corresponded to. Each landmark behavior also maintained a list of the other landmark behaviors that had previously followed it, and spread expectation to them when it was active, thereby increasing their sensitivity. Because the behaviors were all active in parallel, the distributed map provided constant time localisation and linear time path planning using spreading expectation, although Toto required the external allocation of its goals because it had no internal drives. The research presented in in Matarić and Brooks (1990) can be considered the first instance of a fully behavior-based learning model and representation. Despite its promise, this line of research was not continued; however, this paper may be considered an extension of it since both the experimental task presented later and its implementation were based on it.

More recently, Matarić (1997) and Balch (1997a, 1997b) have added reinforcement learning modules to behavior-based robots. In both cases, learning was performed using hand-designed state and action spaces, and, in Matarić (1997), with heterogenous rewards functions explicitly designed to make learning feasible, at the cost of significant designer effort and flexibility. In both architectures, a reinforcement learning module was used as a coordinator, and simply inserted, rather than integrated, into the system. Further work by Balch (1999) has focused on the effects of reinforcement function choice on behavioral diversity in multi-robot teams.

2.2 Reinforcement Learning

Reinforcement learning aims to solve the problem of learning to maximise a numerical reward signal over time in a given environment (Sutton & Barto, 1998). The reward signal is the only feedback obtained from the environment, and thus reinforcement learning falls somewhere between unsupervised learning (where no signal is given at all) and supervised learning (where a signal indicating the correct action is given) (Mitchell, 1997).

More specifically, given a set of (Markov) states S and a set of actions A, reinforcement learning involves either learning the values of each $s \in S$ (the state value prediction problem) or the value of each state-action pair (s, a), where $s \in S$ and $a \in A$ (the control problem) (Sutton & Barto, 1998). For most tasks, these values can only be estimated given experience of the reward received at each state or from each state-action pair through interaction with the environment. This estimate is usually achieved by building a table that contains an element for each desired value and using a reinforcement learning method to estimate the value of each element. A comprehensive introduction to the field is given in Sutton and Barto (1998).

Reinforcement learning is attractive to researchers in robotics because it provides a principled way to build agents whose actions are guided by a set of internal drives. It has a sound theoretical basis, can allow for the principled integration of a priori knowledge, handles stochastic environments and rewards that take multiple steps to obtain, and is intuitively appealing. Because it has so many attractive properties, several researchers have added reinforcement learning capabilities to their robots. An early example of this was the development of Obelix (Mahadevan & Connell, 1992), a robot that learned to push boxes by reinforcement. Although Obelix was able to learn in real time, it required a hand-discretised state space and the use of statistical clustering in order to do so, even though the robot's sensor space was only eighteen bits.

The straightforward application of reinforcement learning to robot applications invariably leads to similar problems. Since such models typically use the robot's sensor space as the reinforcement learning state space, they suffer from serious performance and scaling problems – a robot with just sixteen bits of sensor space has over sixty five thousand states. Convergence in such a large state space will take a reinforcement learning algorithm a very long time. One solution to this problem is the use of simulators, in which very long training times are acceptable (e.g., Toombs, Phillips, and Smith (1998)); however, agents that take unrealistic amounts of time to learn a task cannot be considered situated[1]. These problems have led some researchers to develop hierarchical reinforcement learning methods that aim to make learning more tractable through the use of varying levels of detail (e.g., Digney (1998)), and others to use statistical methods to speed up learning (e.g., Smart and Kaelbling (2000)). Another approach is the use of a function approximation method to approximate the value table, although this introduces its own issues (Sutton & Barto, 1998).

Evolutionary approaches to learning by reinforcement have also been proposed (Moriarty, Schultz, & Grefenstette, 1999), in which the agent's control policy is directly evolved to fit a particular task. Although these methods can be competitive with standard reinforcement learning methods, they introduce their own set of difficult design decisions (Moriarty et al., 1999). Furthermore, they typically require several generations of training time, either on a real robot, which makes them unsuitable for the types of problems that we are interested in, or in a simulator onboard a situated agent, which becomes computationally expensive.

The fundamental problem with using reinforcement learning methods in mobile robots is that they were not developed with the problems faced by situated agents in mind. Matarić (1994) gives an important criticism of the direct application of reinforcement learning to behavior-based robotics which reflects the idea that the implicit assumptions made in the reinforcement learning literature need to be reexamined in the context of situated agents.

2.3 Layered Learning

Layered learning was introduced by Stone and Veloso (2000) to deal with problems where learning a direct mapping from input to output is not feasible, and where a hierarchical task decomposition is given. The method involves using machine learning at several layers in an agent's control system, with each layer's learning directly affecting that of subsequent layers through the provision of its training examples[2] or the construction of its input or output features (Stone & Veloso, 2000).

The layered learning method has generated impressive results – for example, simulated soccer playing robots developed using it have twice won RoboCup, the robotic soccer championship (Stone & Veloso, 2000). However, despite its obvious promise, layered learning has not yet been applied to a fully situated agent. Most implementations have been in simulation (Stone & Veloso, 2000; Whiteson & Stone, 2003), where training times can be much longer than those that would be acceptable for real robots. Furthermore, the original stipulation that one layer should finish learning before another can start (Stone & Veloso, 2000) is not realistic in situated environments where change is constant (although recent research has involved concurrent learning (Whiteson & Stone, 2003)).

One relevant application of the layered learning approach is the use of Kohonen networks to discretise continuous input and output spaces in order to make them suitable for reinforcement learning algorithms, in Smith (2002). Although the results thus obtained are promising, the algorithm is hampered by the requirement that the Kohonen map's parameters must be determined experimentally, and by the network's fixed dimensionality. The latter problem could potentially be solved through the use of more dynamic self-organising network types (e.g., Grow When Required (Marsland, Shapiro, &

[1]Note that a simulated agent may be considered situated provided the simulation is sufficiently detailed and the action-environment-perception loop is closed, provided such a robot must be able to learn to perform its task within a realistic time-frame – this includes hybrid methods (e.g., anytime learning (Grefenstette & Ramsey, 1992)).

[2]Although the use of training examples is not appropriate in situated learning, one learning algorithm in a situated agent could for example bias the kinds of learning opportunities another receives.

Nehmzow, 2002)), but the former problem implies that learning is only feasible when it is task-specific. Since Kohonen networks are topological maps, the model presented in Smith (2002) is in some respects similar to the one presented here; however, it was not intended for use in situated agents and does not take the important interaction between the state and action spaces into account, and thus simply uses a separate map for each.

3 Behavior-Based Reinforcement Learning

In this section, we develop a model of reinforcement learning in situated agents motivated by the behavior-based emphasis on distributed control and layered competencies. The next section considers the requirements of reinforcement learning in situated agents, and how these create a different set of concerns from those emphasised in the reinforcement learning literature. Following that, we develop the architecture by first describing the concept of layering reinforcement learning over a topological map, then considering how learning can be made distributed, and finally introducing the idea of asynchronous reinforcement learning. We then provide some examples of cases in which learning could be useful, and summarise.

3.1 Reinforcement Learning in Situated Agents

Although Reinforcement Learning has a strong theoretical basis, it was not developed with the problems facing situated agents in mind – instead, most of reinforcement learning theory assumes abstract state and action spaces and emphasises asymptotic convergence and optimality guarantees. Situated reinforcement learning leads to a different set of issues.

1. **Situated agents are living a life** (Agre & Chapman, 1990). A situated agent has more than one task, and more than one concern. For example, a soda-can collecting robot must also avoid obstacles, navigate, and recharge its batteries when necessary. A reinforcement learning system will make up only one part of the robot's control system, and may have to share control with other reinforcement learning systems. One implication of this is that a situated agent will likely have many sensors and many motor behaviors, not all of which will be relevant to the task at hand. Another implication is that since the robot may be switching between multiple policies (some of which may be reinforcement learning policies with different goals), a reinforcement learning component should be able to learn from transitions executed outside of its own policy, which means that on-policy learning methods may not be appropriate. Finally, the presence of multiple reinforcement signals would require some form of action selection policy (e.g., W-learning (Humphrys, 1996)) or a simple switching or override mechanism.
2. **Reinforcement should emanate from internal drives (e.g., hunger), rather than external conditions (e.g., death)** (Brooks, 1991b). These drives could be either directly engineered or evolved, but would be basic to the agent and not modifiable by its reinforcement learning mechanism. Associative learning could be employed to provide more informative feedback in cases when reinforcement is highly delayed (as suggested in Matarić (1994) for effective learning).
3. **Raw sensory and motor states are not good reinforcement learning states and actions.** Using the sensory space and motor space of the robot as the reinforcement learning state and action space has major and immediate disadvantages. The state (and action) spaces are unlikely to be directly task-relevant, and learning suffers from scaling problems because the addition of extra bits results in an immediate combinatorial explosion. Furthermore, raw sensor and motor descriptors are not appropriate for layered control – using them ignores the presence of motor behaviors and sensory affordances that can be specifically engineered or evolved to aid control and perception. *In short, reinforcement learning over the sensory and motor states of a robot is very likely to be at the wrong level of abstraction.*
4. **A situated agent must learn in a reasonable time relative to its lifespan**. Any learning algorithm which requires thousands of trials to produce good behavior is not suited to a real robot that will likely suffer mechanical failure when run continuously for more than a few hours. A situated agent requires learning that results in a sufficiently good solution to achieve its task in real time. It should act optimally given the knowledge it has *and the time it can reasonably dedicate to learning the task*. The use of task-relevant state spaces in topological maps and asynchronous reinforcement learning (both introduced later) aim to make this easier.

5. **Asymptotic exploration is too slow**. Although the use of ϵ-greedy action selection methods (as are typically employed for exploration (Sutton & Barto, 1998)) provide asymptotic coverage of the state space, they are not likely to do so in a reasonable amount of time, and require the occasional completely irrational action from the agent. The use of optimistic initial values, or some form of "exploration drive" that could be built into the agent separately are likely to be more useful. The inclusion of such a drive with a low priority has the added advantage of allowing the robot to explore only when it has "free time". However, in situations with large state spaces, the robot may have to be satisfied with a suboptimal solution.
6. **Transitions do not take a uniform time**. The use of a global γ parameter to model the devaluation of reward over time is not appropriate in a real environment. When performing an update over a transition, an estimate of the time taken by the transition is already available, since the agent has experienced the transition at least once. Further, future states should not lose value simply because in some abstract way they are in the future; rather they should lose value because time and energy must be expended to get to them. This loss should be factored into the internal reward function for each transition. Similarly, the use of a global λ parameter for TD(λ) is not appropriate because λ-based erosion of eligibility traces implicitly assumes that all transitions take the same amount of time.
7. **Rewards are not received atomically with respect to actions and states**. In some situations, an agent may receive a reward while moving from one state to another, and in others it may receive a reward sometime during its presence at a particular state. The characteristics of the task must be taken into careful consideration when deciding on a reinforcement model.
8. **Transitions take a long time relative to updates**. In the case of a situated agent, the time taken to complete a transition and the time spent at each state are likely to be very much longer than the time required to execute a single update equation. Furthermore, since we later show that reinforcement learning can be performed in a distributed fashion with one process per state node, in principle all of the nodes can perform an update in parallel in the time it would take a single update to occur in a serial implementation. This implies that many updates may take place between transitions.
9. **Other learning methods may be required in conjunction with reinforcement learning**. Situated learning is required to provide useful results quickly, and in some circumstances reinforcement learning by itself may perform poorly. Fortunately, the reinforcement learning complex provides an underlying representation that is well suited to the inclusion of other learning algorithms through the modification of the reward function, the seeding of the initial state or state-action values, or the selection of motor behaviors and sensory inputs.

Although the points listed above range from design suggestions to fundamental underlying assumptions, they represent a different set of concerns than those emphasised by the reinforcement learning literature. One of the reasons that it has so far proved difficult to use reinforcement learning in situated agents has been the lack of recognition of the fact that its methods cannot be simply applied to the situated case; they must be translated for it.

3.2 Reinforcement Learning over Topological Maps

The use of a robot's sensor space directly as the reinforcement learning state space results in a very large, redundant state space where states only have the Markov property for reactive tasks. The size of the state space means that it is difficult to achieve good coverage of the state space and convergence of the state or state-action value table in a reasonable amount of time, often forcing the use of function approximation or generalisation techniques. Because of this, there are very few known examples of behavior-based robots developing useful skills in real time using reinforcement learning.

The model proposed here makes use of an intermediate layer that learns a topological map of the sensor space; reinforcement learning takes place over this map. We define topological map as a graph with a set of nodes N and a set of edges E such that each $n \in N$ represents a distinct state in the problem space and an edge $e = < n_i, n_j >$ indicates that state n_i is topologically adjacent to state n_j *with respect to the behavioral capabilities of the agent*. This means that the activation of some simple behavioral sequence will (perhaps with some probability) move the problem from state n_i to state n_j. Such a map can be built in a distributed fashion through the allocation for each node in the map of a process (or behavior) capable of recognising when it should be active and forming links to subsequently active nodes (e.g., Matarić and Brooks (1990)).

The use of a topological map as the state space for a reinforcement learning algorithm has three major advantages over using the robot's sensor space directly. First, it discards irrelevant sensor input and results in a much smaller and task-relevant state space. This state space will scale well with the addition of new sensory capabilities to the robot because it is task dependent rather than sensor dependent – new sensors will increase the robot's ability to distinguish between states, or perhaps present a slightly richer set of states, but will not introduce an immediate combinatorial explosion. Reinforcement learning over a topological map is therefore much more likely to be tractable than reinforcement learning over a large state space.

Second, the map's connectivity allows for a smaller action space, where actions are movements between nodes in the map rather than raw motor commands. Since such actions will naturally correspond to behaviors in a behavior-based robot, the reinforcement learning layer can be added on top of an existing behavior-based system without greatly disturbing the existing architecture and without requiring exclusive control of the robot's effectors.

Finally, the states in the topological space are much more likely to be Markov states than raw (or even pre-processed) sensor snapshots. This extends the range of reinforcement learning methods for behavior based robotics to tasks that are not strictly reactive, and removes the need for generalisation, because similar but distinct states and actions are no longer likely to have similar values.

An important aspect of the proposed architecture is the interaction of an assumed behavioral substrate, the topological map, and the reinforcement learning algorithm. The behavioral substrate should make learning the topological map feasible, and provide the discrete actions which allow for movement between nodes on the topological map. Rather than simply using the topological map as a discretisation, the interaction between the topological map and the behavioral substrate is sufficient for it to be considered a grounded representation. The topological map, in turn, makes the use of reinforcement learning feasible. Finally, the strategy used for exploration at the reinforcement learning level may influence the way that the topological map develops, since learning at the topological map level continues at the same time as learning at the reinforcement learning level.

This emphasis on interaction differentiates the model presented so far from previous attempts to layer reinforcement learning over other learning elements. For example, Smith (2002) introduced a similar architecture, where a Kohonen network (Kohonen, 1989) is used to discretise continuous input and output spaces, and reinforcement learning is performed over the resulting discretisation. However, two separate maps are used for the purposes of discretisation only, and the system does not take the relationship between the state and action space into account. Furthermore, because Smith uses a Kohonen map, the number of nodes in the map does not change, although their position does.

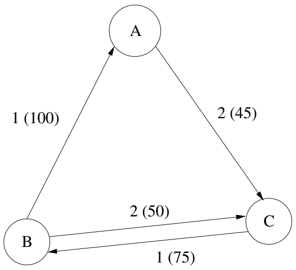
The major implication of the reliance on a topological mapping level is that it requires a tractably maintainable map that provides a good abstraction for the task at hand and can be grounded in the real world. Although there are methods (for example, Grow When Required (Marsland et al., 2002)) that can automatically create and update topological maps for a given state space with no other knowledge, these methods are only likely to be of use when nothing is known about the sensor space at all. In real robot systems, a priori knowledge about the relative importance of different sensor inputs, the relationships between different sensors, and the types of sensor states that are important for the task at hand are all likely to be crucial for the development of a topological map learning layer. In such cases the development of that layer may be a harder problem than the application of the reinforcement learning model developed here on top of it.

3.3 Distributed Reinforcement Learning

Reinforcement learning is typically implemented using a single control process that updates a single state or state-action value table. However, because a topological map is a dynamic structure, and because behavior-based principles require distributed representation and parallel computation where possible, a distributed structure updated by many processes in parallel would be preferable. This section describes how the reinforcement learning update equations can be adapted to run in a distributed fashion over a distributed map.

When performing reinforcement learning over a topological map (with nodes representing states and edges representing actions), we can view the learning as taking place over the nodes and edges of the map rather than over a table with a row for each node and a column for each action type. Figure 1 illustrates the graphical and tabular representations for a simple example action-value set with states A, B and C, and action types 1 and 2, with the action-values given in brackets in the graph.

In a distributed topological map, each node would have its own process which would be respon-



	1	2
A	–	45
B	100	50
C	75	–

Figure 1: Graphical and tabular action value representations.

sible for detecting when the node it corresponds to should be active, and when a transition from it to another node has occurred. This allows each node in the map to maintain its own list of transitions.

In order to add reinforcement learning to the topological map, each process must be augmented with code to perform an update over either the state (node) or state-action (node-edge) spaces, using only information that can be obtained from the current node, one of the nodes directly connected to it, and a reward signal which must be globally available. When reinforcement is performed over a distributed topological map, we use the term *reinforcement complex* rather than reinforcement value table to refer to the resulting distributed structure.

Since reinforcement learning update methods are intrinsically local, they require very little modification in order to be implemented in a distributed fashion. We consider only Temporal Difference methods here. More detail on how to implement Monte Carlo methods and TD(λ) in a distributed fashion are given in Konidaris (2003).

Temporal difference methods are the easiest methods to implement in a distributed fashion because temporal update equations involve only local terms. The standard one-step temporal difference update equation (known as TD(0)) is:

$$V(s_t) := V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) \quad (1)$$

where α and γ are global constants, $V(s_t)$ is the value of the active state at time t, and r_t is the reward received at time t (Sutton & Barto, 1998). The equation represents the idea that the value of a state ($V(s_t)$) should move (with step size α) toward the reward obtained by being there (r_{t+1}) plus the discounted value of the state encountered next ($V(s_{t+1})$). In order to implement this update equation, each node's process has only to note its own value (this gives us $V(s_t)$), the value of the node that is active immediately after it (this gives us $V(s_{t+1})$), and the reward received during the transition (this gives us r_{t+1}). It should also record the behaviors activated to cause the transition, and establish a link between the two nodes if one is not already present.

The update equation used for state-action value updates (known as Sarsa) is a slightly more difficult case. The Sarsa equation is:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \quad (2)$$

where $Q(s_t, a_t)$ is the value of taking action a_t from state s_t. This requires a node to have access to the value of the state-action pair following it, as well as the reward obtained between activations. One way to reduce the information sharing required would be to perform the update in terms of a state value, since state values can be computed using state-action values. The update equation would then be:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - Q(s_t, a_t)) \quad (3)$$

where $V(s_{t+1})$ can either be the expected value of s_{t+1} calculated probabilistically, or simply the expected value of the action with the highest value from that state. The latter case is equivalent to Q-learning since then $V(s_{t+1}) = \max_{a_t} Q(s_{t+1}, a_{t+1})$.

3.4 Asynchronous Reinforcement Learning

Since the time required by a situated agent to move in the real world is very much longer than that required to perform an update, rather than performing updates once per transition a situated agent should be performing them all the time, over all nodes in parallel. In order to do this, the reliance of the update equations on the concept of "the transition just experienced" must be removed. Therefore, it makes sense to use all of the experiences the agent has obtained so far to provide state and state-action value estimates, instead of simply the reward and state values it last experienced. Experienced values are thus used to create a model from which state or state-action value estimates are taken. For example, each node could update its state-action values using the following equation:

$$Q_{t+1}(s,a) := Q_t(s,a) + \alpha(r_{s,a} + \gamma E_{s,a}\{V(s_{t+1})\} - Q_t(s,a)) \tag{4}$$

where $r_{s,a}$ could be estimated as the average of all rewards received after executing a at s, and $E_{s,a}\{V(s_{t+1})\}$ would be the expected state value obtained after the execution of action a from state s. The expected state value could be the weighted (by observed probability) average of the states visited immediately after s, with each state value taken as the value of the maximum action available from that state. The γ parameter could be set to 1, or to a transition-specific decay value. This is just a rewrite of equation 3 using expected reward and expected state value rather than sample reward and sample state value, but it allows the update equation to be executed in parallel, *all the time*. Although this method requires some extra computation because two weighted sums must be computed for each update, neither computation would be difficult to build in hardware or using artificial neurons.

This draws from three ideas in the Reinforcement Learning literature. Like Asynchronous Dynamic Programming (Bertsekas & Tsitsiklis, 1989), equation 4 uses a model to perform what is termed a *full backup*, which uses the expected value of a state or state-action pair rather than a sample value, and it does so in no pre-defined order. However, unlike Asynchronous Dynamic Programming, the model used is derived from experience with the environment, and is not given a priori. Nevertheless, since equation 4 is just an Asynchronous Dynamic Programming update with a learned rather than given model, it inherits the same convergence conditions and guarantees given by Bertsekas and Tsitsiklis (1989), except that it will converge to an optimal policy with respect to *the learned problem model* rather than one given a priori.

Equation 4 is also similar to *batch-updating*, where the update rules from a given transition experience are repeatedly applied, but differs in that it does not simply repeat previous episodes, but uses a model of the environment to generate value estimates, and performs backups over all the nodes in the distributed map. Finally, it is similar to the Dyna architecture proposed by Sutton (1990) in that a model of the environment is built and used for reinforcement, but differs in that the updates occur in parallel and all the time, use the model to generate expected rather than sample state and state-action values (although Dyna could easily be adapted to allow this) and does so for all state-action pairs in each state rather than a single one.

A useful consequence of performing asynchronous updates over a topological map is that several reinforcement learning complexes learning in the same (or a similar) state space can learn from the experiences generated by each other. This makes it easier to add new drives (along with their associated reinforcement complexes), or even new learners with different parameters, to the system provided some action-selection mechanism (which could be fixed or adaptive) is present.

Ideally, the use of asynchronous updates leads to the convergence of the values in the reinforcement learning complex *between transitions*. Since the complex converges to an optimal policy with respect to the learned model, the agent must then be making the best possible choices at each transition, given its drives and the information that it has obtained from the environment.

3.5 Example Application Scenarios

One situation where the reinforcement learning architecture proposed here would be useful is the case of a rat learning to find a piece of food in a maze. The nodes in the topological map would correspond to landmarks in the maze, with a connection between two of them indicating that the rat is able to move directly from the first to the second, with reinforcement based on the rat's hunger drive. The experiment presented later in this paper is based on this example. Here, a potential application of an additional learning method could be the use of associative learning to modify the reinforcement function so that locations where the smell of cheese is present receive some fraction of the reward received for finding the food.

Another application could be the use of reinforcement learning in the development of simple motor skills for a robot with many actuators. For example, given a set of motor behaviors and joint angle sensors, a robot with a mechanised arm could use the reinforcement learning model proposed here to learn to reach out and touch an object. In this case the joint angle sensors in conjunction with the motor behaviors would provide the basis for the topological map, where nodes would be significant joint angle configurations and edges between them would indicate that movement between two configurations is possible with some short sequence of motor behaviors. In this case a self-organising map (such as Grow When Required (Marsland et al., 2002) with appropriate input space scaling factors) could be used to create the topological map. The robot would have an interval drive that rewards it for touching the object, and the use of visual feedback could provide a heuristic that could modify the reinforcement function. Although this task seems easy given visual feedback it might be possible for the robot to learn to do it quickly and with little visual feedback or error with the aid of reinforcement learning.

Reinforcement learning could also be used for more complex motor coordination tasks, such as changing gear in a car with a manual transmission. This requires a fairly difficult sequence of actions, using a leg to engage the clutch and an arm to change gear. The map here would again be based on the joint angle sensors for the arm and leg and a set of motor behaviors. Here, the use of social imitation could serve to seed the initial state values in order to make the task tractable – this is a fairly difficult learning task that takes humans a fair amount of time, effort and instruction to learn to perform smoothly.

In all three examples, the selection of the relevant sensors and motor behaviors is crucial. For example, it would be very difficult for a robot to learn to touch an object with its arm when all of the internal joint sensors in its entire body were considered as input to a topological map, even though some of them might be relevant. For example, although it would be difficult to touch the ball while facing away from it, the addition of a behavior that orients the robot to a standard position relative to the ball before attempting to touch the ball would probably be a better choice than including extra sensors in the reinforcement state space. The integration of other learning methods may aid in the selection of the relevant sensors and motor behaviors, and may also be useful in speeding up learning, or making it feasible in the first place.

3.6 Summary

This section has presented a model of reinforcement learning for autonomous agents motivated by the behavior-based emphasis on layered competencies and distributed control. The model is intended to produce behavioral benefits in real time when used in a real robot. It is novel for three reasons. First, it performs reinforcement learning over a learned topological map, rather than directly over the robot's sensor space. This aims to make learning feasible through the use of a small, relevant space tailored for the task at hand. Second, reinforcement learning is performed in a distributed fashion, resulting in a reinforcement learning complex embedded in a distributed topological map rather than a single state or state-action value table updated by a single control process, allowing for a dynamic structure that could potentially be updated in parallel with the use of parallel hardware. Finally, in order to take advantage of this parallelism, and the fact that situated agents will take much longer to make a transition than to perform an update, learning is asynchronous, and takes place all the time. Experiences are used to update an internal distributed model of the environment which is used as a basis for reinforcement learning, rather than being used in the reinforcement learning directly.

In order to implement the architecture, an agent would first need be able to maintain the distributed topological map, and then would need to be able to obtain and update reward and value estimates. Map building would usually be achieved via a behavior that creates a new node behavior when none are active but one should be. Each node would be capable of tracking the nodes active immediately after it, and the behavioral sequence required to get there. Reinforcement learning could then be implemented by adding code to each node behavior to keep track of rewards received during transitions from it, and to update its reward and value estimates with them, with each node behavior running its update equations continuously.

4 The Experiment: Puck Foraging in an Artificial Arena

In this section, we present an experimental task designed to test the model presented in this paper. The experiment aims to augment the distributed map building system developed by Matarić and Brooks (1990) with the new reinforcement learning model and show that this can produce complex, goal-directed and path-planning behavior in an agent that performs puck foraging in an artificial arena.

The following section describes the experimental task and the control strategies used in it, and is followed by a brief outline of the evaluation criteria used. We then introduce the three test arena configurations used in the experiment, and outline the aspects of the system that each was designed to test.

4.1 Overview

The experiment outlined here is intended as an abstraction of the rat in a maze example given in the previous section, which is itself an abstraction of the kinds of tasks commonly faced by foraging animals. It models an agent living in a static environment with obstacles that it must avoid, but that it can use as landmarks for the purposes of navigation. The agent is driven by three internal needs – the need to find food, the need to explore its environment, and the need to return home. These needs are in turn activated and deactivated by a circadian cycle.

A mobile robot is placed in an artificial arena containing orthogonal walls (henceforth referred to as "vertical" and "horizontal" walls, since this is how they appear in figures) and one or more food pucks. The robot must start with no prior knowledge about the layout of the arena, and must navigate it for ten cycles. Each cycle is made up of three phases:

1. **Foraging**, where the robot should attempt to find a food puck (there may be more than one) in as short a time as possible. As soon as the robot has found a food puck, it switches to the exploration phase of the cycle. If it cannot find a food puck within three minutes, it must skip the exploration phase and move directly to the homing phase.
2. **Exploration**, where the robot should explore areas of the arena that are relatively unexplored for the remainder of the first three minutes. After that, the robot switches to the homing phase. The exploration phase is intended to allow the robot to build up a more accurate and complete map of its environment, if it has time after finding food.
3. **Homing**, where the robot must return to the area where it was first started. This is intended as analogous to nightfall, where the agent must return to its home to sleep. The robot moves to the next cycle and begins foraging again.

During its run, the robot is required to follow walls, and decide which action to take at each one. The robot accomplishes this by building and updating a distributed topological map of the arena and performing reinforcement learning over it. Figure 2 depicts an example scenario where a maze configuration (on the left) is split into the individually recognisable walls or landmarks (in the middle) and a state node is allocated to each, with arrows indicating that the agent is able to move from one to another (on the right).

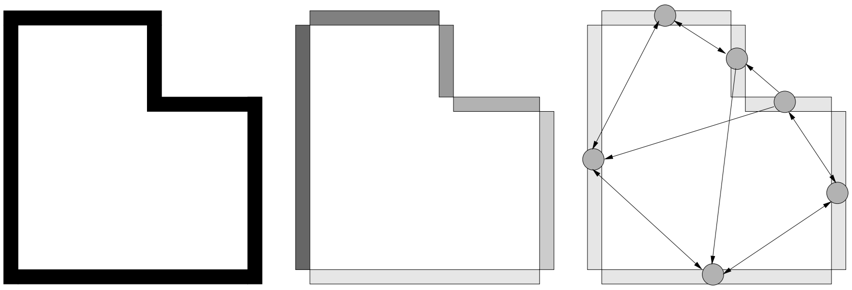


Figure 2: An example arena configuration (left), split into distinct walls (center) and represented as a topological map (right), with a node for each landmark and arrows indicating potential movement.

At each wall, the robot is restricted to one of three types of actions – turn right, left, or go straight – at either end of the wall, giving six actions in total. The robot therefore has to follow the wall until it reaches the desired end, and execute the desired action. However, the robot may not turn away from the wall, so only four of the six potential actions are available for walls. When the robot is in a corridor, it may choose from all six. Figure 3 shows the available actions for horizontal walls and corridors (the vertical case is similar).

Not all actions are possible in all cases – for example, if the left side of a corridor continues on to become a left wall while the right side does not, the robot may not turn left at that end; a

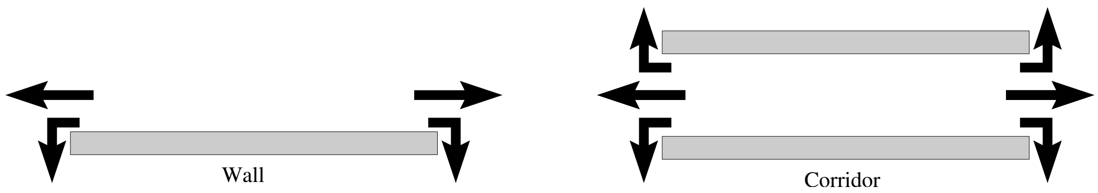


Figure 3: Potential actions for a wall following robot at a single wall (left) and a corridor (right).

similar problem occurs with corners. Therefore, the robot must determine whether or not an action is possible, and avoid attempting illegal actions.

The state space for the reinforcement learning function was therefore the set of landmarks present in the distributed map, and the action space was the set of legal actions at each landmark. A state transition thus consisted of a landmark, a turn taken there, and the first landmark detected after the execution of the turn, with the transition being considered completed once the second landmark had been detected.

In order to implement the robot's internal drives, and to provide a useful metric for comparing various models, each drive is given an internal reward function. The following equations were used for the foraging, exploration, and homing rewards respectively:

$$f(x) = \begin{cases} 200 & \text{when a food puck is in view} \\ -1 & \text{otherwise} \end{cases}$$

$$e(x) = \begin{cases} 200 - 200\frac{n_t}{n_{ave}} & n_{ave} > 0 \\ 200 & \text{otherwise} \end{cases}$$

$$h(x) = \begin{cases} 200 & \text{when the robot is ``home''} \\ -1 & \text{otherwise} \end{cases}$$

where n_t is the number of times that the transition just executed has been taken in total, and n_{ave} is the average number of previous executions over all the transitions in the map. The exploration reward function was executed once per transition, while the other were executed with a constant time delay set so that the robot would receive a penalty of at or near 200 for failing to find the puck before the end of a cycle.

For simplicity, at each choice point the robot aimed to maximize the sum over time of the value of the reinforcement function corresponding to the current phase, rather than attempting to maximize some combination of all three. Three decision models were then used for the experiment:

1. **Random Movement**, where the robot chose an action at random from the set of currently legal actions. This agent built an internal distributed map of the arena but used it only to determine which actions were legal. This model corresponds to the strategy typically employed in behaviour-based systems.
2. **Layered Standard Reinforcement Learning**, where the robot built an internal distributed map of the arena and used Q-learning (Watkins & Dayan, 1992), a standard single-step temporal difference learning algorithm, over it. In this case, each time a transition was executed, exactly one update was made to each drive's reinforcement complex. This represents the application of traditional reinforcement learning techniques on top of a topological map.
3. **Asynchronous Reinforcement Learning**, where the robot built an internal distributed map of the arena that used the full model developed in this paper (which we will label ATD, for asynchronous temporal difference learning) over it, and constitutes the first implementation of a fully behaviour-based reinforcement learning robot. In this model, all three reinforcement complexes were updated continuously, although only the values for the currently active drive were used to decide what to do next.

Note that in both reinforcement learning cases, reinforcement learning and map learning occur at the same time, and that the robot often has to make decisions with an incomplete map.

4.2 Evaluation

When evaluating each model quantitatively, the reward values of each internal drive over time (averaged over a number of runs) were directly compared, thereby using the reward functions as performance metrics. However, since the exploration phase was of varying length and did not occur in every cycle, the results obtained for it for each model could not be directly compared. The average change of state value function in the map over time was also recorded, along with the transitions that added reward to the map, in order to examine the convergence properties of the asynchronous system.

In order to evaluate each model qualitatively, a visualisation of the distributed map learned by the robot and the action values for it was studied. Recordings were also made of the robot's movements so that specific instances could be replayed and examined, and internal data from the state of the robot was used to obtain further information about the reasons behind the choices made.

4.3 The Arenas

Diagrams of the three arena configurations used in the experiment are given in Figure 4. Light shading indicates the regions used as the robot's home area for each arena, and the black circles represent pucks. The robot was always started in the bottom left corner of each arena, facing right.

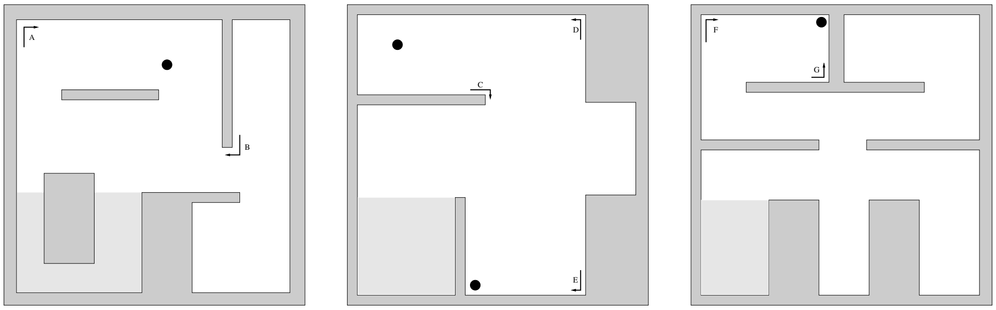


Figure 4: Overhead diagrams of the three experimental arenas, with the robot's home area shaded in the bottom left of each.

The first arena was used as a testing platform during development, and as a simple trial problem instance designed to verify that each system worked. It was deliberately friendly towards the reinforcement learning agents. The transition labelled A was the only transition in the first arena leading to a puck reward, and was only a few transitions away from the robot's home area. In addition, the area on the right functioned as a kind of trap, which, once entered, could only be escaped through the turn marked B. Thus, both reinforcement learning systems were expected to be able to learn to find the puck and to return home relatively quickly, while the random model was expected to have mixed results finding the puck and difficulty returning home.

The second arena configuration was intended to be hostile to reinforcement learning agents and relatively friendly toward random agents. Because turns out of the home area were never likely to form perfectly straight lines, the robot might then encounter any one of the walls in the right central configuration. In addition, one of the pucks was taken away at the end of the fifth cycle. For the reinforcement learning agents, this was to be the last puck found during a foraging phase (either if none had been seen so far) and either for the random agent. Finally, the second arena was designed so that the random agent was fairly likely to eventually stumble across either the puck (using transitions C, D or E) and the way home. This combination of noisy transitions and a modified environment was intended to test how the reinforcement learning models could perform (and recover from change) in a difficult environment where a random agent could do well.

The third arena was designed to test the ability of the reinforcement learning robots to learn a long path to the puck and back again in the most complex task environment the robot was required to face. The robot had to make five consecutive correct decisions to go directly to the puck (with transitions F and G as scoring transitions) from its home area, and find a potentially even longer path home. The long paths were intended to highlight the difference between the synchronous and asynchronous reinforcement learning systems, where the agent using the asynchronous model was expected to be able to learn to find the path almost immediately after finding the puck for the first time, whereas the agent using the synchronous model was expected to take longer.

5 Implementation

In this section we briefly outline the implementation of the experimental task. A more detailed description can be found in Konidaris (2003).

5.1 The Environment

Each arena was built on a 90cm^2 wooden base, with the walls constructed using pieces of styrofoam and insulation tape, and covered with sheets of white cardboard secured with drawing pins. The same type of cardboard was used to round off sharp internal corners. The use of the cardboard served to provide a smooth response for the infra-red sensors used, and the rounded corners simplified cornering and wall-following behaviour. The other materials were chosen because they were readily available. Three white wooden cylinders were used as food pucks, with a strip of black insulation tape marking them for easy visual detection. The three configurations are shown in Figure 5.

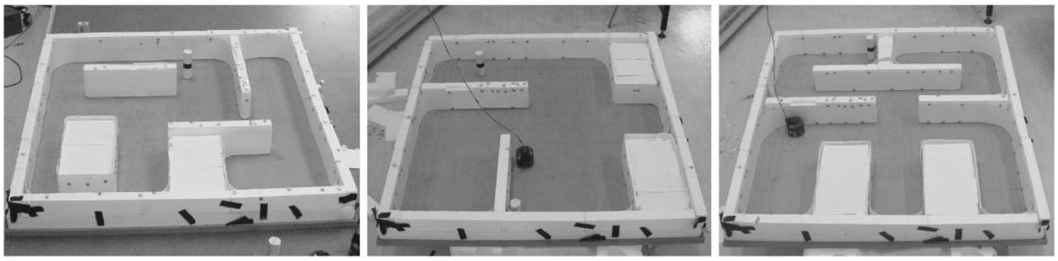


Figure 5: The three artificial arenas.

5.2 The Robot

Dangerous Beans, the robot used to perform the experimental task, was a standard issue K-Team Khepera robot equipped with a pixel array extension turret. The Khepera has a diameter of approximately 55mm, eight infra-red proximity and ambient light sensors, and two wheels with incremental encoders (K-Team SA, 1999b). The pixel array extension provided a single line of 64 grey-level intensity pixels with a viewing angle of 36^o (K-Team SA, 1999a). The Khepera's onboard infra-red sensors were used for obstacle avoidance and wall following, while the pixel array was used for puck detection. Figure 6 shows Dangerous Beans next to an overhead sensory schematic. The infra-red sensors are numbered from 0 to 7, and the angle of view of the pixel array turret is indicated.

Dangerous Beans was controlled through a serial cable suspended from a counterbalanced swivelling tether, connected to a standard Linux machine running a control program written in `C`. Each behavior instance was allocated its own thread, and communication was achieved through the use of global variables.

5.3 Distributed Map Building

This section details the development of Dangerous Beans's control system up to and including its distributed map-building layer. We separate this portion of the system from the reinforcement learning layer because it is essentially a replication of Matarić and Brooks (1990).

The behavioural structure of the control system used for Dangerous Beans is depicted in Figure 7. The behavioural modules with a darker shade are dependent on those with a lighter shade, either because they rely on their output, or because they rely on the behaviour emergent from their execution. The dashed arrows represent corrective relationships (where higher level modules provide corrective information to lower level ones), and behaviours shown in dashed boxes are present in multiple instantiations. Solid arrows represent input-output relationships.

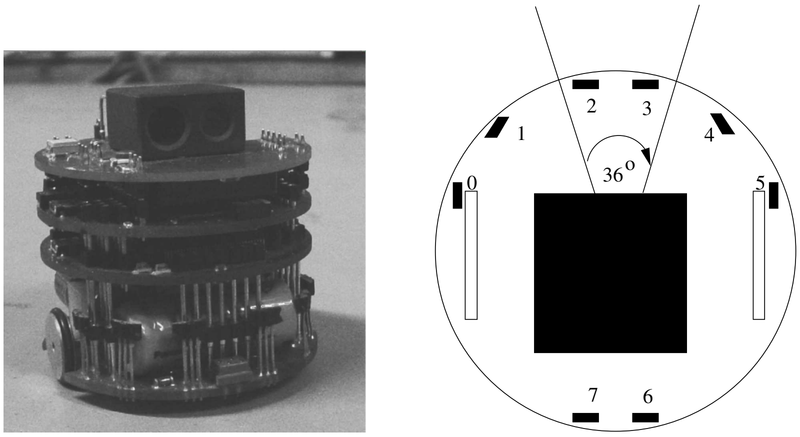


Figure 6: Dangerous Beans: photo and overhead sensor schematic.

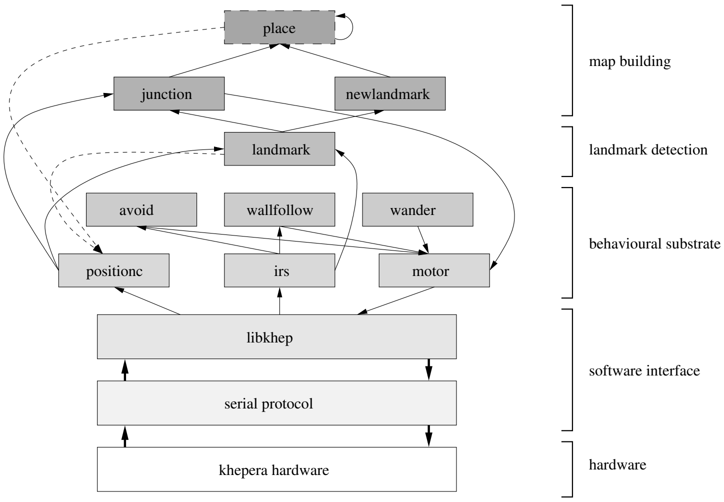


Figure 7: Dangerous Beans: behavioral structure (map building).

The following sections describe the behavioral substrate, landmark detection, and map building layers in turn.

5.3.1 Behavioral Substrate

The behavioral substrate developed for Dangerous Beans was required to produce sufficiently robust wall-following behavior to allow for consistent and reliable landmark detection and map building.

Two behaviours, `irs` and `motor`, handled the interface between other behaviours and the robot's sensors and actuators. In addition, the `positionc` behaviour performed dead-reckoning position estimation based on encoder readings from the khepera's wheels.

The `wander` and `avoid` behaviors performed threshold-based collision-free wandering, using gentle turns away from lateral obstacles to obtain behavior that allow the `wallfollow` behavior to perform wall following by attempting to keep lateral sensor readings at a constant level of activation when they were non-zero. This resulted in wall-following behavior that was as robust as could be expected given the short range of the khepera's sensors.

5.3.2 Landmark Detection

The `landmark` behaviour performed landmark detection, and broadcast the current landmark type, heading, and number of consecutive readings. The landmark type took on values of either right wall, left wall, corridor, or empty space, and the current heading was given as one of 0, $\frac{\pi}{2}$, π or $\frac{3\pi}{2}$

radians. The behaviour used the dead-reckoning angle to estimate the angle of a wall, and then (if the landmark had been detected at least 4 times) supplied a corrected angle back to `positionc` to minimise dead-reckoning angular error in the absence of a compass. The accuracy achieved using the corrected angular estimates was sufficient for landmark discrimination in this case, where walls are known to be horizontal and vertical only.

The behaviour used a simple statistical approach similar to that given in (Matarić & Brooks, 1990). A set of 50 thresholded samples were taken from the left and right lateral sensors, and each sample was thresholded. The landmark was determined to be a corridor if at least 25 samples showed left activation and 25 showed right activation; failing that, it was determined to be either a left or right wall if at least 30 samples of the relevant side were above the threshold. If neither condition was met the landmark type was set to the default value of free space.

A new landmark was detected if the type of landmark changed, or if the estimated angle of the robot differed from that of the currently active landmark by 0.8 radians. The estimated angle of the landmark was selected as the one of the four orthogonal directions that walls are expected to lie along nearest to the current estimated angle.

5.3.3 Map Building

The layer of behaviours responsible for map-building maintained a distributed map by creating new "place" behaviours for novel landmarks and linking places together when they appeared sequentially.

Each landmark was allocated its own `place` behavior, which maintained a landmark descriptor consisting of its type, angle, estimated coordinates, and connectivity information. The descriptor was used by each `place` behaviour to continuously compute the probability that it corresponded to the current landmark, with the `place` behaviour with the highest probability judged to correspond to the current landmark. `Place` behaviours not of the correct type and angle immediately set their probabilities to zero, while those with the correct type and angle were assigned a match probability inversely proportionate to estimated distance, reaching zero at about 20cm from the landmark.

Each `place` behaviour also maintained a linked list of transitions, which stored the `place` behaviours that became active immediately after them, the type of turn (left, right, or straight, with an extra direction modifier to indicate which end of the landmark the turn was from) that resulted in the transition, and how many times that combination had occurred so far.

Although the system described in (Matarić & Brooks, 1990) uses expectation as a deadlock breaker before dead-reckoning, because of the higher branching factors and more complex maps created here, dead reckoning was required fairly frequently and thus expectation was not used to modify the matching probability.

The `newlandmark` behaviour was responsible for detecting when no place behaviour had a sufficiently high probability of corresponding to the current landmark and allocating a new one for it. For simplicity, the `newlandmark` behaviour also determined which place behaviour was the current best, and when to merge landmarks. Landmarks were merged when they were both strongly active at the same time, and overlapped significantly. Duplicate landmarks were artifacts of the fact that Dangerous Beans sometimes encountered a wall half way along its length, and therefore only created a landmark behaviour covering half of it, allowing for a new behaviour to erroneously be created if the wall was later encountered on the unexplored side. This problem does not occur in the model used by (Matarić & Brooks, 1990) because of its more strict wall-following behaviour, but it is a significant problem here. The merging procedure adopted here solved it in all observed cases.

Finally, each `place` behaviour was responsible for correcting the current estimated position of the robot according to the expected position of the landmark. This simple corrective approach proved mostly sufficient for the simplified environment used in the experiments – occasionally the correction mechanisms failed in some cases, and runs where this occurred were restarted. In most cases failures occurred because of inaccurate angle estimates over long empty spaces where the robot could not obtain angular corrective information, and could have been avoided through the addition of a direction sense (e.g., a polarisation compass (Schmolke & Mallot, 2002)), the use of a more sophisticated correction approach (e.g., Choset and Nagatani (2001)) or the use of a method for landmark disambiguation not based on dead reckoning (e.g., neighbourhood characteristics (Dudek, Freedman, & Hadjres, 1993)).

The `junction` behavior monitored the current `landmark` behavior and when the currently detected landmark changed, picked a random turn to perform from the set of legal ones for its type. It also updated a global variable indicating the last turn taken, which was used by `place` behaviors when noting transitions.

Figure 8 is a visualisation of the distributed mapping data produced by Dangerous Beans on the first test arena. Landmarks are represented by rectangles, each with a central circle and two end circles. Corridors have two rectangles with the circles placed between them. Lines represent transitions, where a line from the circle at the end of one landmark to the circle in the middle of another indicates that Dangerous Beans has been able to move from the first landmark to the second. The map contains 17 landmarks and 32 edges, although some edges are not distinguishable here because they have different turn types but are between the same landmarks.

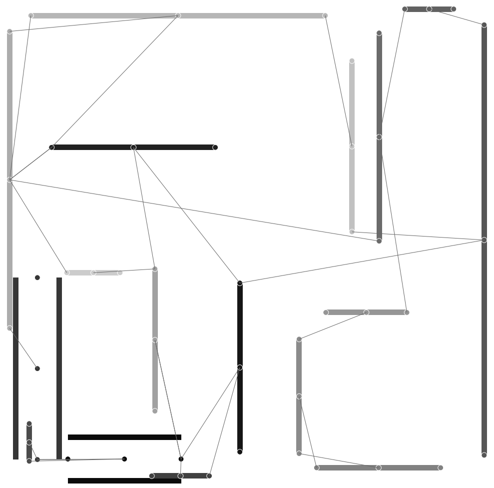


Figure 8: A visualisation of a distributed topological map of the first arena.

The slightly exaggerated length of all of the landmarks is an artifact of the landmark recognition algorithm used. This means that some landmarks may appear to overlap (for example in the bottom left corner) but are actually just close together.

5.4 Distributed Reinforcement Learning

This section describes the additional control structures added to Dangerous Beans to enable it to perform distributed reinforcement learning over its distributed topological map. The behavioural structure used in the experiments was largely the same as that given in Figure 7, with four additional behaviors and one modified behavior. The following sections describe these changes.

5.4.1 Internal Drives

In order to express the three drives required in the experiment, three reward behaviours were added to Dangerous Beans, each exposing a global reward variable that could be read by other behaviors. The equations given in section 4.1 were run roughly once per second.

The `seepuck` behaviour determined when the robot was facing a puck, and should receive a puck reward. A simple averaging thresholding algorithm was used to spot the dark strip of the puck against a light background. Reward was inhibited for 20 seconds after each puck sighting to avoid issuing multiple rewards for the same puck. The `homing` behaviour checked whether or not the robot's estimated position was within some arena-specific range of the robot's original location, so that any location within this boundary was considered home, and required the robot to be at least 10cm outside the area and return before allocating reward again. The `explore` behaviour was given the number of times each transition had already been taken as it was taken again, and using this along with the overall average computed from the set of `place` behaviours, determined a transition reward according to the exploration reward equation.

Finally, the `circadian` behaviour was responsible for keeping track of the current cycle and active phase of the robot, and switching phase when required. It exposed a global variable representing the current phase (and thereby active desire) for other behaviours to use when making decisions.

5.4.2 Making Choices

The `junction` behaviour was extended to allow place behaviours to signal turn requests they wanted carried out by posting their requests to a global variable that was checked every time a decision had to be made. The `place` behaviour was modified so that it only posted once per activation (unless the current drive changed while it was active, in which case it was allowed to post again), and according to whichever control strategy was being used at the time.

The `junction` behaviour executed the requested turn if possible; some turns had to be ignored when they could not be executed because of the presence of an adjoining obstacle. Therefore, each `place` had a counter for each turn, which was incremented when the turn was taken and decremented when `junction` signalled that it could not be. When this counter reached -2 the turn was banned, and not considered for any further decision making or reinforcement learning purposes.

The `junction` behaviour was also responsible for determining when the robot was headed along the wall in the wrong direction given the requested turn, and reversing the robot's direction without losing contact with the wall.

5.4.3 Place and Transition Values

Since the robot was likely to spot a puck shortly after making a transition, and could not guarantee that simply by being at a particular landmark it would see the puck, reward was allocated to transitions rather than places. Three separate action value estimates were kept (one for each drive), so that although all three learned from all transitions, there were three independent reinforcement learning complexes embedded in the topological map.

Each transition received the difference in reward between the time the robot left its "from" landmark to the time it left its "to" landmark. In order to record the reward obtained by each transition, each `place` behaviour kept a record of all of the reward levels as soon as it became inactive. The transition made was then noted, and when the place that it led to became inactive again, the transition received the difference between the initially noted levels and the levels after the deactivation of the subsequent `place` behavior. Each transition kept a total of the reward it had received along with the total number of times it had been taken, and the number of those times where a negative reward was received.

The update equation used for the asynchronous reinforcement learning model (run by each place behaviour at all times for all transitions) was:

$$Q_{t+1}(s,a) := Q_t(s,a) + \alpha(r_{s,a} + E_{s,a}\{V(s_{t+1})\} - Q_t(s,a)) \tag{5}$$

where α was the learning step parameter (set to 0.1[3]), $Q_t(s,a)$ was the value of taking action (turn) a at state (place) s at time t, $r_{s,a}$ was the expected reward received for taking action a at state s and $E_{s,a}\{V(s_{t+1})\}$ was the expected state value after action a at state s, at time t. Each `place` stored the Q values for each of its possible turns, and during the update $E_{s,a}\{V(s_{t+1})\}$ was calculated for each turn by computing the sum of the values (weighted by observed probability) of each state encountered after taking turn a at state s. The expected reward term $r_{s,a}$ was computed for each action as the average reward obtained over all executions of the transitions using turn a from the state. For the exploration reward function, the estimated reward was computed directly from the equations given in section 4.1, since previous exploration rewards for a particular turn were not useful in estimating its current value.

Since the task was effectively episodic, when a transition had a positive reward its contribution to the expected value of its "from" state did not include the expected value of its "to" state. This has the same effect as considering positive rewards to end the episode, and prevented positive feedback loops where states could have obtained infinite expected values.

In the synchronous update case, the value function for each state-action pair was only updated immediately after a transition from the state using the action was completed, and instead of an average reward, the reward obtained was used directly. The update equation used for the synchronous case was:

$$Q_{t+1}(s,a) := Q_t(s,a) + \alpha(r_{t+1} + V(s_{t+1}) - Q_t(s,a)) \tag{6}$$

[3]This is the most common value used in Sutton and Barto (1998). Due to time constraints, no systematic evaluation of its effect was performed.

where now r_t was the expected reward received at time t, and $V(s_t)$ was the value of the state active at time t. Since the value of each state was taken as the expected value of the maximum action that can be taken there, the synchronous case is equivalent to Q-learning (Watkins & Dayan, 1992).

In order to encourage exploration, actions that had not yet been taken from a given state were assigned initial values of 50 for both homing and puck rewards. Initial exploration rewards were set to 200. All initial reward estimates were immediately replaced by the received reward for the asynchronous model.

For the reinforcement learning robots, when a `place` behaviour became active, it would post a turn request to the `junction` behaviour using the action with the highest action value, with ties broken randomly. When all of the action values available were negative, or when the requested action could not be taken, a random action was chosen. In all cases, only legal turns (those allowed by landmark type and so far not found to be impossible) were considered.

6 Results

The critical test for a learning method that claims to be able to improve the performance of an existing robot system is whether or not it can perform as required in the real world, in real time. In this section we present the results of the experiment presented in Section 4.1, which show that the model developed in this paper is able to learn a path to the puck and back to its home area rapidly, outperforming both alternative models in all cases. We further demonstrate that since the asynchronous model's reinforcement learning complex converges between decisions, Dangerous Beans achieves goal-directed behavior that is at all times as good as can be achieved given its drives and the knowledge it has. The following sections present and analyse the results obtained for each arena individually, consider the issue of convergence, and then draw conclusions from all of the data presented.

6.1 The First Arena

In the first arena, both reinforcement learning systems were able to learn to find direct routes to the single puck and back to the home area quickly and consistently. Figure 9 shows the puck (9a) and home (9b) rewards obtained over time, averaged over seven runs, for each of the system types, with the error bars indicating standard error.

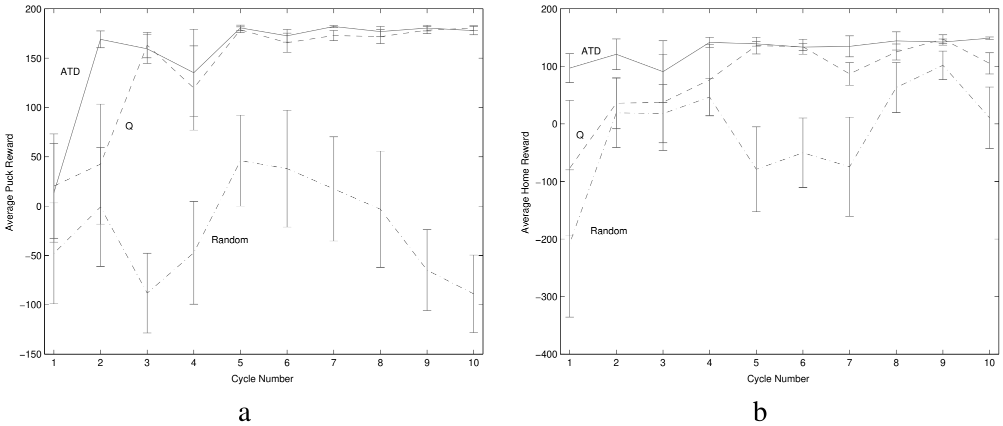
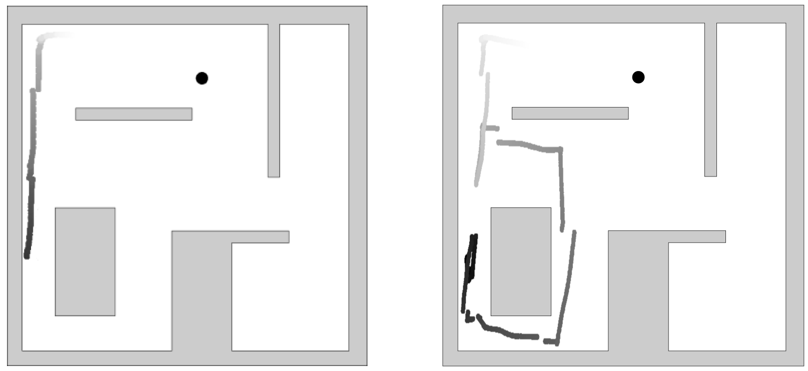


Figure 9: Average puck (a) and home (b) reward over time for the first arena.

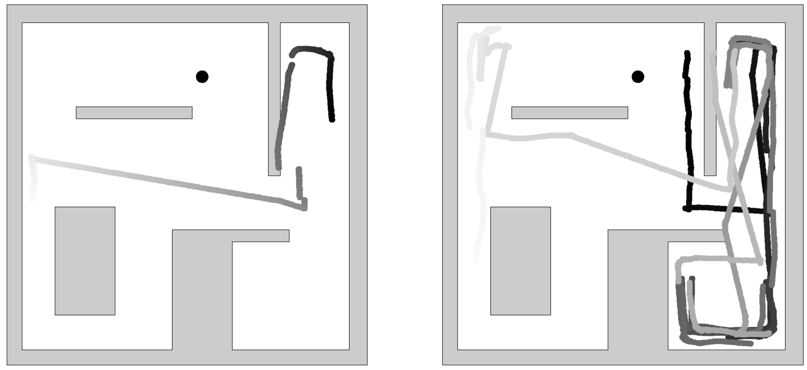
As expected, both reinforcement learning models learned good solutions quickly, progressing in both cases from near-random results with a wide spread of reward values (indicated by the large error bars) in the first cycle to nearly uniformly good results (indicated by the very small error bars) by the fifth cycle. The asynchronous model even appears to have been able to return to the homing area quickly at the end of the first cycle, which was likely the result of an active exploration strategy and rapid learning. In contrast, the random control strategy performed poorly, resulting in a low average reward with large error bars throughout, as expected given the trap on the right of the arena.



a b

Figure 10: Sample learned (a) and random (b) routes to the puck in the first arena. Note that the robot's trail fades over time.

The left part of Figure 10 shows the route learned in nearly all cases by both reinforcement learning models to the puck[4]. Note that the breaks in the path were caused by landmark-based correction, and that the robot receives the puck reward as soon as it can see the puck. On the right is a sample path taken by the random algorithm to find the puck. As expected, the random algorithm does not move directly towards the puck and instead finds it by chance. This path is nevertheless quite short because when the random agent wandered into the trap on the right or doubled back on itself repeatedly it virtually never encountered the puck before the end of the cycle.



a b

Figure 11: Sample learned (a) and random (b) routes home in the first arena.

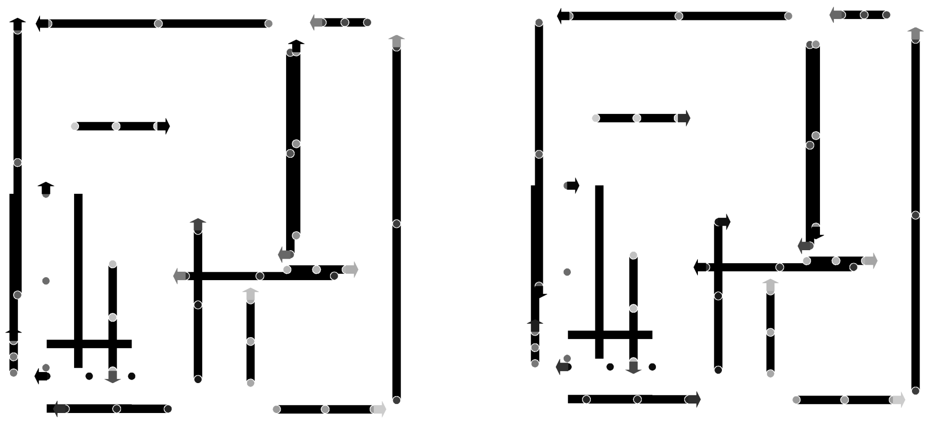
Figure 11 shows typical routes home for the reinforcement models (on the left) and the random agent (on the right). Note that the random agent gets stuck in the trap on the right for some time, eventually wandering home, whereas the reinforcement learning agents escape immediately.

Figure 12 shows the robot's preferred transitions for the puck and homing phase at the end of one of the asynchronous runs, with darker arrows indicating higher values. It is clear from both of these maps that the reinforcement value complex propagated useful values over the entire map.

6.2 The Second Arena

For both reinforcement learning robots in the second arena, the puck near the top of the arena was visited last before the end of the fifth cycle in all seven runs and therefore removed. In the random runs the same puck was removed in order to make the results maximally comparable.

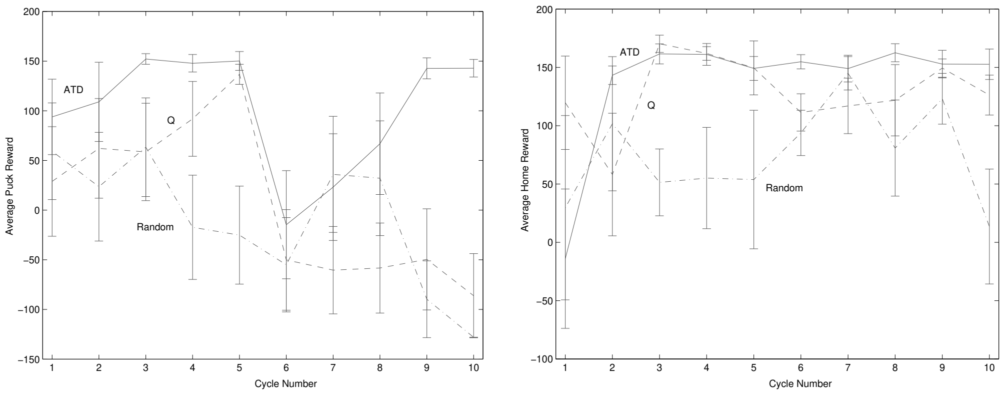
[4]These figures and the similar ones that follow were obtained by the superimposition of dead reckoning position estimation on a scale drawing of each map.



a b

Figure 12: Preferred transitions maps for the puck (a) and homing (b) reinforcement complexes taken from a sample asynchronous robot at the end of ten cycles in the first arena.

As can be seen from the graph of the average puck reward obtained over time for the systems in Figure 13a, both reinforcement learning models learned to find a puck relatively quickly at first, and then experience a sharp drop in performance at the end of the fifth cycle when it was removed, along with a high variation in reward as indicated by the large error bars. Although the asynchronous model is able to recover and return to a consistently good solution by the ninth cycle, the synchronous model does not on average perform much better or more consistently than the random system by the end of the run.



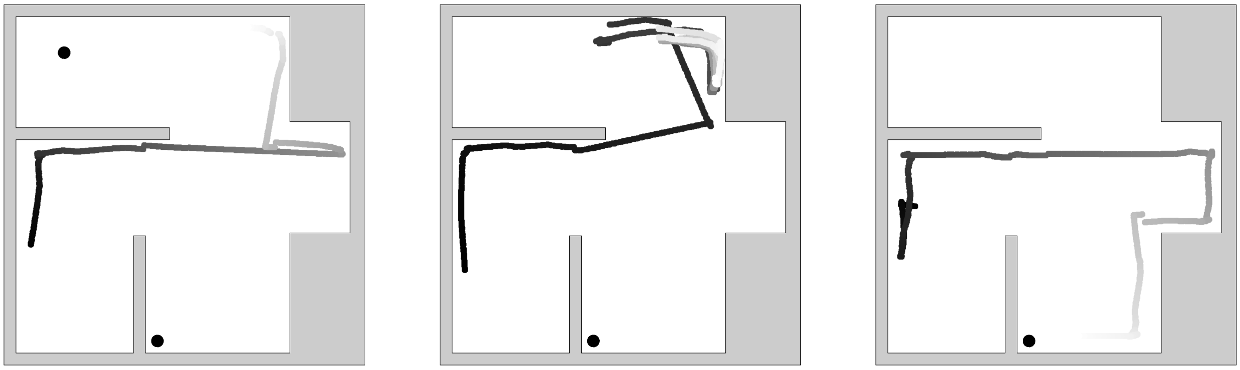
(a) (b)

Figure 13: Average puck (a) and home (b) reward over time for the second arena.

Figure 13b shows that both reinforcement learning types were able to learn to get back to the home area quickly, although the synchronous algorithm experiences a drop in performance and increase in standard error from the sixth cycle, only recovering around the ninth cycle. This seems to indicate that the synchronous algorithm is less robust than the asynchronous one.

The asynchronous model is therefore able to able to learn to adjust its value complex relatively quickly in the face of a noisy, modified environment. It does this despite the fact that the expected values it calculates are averages over all rewards received, so that some residual puck reward must remain at any transition where a puck has ever been sighted (this could be remedied by the use of an average over the last few sightings).

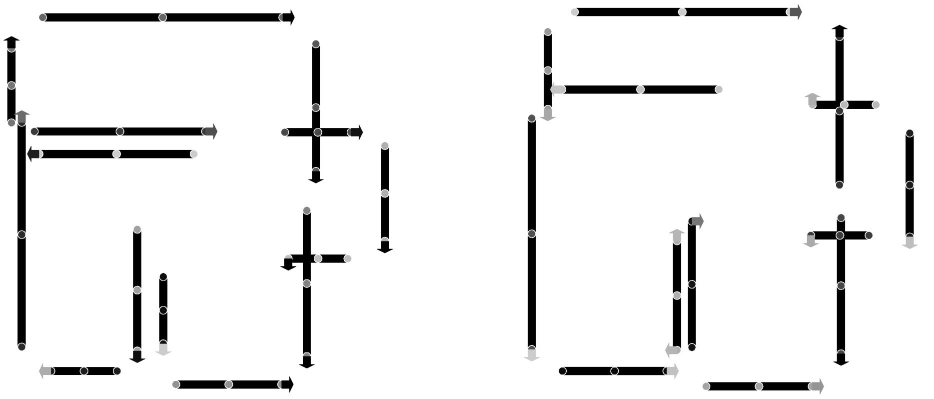
Figure 14 shows the puck finding behaviour displayed by the reinforcement learning robots. The figure on the left shows an example of the puck finding path initially learned by both reinforcement



a b c

Figure 14: Learned puck finding behaviour in the second arena: a) the initial puck finding path, b) the behavior exhibited by both reinforcement learning models when that puck is taken away, and c) the alternate route later learned by the asynchronous model.

models, and the figure in the middle displays the behaviour exhibited initially by both robot types after that puck has been taken away, where both robots repeatedly execute the transition that had previously led to a puck sighting. However, the asynchronous model is later consistently able to learn to take the alternate puck finding route, shown in the figure on the right, while the synchronous one is not.



a b

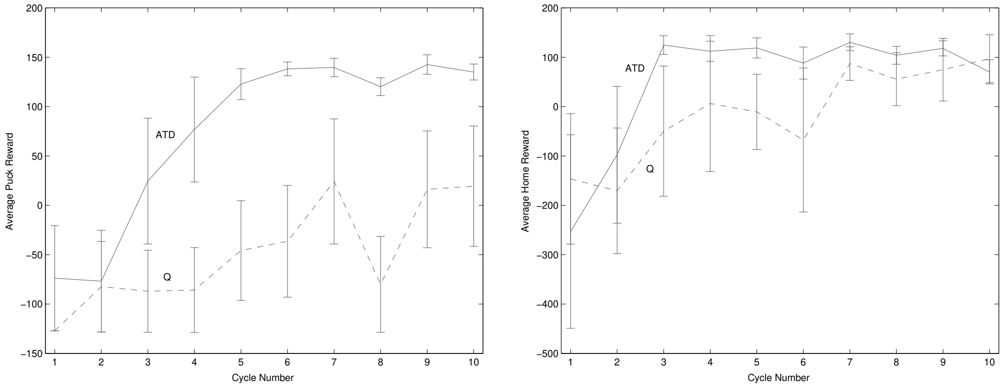
Figure 15: Preferred puck transitions maps for the second arena, from sample asynchronous (a) and synchronous (b) reinforcement learning robots after the eighth cycle.

Figure 15 shows the preferred puck transition maps after the eighth cycle for the asynchronous and synchronous robots. The map obtained from the asynchronous model (on the left) has adjusted well to the removal of the top puck and now directs movement to the lower puck from everywhere in the graph (note that two pairs of walls in the left map appear to be on the wrong side of each other due to dead reckoning error). The map obtained from the synchronous model, however, has not adjusted as well and still contains regions where movement would be directed toward the transition where the puck has been removed.

6.3 The Third Arena

The third arena was the most difficult arena faced by the robot, with the longest path to the puck and the most complex map. Because it had already been shown to perform poorly, no runs were performed with the random robots. In addition, data from only five reinforcement learning runs were used rather than seven.

Figure 16a shows the average puck reward over time for the third arena. It demonstrates decisively that the asynchronous algorithm outperforms the synchronous one when a long path to the goal must be constructed. The asynchronous algorithm consistently found and learned a short path to the puck by the sixth cycle, whereas the synchronous algorithm did not manage to consistently find a good path at all.



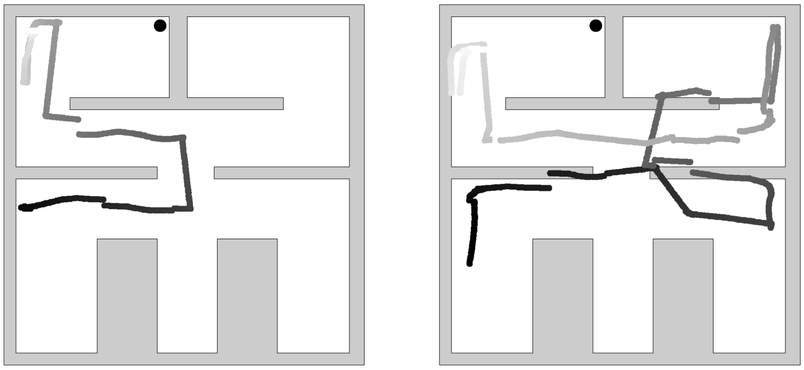
(a) (b)

Figure 16: Average puck (a) and home (b) reward over time for the third arena.

The difference between the two learning models is less pronounced in Figure 16b, which shows the average home reward obtained by the two robot types over time. The asynchronous model again consistently finds a good solution quickly, at around the fourth cycle, while the synchronous model takes longer, reaching the same conditions at around the seventh cycle, but still performs well.

A potential explanation for the difference in performance between the two rewards could be that since both models explore, and both must initially start all runs in the home area, the synchronous robot would experience many transitions near the home area and thus be able to build a path earlier than in the case of the puck, where it would be much less likely to experience a puck sighting repeatedly without having built a path to it first. The synchronous model is also likely to have explored the arena less thoroughly than the asynchronous model, and thus in many cases did not have had to find its way back from as far away.

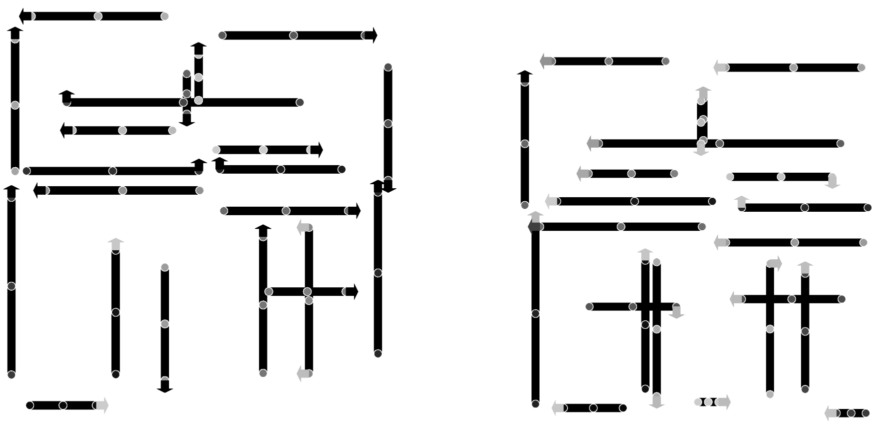
The path commonly learned by the asynchronous robots is shown on the left side of Figure 17. Even though this is a fairly complex arena, the robot manages to learn a direct path to the puck. A representative path for the synchronous model is shown on the right, and is clearly not as direct as the path learned by the asynchronous robots.



a b

Figure 17: Typical learned puck Routes in the third arena for the asynchronous (a) and synchronous (b) reinforcement learning robots.

Figure 18 shows the preferred puck transitions for asynchronous and synchronous robots. The map obtained from the asynchronous model (on the left) shows that the path to the puck has been propagated throughout the map, whereas it is clear from the map obtained from the synchronous model that the path to the puck has propagated slowly, with only the transitions very close to the puck transition having high values (indicated by dark arrows).



a b

Figure 18: Sample preferred puck transitions maps for asynchronous (a) and synchronous (b) reinforcement learning robots in the third arena.

One revealing aspect of the robot type's behaviour was the apparent repetition of transitions by the synchronous robots, where several transition experiences were required to drive the optimistic initial transition values down. The asynchronous robots repeated transitions only when it took multiple transition attempts to determine that an unexplored transition was illegal. This gave it more time to explore and allowed for wider coverage of the map, and may have contributed towards its superior performance.

6.4 Convergence

The final issue to be considered is that of convergence. Previously, we suggested that the amount of time a situated agent takes to make a transition may be sufficient to allow an asynchronous reinforcement learning algorithm over a topological map to converge between transitions.

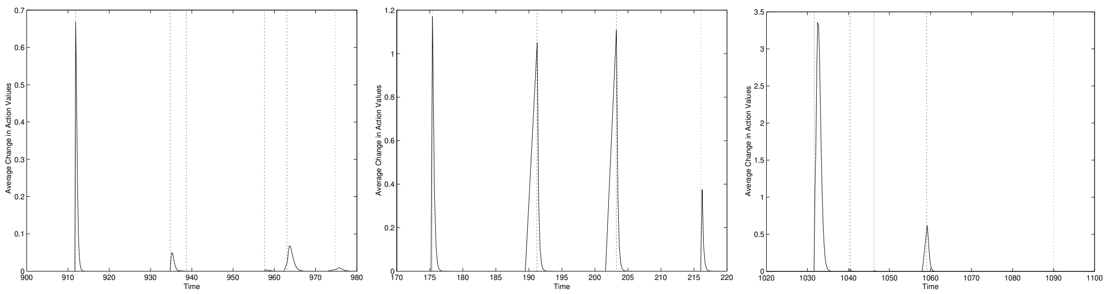


Figure 19: Samples of the average action value change over time from representative asynchronous reinforcement learning robots.

Figure 19 contains three samples (one from each arena, of 80, 50 and 80 seconds length, respectively) showing the average change in action value over time, where the dotted vertical lines mark transition occurrences. Transitions function as event points, where a decision as to what to do next is required and where the reinforcement learning complex receives new data. The third graph is from a run in the third arena where the puck was discovered after 17 minutes, and the robot had built a nearly complete map of the arena. In all three graphs, the action values are disturbed by each event point but converge comfortably before the next one occurs.

Although the remainder of the data shows a similar pattern, convergence cannot be conclusively demonstrated here because data could not be collected sufficiently rapidly to rule out the possibility of very small changes being made before some event points. For example, in the second graph in Figure 19, although it appears that some of the spikes begin before an event point, this is an artifact of the sampling rate of the event point data and the line interpolation graphing method used. Nevertheless it seems clear that reinforcement learning complex converges between decisions, and therefore no reinforcement learning method (not even TD(λ)) could usefully speed up learning any further.

6.5 Conclusion

It is clear from the results presented above that both reinforcement learning models perform better than the random movement strategy for this task, and that both are capable of learning to find the puck and return home again quickly. Although the synchronous method (Q-learning) performs roughly as well as the asynchronous method (ATD) when finding fairly short paths, the asynchronous model performs better when a long path must be learned, and recovers more quickly than the synchronous model when the environment changes. The results also suggest that the interplay of the exploration drive, the distributed map and the other drives may have subtle but important effects on the robot's overall performance. Finally, the data obtained from all of the runs suggests that the asynchronous model is able to converge between transitions, so that the choices made by the agent are optimal given the knowledge that it has. The reinforcement learning architecture presented in this paper is therefore feasible and capable of providing definite behavioural benefits for the puck foraging task.

7 Discussion

In the following sections, we discuss the issues raised by the model, experiment and results presented in this paper. We first examine the significance of the model, followed by the limitations of the work presented here. Finally, we consider the implications of this research for reinforcement learning and situated learning in general.

7.1 Significance

This paper has shown that the behavior-based style of robot control and reinforcement learning methods can be fully integrated to produce an autonomous robot that is capable of rapid learning in a real environment. Further application of the architecture presented here has the potential to widen the scope of behavior-based systems by facilitating the synthesis of other robots that, like Dangerous Beans, display reinforcement learning behavior.

Furthermore, this research may form a basis for future work since reinforcement learning provides a principled way to build value-driven learning agents and the reinforcement learning complex provides a structure that allows for the layering and integration of further learning methods into behavior-based control systems.

Finally, this research has indicated that learning in situated agents requires a change in emphasis from the design criteria employed by standard machine learning, and that the integration of further learning methods into behavior-based robotics through layered learning is a promising future research direction.

7.2 Limitations

The work presented here suffers from two types of limitations: those inherent in the reinforcement learning model, and those relating to the experiment and implementation used to test it.

A major limitation of the architecture is the fact that it relies upon a learned topological map. In some situations, it may not be possible to feasibly build or maintain such a map, and in others, the map may become prohibitively large. In cases where the map becomes large, the reinforcement learning layer on top of it is unlikely to be able to learn quickly enough to be useful in real time. Such situations may require the addition of other learning methods, or one or more further topological mapping layers in order obtain a map that is small enough to be useful [5].

The other primary difficulty inherent in the model is that it inherits all of the limitations of reinforcement learning. Reinforcement learning is not appropriate for all types of problems, and imposes significant state property requirements. States must have the Markov property, which may be difficult to obtain in situated environments due to perceptual aliasing, where different states are perceptually indistinguishable. In such cases, the use of further internal state or active perception may be required in order to disambiguate states (Crook & Hayes, 2003).

The experimental design and implementation presented above are not perfect. One of the major drawbacks of the experimental implementation was the level of engineering required to get everything to work in terms of both the environment and the robot control system. The environment had to be engineered to a certain extent because of the sensory limitations of our robot, and a great deal of effort was required to get the distributed map building system (in particular the dead-reckoning correction) performing as required. Most of these problems could have been solved with the addition of more sensory capabilities to Dangerous Beans – for example, the dead-reckoning system would have been much more accurate with the addition of a direction sense, and could even have been dropped completely from the control system had the robot had sufficient sensing to unambigously differentiate walls. However, none of these limitations significantly detract from the point that system was built to prove.

Finally, the experimental results given here are the results of the use of the architecture in a single application area only. Although these results strongly suggest that it is promising and may work in other domains, further experiments will be required in order to confirm this.

7.3 Implications

This section considers some of the implications of the research presented here for situated reinforcement learning and robot learning in general. The following sections consider the implications for situated reinforcement learning, planning behavior, layered learning and emergent representations in turn, with the last section providing a highly speculative discussion of the role of learning models in the study of situated representation.

7.3.1 Situated Reinforcement Learning

A situated agent is required to learn in real time, using a reasonable amount of computation, in a manner that provides behavioral benefits within its lifetime. The evolution of complex methods to make already present learning strategies feasible is difficult to justify, especially when the naive implementations of such methods can provide no behavioral benefit to an agent in its lifetime. Thus, although a great deal of work has gone into making reinforcement learning using a single control process feasible over very large state spaces, a similar effort should be directed towards the development of methods (such as layering reinforcement learning over a topological map) that create conditions under which it is feasible in the first place. Two promising approaches to this are the use of a priori knowledge and learning bias (Bryson, 2002) and layered learning (Stone & Veloso, 2000).

The results presented here suggest that a learning method developed specifically for the situated case can generate real time adaptive behavior, and that situated learning requires a different design methodology than standard machine learning, thus meriting further study in its own right.

7.3.2 Planning Behavior

One of the original criticisms of behavior-based robotics was that systems built using it would never be able to plan, because its emphasis on distributed control, reactive behavior and a lack of syntactic representations preclude the use of traditional planning algorithms (Brooks, 1987). Although the situated view of intelligent agents does not consider the construction and execution of plans to be the primary activity performed by an intelligent agent in the same way that classical artificial intelligence does (Agre & Chapman, 1990, Brooks, 1987), the generation of some form of planning behavior is still an important aspect of intelligence.

[5]Layering further topological mapping layers can be considered in some sense equivalent to hierarchical reinforcement learning methods (e.g., Digney (1998)).

As argued by Sutton (1991) and Barto, Bradtke, and Singh (1995), model-based reinforcement learning methods incrementally converge to a compiled reactive plan, resulting in a simple, local policy that optimally maximises total future reward. Therefore, given a particular model of the problem space, once a model-based reinforcement learning method (such as the one used in this paper) has converged, any decision made using the resulting policy is optimal and implicitly takes into account the value of all future actions. Reinforcement learning using a model is therefore a kind of stochastic, real time planning algorithm (Barto et al., 1995).

The results presented here show that Dangerous Beans can be said to be displaying planning behavior, because its reinforcement learning complex appears to converge between decisions. Therefore, the decisions it makes are globally optimal given its drives and the knowledge it has, even though they are in a sense reactive decisions. Dangerous Beans is thus a behavior-based robot capable of planning without the use of a planner in the classical sense, because although it makes reactive decisions, the values it uses to make them implicitly reflect global expected return, and are thus not prone to the myopia usually exhibited by reactive systems.

7.3.3 Layered Learning

Layering reinforcement learning on top of a topological map is just one instance of the layered learning approach introduced by Stone and Veloso (2000), which is a powerful and general idea that has not yet been fully explored.

In the original layered learning methodology, one layer was required to complete its learning task before the next could begin (Stone & Veloso, 2000). One of the interesting implications of this research is that when two layers are learning at the same time, a kind of feedback between layered learning systems is possible, where the performance of each algorithm biases the other's learning opportunities. Since most machine learning research has concentrated on only one learning method in isolation from all the others, there may be significant scope for future research into the kinds of biases that two interacting learning algorithms can impose on each other.

Another interesting aspect of using one type of learning to make another feasible is that it suggests an information requirement ordering for learning models. There may be a hierarchical relationship between all learning methods governing at which level of behavior in a situated agent's control system they can appear. Perhaps a similar evolutionary ordering exists where species must evolve some types of learning methods before others in order to obtain the behavioral benefits that could give them a fitness advantage. For example, once Dangerous Beans was capable of distributed map learning, the addition of a reinforcement learning layer provided it with significant behavioral benefits but required far less engineering effort than that required to develop the map learning layer in the first place. There may be scope for the investigation of these kinds of relationships between learning models through artificial evolution (Harvey, 1995).

Layered learning may also have interesting implications in terms of emergent behavior. Since the interaction of multiple control processes and a complex environment results in complex behavior, it is reasonable to expect that the interaction of multiple learning models, multiple control processes, and a complex environment will likewise result in complex learning behavior.

7.3.4 Emergent Representations

The behavior-based approach to artificial intelligence has resulted in a change in the way that many artificial intelligence researchers view behavior. Behavior is now considered to be the emergent result of the complex interaction between an agent's multiple control processes, morphology, and environment. However, there has been no corresponding change in the way that researchers view representation. Some behavior-based roboticists simply ignore it, while others maintain the classical view that representation is the result of an internal syntactic symbol system.

Simply ignoring the role of representation in intelligent behavior is not a tenable position. The real question is not whether or not representation must exist in an intelligent agent, but rather *in what form* and *in what manner* it exists. Although Brooks (1991a) is often considered an argument against any form of representation whatsoever[6], it is actually an argument against the use of centralised syntactic representational systems for control. In fact, Brooks (1991a) claims that useful representations should be distributed and emergent, and that such things would be sufficiently different from what are traditionally considered representations that they should be called something else.

[6] It is, after all, entitled "Intelligence without Representation".

One powerful way to study representations in situated agents is through learning. Representations themselves do not offer an agent a behavioral advantage – rather, the simple fact that anything new an agent wishes to represent must first be learned implies that the types of learning methods used by the agent dictate its representations and the behavioral advantages it receives.

This imposes a dual constraint on the types of representations an agent will find beneficial: the relevant learning model must be feasible in that it must be able to learn in real time, and the behavior it generates must be useful, in that it provides behavioral advantages appropriate to the agent's level of competence. Since situated learning is only feasible when it is task-specific, it follows that such representations are also likely to be task specific. In the same way that there is never likely to be a single general purpose tractable learning algorithm, there is no known representational system that is capable of handling a wide spectrum of knowledge at different levels of detail without becoming computationally intractable. Situated intelligence should be expected to develop in such a way as to facilitate cheap computation and rapid learning, whenever possible.

If we assume that representation arises through the presence of learning methods, it follows that representations must form from the structures generated by those methods and their interaction with each other. These representations would be emergent in the sense that they would not be composed of atomic, syntactic symbols, but would instead be complex entities formed by the association of several task-specific structures at different levels of detail, organised into a loosely hierarchical distributed structure. Such complexes may only be identifiable as symbols given a particular linking context.

For example, when observing the behavior of Dangerous Beans, an outside observer would say that the robot has a representation of a map, and a representation of the path to the puck. However, Dangerous Beans has a distributed map, which is an emergent structure – it exists because of the behavior and interaction of its behavioral modules with the environment and each other. Similarly, nowhere in Dangerous Beans' control structure is there a path representation. Rather, there is a reinforcement learning complex, another emergent structure, that consists of a set of values that cause the agent to make certain choices when it is at certain places. The path is an emergent property of these choices, and it results from the interaction of the robot's internal drives, the distributed map, the reinforcement learning complex embedded in it, and the environment itself – the path to the puck does not exist outside of the interaction of these elements.

8 Conclusion

The contribution of this paper is threefold. First, it has introduced an architecture that integrates the behavior-based style of robot architecture and reinforcement learning. Second, it has detailed the development of a mobile robot that uses this model to learn to solve a difficult problem, resulting in data that supports the claim that the architecture is capable of learning in real time. Finally, through the development of a fully behavior-based layered learning system, some progress has been made towards bringing these two powerful and important ideas together.

If behavior-based systems are to have any hope of moving beyond insect-level intelligence, they must begin to incorporate learning mechanisms at all levels of behavior. This is more than just a matter of inserting learning models into behavior-based systems – it is a matter of understanding what is required to make learning feasible in the real world, how to layer learning methods so that their interaction facilitates the generation of complex behavior, and how to truly integrate learning into behavior-based systems. The research presented here represents one small step in that direction.

Acknowledgements

We would like to thank Joanna Bryson and the anonymous reviewers for for their invaluable comments, and the University of Edinburgh for the use of its resources. George Konidaris was supported by a Commonwealth Scholarship (ref. ZACS-2002-344) during this research.

References

Agre, P., & Chapman, D. (1990). What are plans for? In P. Maes (Ed.), *New architectures for autonomous agents: Task-level decomposition and emergent functionality*. Cambridge, MA: MIT Press.

Balch, T. (1997a). *Clay: Integrating motor schemas and reinforcement learning* (Technical Report GIT-CC-97-11). College of Computing, Georgia Institute of Technology.

Balch, T. (1997b). Integrating RL and behavior-based control for soccer. *RoboCup-97: Proceedings of the First Robot World Cup Soccer Games and Conferences*. Springer-Verlag.

Balch, T. (1999). Reward and diversity in multirobot foraging. *Proceedings of the IJCAI Workshop on Agents Learning About, From and With Other Agents*.

Barto, A., Bradtke, S., & Singh, S. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, *72*.

Bertsekas, D., & Tsitsiklis, J. (1989). *Parallel and distributed computation: Numerical methods*. Prentice-Hall.

Brooks, R. (1987). Planning is just a way of avoiding figuring out what to do next. In R. Brooks (Ed.), *Cambrian intelligence : the early history of the new AI* (pp. 103–110). Cambridge, Massachusetts: The MIT Press.

Brooks, R. (1991a). Intelligence without representation. In J. Haugeland (Ed.), *Mind design II* (pp. 395–420). Cambridge, Massachusetts: MIT Press.

Brooks, R. (1991b). The role of learning in autonomous robots. *Proceedings of the Fourth Annual Workshop on Computational Learning Theory (COLT '91)* (pp. 5–10). Santa Cruz, CA.

Bryson, J. (2002). Modularity and specialized learning: Reexamining behavior-based artificial intelligence. *Proceedings of the Workshop on Adaptive Behavior in Anticipatory Learning Systems*. Springer.

Choset, H., & Nagatani, K. (2001). Topological simultaneous localization and mapping (SLAM): Towards exact localization without explicit localization. *IEEE Transactions on Robotics and Automation*, *17*(2).

Crook, P., & Hayes, G. (2003). Learning in a state of confusion: Perceptual aliasing in grid world navigation. *Proceedings of the 4th British Conference on (Mobile) Robotics: Towards Intelligent Mobile Robots (TIMR 2003)*.

Digney, B. (1998). Learning hierarchical control structures for multiple tasks and changing environments. In R. Pfeifer, B. Blumberg, J. Meyer, & S. Wilson (Eds.), *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*. MIT Press.

Dudek, G., Freedman, P., & Hadjres, S. (1993). Using local information in a non-local way for mapping graph-like worlds. *Proceedings of the International Joint Conference of Artificial Intelligence*. Chambery, France.

Grefenstette, J., & Ramsey, C. (1992). An approach to anytime learning. *Proceedings of the Ninth International Conference on Machine Learning*.

Harvey, I. (1995). The artificial evolution of adaptive behaviour. DPhil thesis, School of Cognitive and Computing Sciences, University of Sussex.

Humphrys, M. (1996). Action selection methods using reinforcement learning. *From Animals to Animats 4: the Fourth International Conference on the Simulation of Adaptive Behaviour (SAB-96)*. MIT Press.

K-Team SA (1999a). *Khepera K213 vision turret user manual*. Lausanne, Switzerland.

K-Team SA (1999b). *Khepera user manual*. Lausanne, Switzerland.

Kohonen, T. (1989). *Self-organization and associative memory*. Springer-Verlag, 3rd edition.

Konidaris, G. (2003). Behaviour-based reinforcement learning. Master's thesis, School of Informatics, University of Edinburgh.

Maes, P., & Brooks, R. (1990). Learning to coordinate behaviors. *Proceedings of the American Association of Artificial Intelligence* (pp. 796–802). Boston, MA.

Mahadevan, S., & Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, *55*(2–3), 311–365.

Marsland, S., Shapiro, J., & Nehmzow, U. (2002). A self-organising network that grows when required. *Neural Networks*, *15*(8–9), 1041–1058.

Matarić, M. (1994). Reward functions for accelerated learning. *International Conference on Machine Learning* (pp. 181–189).

Matarić, M. (1997). Reinforcement learning in the multi-robot domain. *Autonomous Robots*, *4*(1), 73–83.

Matarić, M., & Brooks, R. (1990). Learning a distributed map representation based on navigation behaviors. In R. Brooks (Ed.), *Cambrian intelligence : the early history of the new AI*. Cambridge, Massachusetts: The MIT Press.

Mitchell, T. (1997). *Machine learning*. McGraw-Hill.

Moriarty, D., Schultz, A., & Grefenstette, J. (1999). Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, *11*.

Schmolke, A., & Mallot, H. (2002). Polarization compass for robot navigation. *The Fifth German Workshop on Artificial Life* (pp. 163–167).

Smart, W., & Kaelbling, L. (2000). Practical reinforcement learning in continuous spaces. *Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 903–910).

Smith, A. J. (2002). Applications of the self-organising map to reinforcement learning. *Neural Networks*, *15*, 1107–1124.

Stone, P., & Veloso, M. (2000). Layered learning. *Proceedings of the 11th European Conference on Machine Learning* (pp. 369–381). Springer.

Sutton, R. (1990). Reinforcement learning architectures for animats. In J. Meyer, & S. Wilson (Eds.), *From Animals to Animats: Proceedings of the International Conference on Simulation of Adaptive Behavior*. MIT Press.

Sutton, R. (1991). Planning by incremental dynamic programming. *Proceedings of the Ninth Conference on Machine Learning* (pp. 353–357).

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Toombs, S., Phillips, W., & Smith, L. (1998). Reinforcement landmark learning. In R. Pfeifer, B. Blumberg, J. Meyer, & S. Wilson (Eds.), *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*. MIT Press.

Watkins, C., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*, 279–292.

Whiteson, S., & Stone, P. (2003). Concurrent layered learning. *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*.