$\mathcal{Q}$-Learning in Continuous State and Action Spaces

Chris Gaskett, David Wettergreen, and Alexander Zelinsky

Robotic Systems Laboratory
Department of Systems Engineering
Research School of Information Sciences and Engineering
The Australian National University
Canberra, ACT 0200 Australia
[cg|dsw|alex]@syseng.anu.edu.au

Abstract. $\mathcal{Q}$-learning can be used to learn a control policy that maximises a scalar reward through interaction with the environment. $\mathcal{Q}$-learning is commonly applied to problems with discrete states and actions. We describe a method suitable for control tasks which require continuous actions, in response to continuous states. The system consists of a neural network coupled with a novel interpolator. Simulation results are presented for a non-holonomic control task. Advantage Learning, a variation of $\mathcal{Q}$-learning, is shown enhance learning speed and reliability for this task.

1 Introduction

Reinforcement learning systems learn by trial-and-error which actions are most valuable in which situations (states) [1]. Feedback is provided in the form of a scalar *reward* signal which may be delayed. The reward signal is defined in relation to the task to be achieved; reward is given when the system is successfully achieving the task. The *value* is updated incrementally with experience and is defined as a discounted sum of expected future reward. The learning systems choice of actions in response to states is called its *policy*. Reinforcement learning lies between the extremes of supervised learning, where the policy is taught by an expert, and unsupervised learning, where no feedback is given and the task is to find structure in data.

There are two prevalent approaches to reinforcement learning: $\mathcal{Q}$-learning and actor-critic learning. In $\mathcal{Q}$-learning [2] the expected value of each action in each state is stored. In $\mathcal{Q}$-learning the policy is formed by executing the action with the highest expected value. In actor-critic learning [3] a critic learns the value of each state. The value is the expected reward over time from the environment under the current policy. The actor tries to maximise a local reward signal from the critic by choosing actions close to its current policy then changing its policy depending upon feedback from the critic. In turn, the critic adjusts the value of states in response to rewards received following the actor's policy.

The main advantage of $\mathcal{Q}$-learning over actor-critic learning is exploration insensitivity—the ability to learn without necessarily following the current policy. However, actor-critic learning has a major advantage over current implementations of $\mathcal{Q}$-learning; the ability to respond to smoothly varying states with smoothly varying actions. Actor-critic systems can form a continuous mapping from state to action and update this policy based on the local reward signal from the critic. $\mathcal{Q}$-learning is generally considered in the case that states and actions are both discrete. In some real world situations, and especially in control, it is advantageous to treat both states and actions as continuous variables.

This paper describes a continuous state and action $\mathcal{Q}$-learning method and applies it to a simulated control task. Essential characteristics of a continuous state and action $\mathcal{Q}$-learning system are also described. Advantage Learning [4] is found to be an important variation of $\mathcal{Q}$-learning for these tasks.

2 $\mathcal{Q}$-Learning

$\mathcal{Q}$-learning works by incrementally updating the expected values of actions in states. For every possible state, every possible action is assigned a value which is a function of both the immediate reward for taking that action and the expected reward in the future based on the new state that is the result of taking that action. This is expressed by the one-step $\mathcal{Q}$-update equation,

$$\mathcal{Q}(\boldsymbol{x}, \boldsymbol{u}) := (1-\alpha)\,\mathcal{Q}(\boldsymbol{x}, \boldsymbol{u}) + \alpha\,(R + \gamma \max \mathcal{Q}(\boldsymbol{x}_{t+1}, \boldsymbol{u}_{t+1})) \quad , \tag{1}$$

where $\mathcal{Q}$ is the expected value of performing action $\boldsymbol{u}$ in state $\boldsymbol{x}$; $\boldsymbol{x}$ is the state vector; $\boldsymbol{u}$ is the action vector; R is the reward; α is a learning rate which controls convergence and γ is the discount factor. The discount factor makes rewards earned earlier more valuable than those received later.

This method learns the values of all actions, rather than just finding the optimal policy. This knowledge is expensive in terms of the amount of information which has to be stored, but it does bring benefits. $\mathcal{Q}$-learning is exploration insensitive, any action can be carried out at any time and information is gained from this experience. Actor-critic learning does not have this ability, actions must follow or nearly follow the current policy. This exploration insensitivity allows $\mathcal{Q}$-learning to learn from other controllers, even if they are directed toward achieving a different task they can provide valuable data. Knowledge from several $\mathcal{Q}$-learners can be combined, as the values of non-optimal actions are known, a compromise action can be found.

In the standard $\mathcal{Q}$-learning implementation $\mathcal{Q}$-values are stored in a table. One cell is required per combination of state and action. This implementation is not amenable to continuous state and action problems.

3 Continuous States and Actions

Many real world control problems require actions of a continuous nature, in response to continuous state measurements. *It should be possible that actions vary smoothly in response to smooth changes in state.*

But most learning systems, indeed most classical AI techniques, are designed to operate in discrete domains, manipulating symbols rather than real numbered variables. Some problems that we may wish to address, such as high-performance control of mobile robots, cannot be adequately carried out with coarsely coded inputs and outputs. Motor commands need to vary smoothly and accurately in response to continuous changes in state.

$\mathcal{Q}$-learning with discretised states and actions scale poorly. As the number of state and action variables increase, the size of the table used to store $\mathcal{Q}$-values grows exponentially. Accurate control requires that variables be quantised finely, but as these systems fail to generalise between similar states and actions, they require large quantities of training data. If the learning task described in Sect. 7 was attempted with a discrete $\mathcal{Q}$-learning algorithm the number of $\mathcal{Q}$-values to be stored in the table would be extremely large. For example, discretised roughly to seven levels, the eight state variables and two action variables would require almost 300 million elements. Without generalisation, producing this number of experiences is impractical. Using a coarser representation of states leads to aliasing, functionally different situations map to the same state and are thus indistinguishable.

It is possible to avoid these discretisation problems entirely by using learning methods which can deal directly with continuous states and actions.

4 Continuous State and Action $\mathcal{Q}$-Learning

There have been several recent attempts at extending the $\mathcal{Q}$-learning framework to continuous state and action spaces [5, 6, 7, 8, 9].

We believe that there are eight criteria that are necessary and sufficient for a system to be capable of this type of learning. Listed in in Fig. 1, these requirements are a combination of those required for basic $\mathcal{Q}$-learning as described in Sect. 2 combined with the type of continuous behaviour described in Sect. 3. None of the $\mathcal{Q}$-learning systems discussed below appear to fulfil all of these criteria completely. In particular, many systems cannot learn a policy where actions vary smoothly with smooth changes in state (criteria *Continuity*). In these not-quite continuous systems a small change in state cannot cause a small change in action. In effect the function which maps state to action is a staircase—a piecewise constant function.

Sections 4.1–4.6 describe various real valued state and action $\mathcal{Q}$-learning methods and techniques and rate them (in an unfair and biased manner) against the criteria in Fig. 1.

4.1 Adaptive Critic Methods

Werbos's adaptive critic family of methods [5] use several feedforward artificial neural networks to implement reinforcement learning. The adaptive critic family includes methods closely related to actor-critic and $\mathcal{Q}$-learning. A learnt dynamic model assists in assigning reward to components of the action vector (not meeting

Action Selection:	Finds *action* with the highest expected value quickly.
State Evaluation:	Finds *value* of a *state* quickly as required for the $\mathcal{Q}$-update equation (1). A state's value is the value of highest valued action in that state.
$\mathcal{Q}$ Evaluation:	Stores or approximates the entire $\mathcal{Q}$-function as required for the $\mathcal{Q}$-update equation (1).
Model-Free:	Requires no model of system dynamics to be known or learnt.
Flexible Policy:	Allows representation of a broad range of policies to allow freedom in developing a novel controller.
Continuity:	Actions can vary smoothly with smooth changes in state.
State Generalisation:	Generalises between similar states, reducing the amount of exploration required in state space.
Action Generalisation:	Generalises between similar actions, reducing the amount of exploration required in action space.

Fig. 1. Essential capabilities for a continuous state and action $\mathcal{Q}$-learning system

the *Model-Free* criteria). If the dynamic model is already known, or learning one is easier than learning the controller itself, model based adaptive critic methods are an efficient approach to continuous state, continuous action reinforcement learning.

4.2 CMAC Based $\mathcal{Q}$-learning

Santamaria, Ashwin and Sutton [6] have presented results for $\mathcal{Q}$-learning systems using Albus's CMAC (Cerebellar Model Articulation Controller) [10]. The CMAC is a function approximation system which features spatial locality, avoiding the unlearning problem described in Sect. 6. It is a compromise between a look up table and a weight-based approximator. It can generalise between similar states, but it involves discretisation, making it impossible to completely fulfil the *Continuity* criteria. In [6] the inputs to the CMAC are the state and action, the output is the expected value. To find $\mathcal{Q}_{\max}$ this implementation requires a search across all possible actions, calculating the $\mathcal{Q}$-value for each to find the highest. This does not fulfil the *Action Selection* criteria.

Another concern is that approximation resources are used evenly across the state and action spaces. Santamaria et. al. address this by pre-distorting the state information using a priori knowledge so that more important parts of the state space receive more approximation resources.

4.3 $\mathcal{Q}$-AHC

Rummery presents a method which combines $\mathcal{Q}$-learning with actor-critic learning [7]. $\mathcal{Q}$-learning is used to chose between a set of actor-critic learners. Its performance overall was unsatisfactory. In general it either set the actions to

constant settings, making it equivalent to Lin's system for generalising between states [11], or only used one of the actor-critic modules, making it equivalent to a standard actor-critic system. These problems may stem from not fulfilling *$\mathcal{Q}$ Evaluation*, *Action Generalisation* and *State Generalisation* criteria when different actor-critic learners are used. This system is one of the few which can represent non-piecewise constant policies (*Continuity* criteria).

4.4 $\mathcal{Q}$-Kohonen

Touzet describes a $\mathcal{Q}$-learning system based on Kohonen's self organising map [8, 12]. The state, action and expected value are the elements of the feature vector. Actions are chosen by choosing the node which most closely matches the state and a the maximum possible value (one). Unfortunately the actions are always piecewise constant, not fulfilling the *Continuity* criteria.

4.5 $\mathcal{Q}$-Radial Basis

Santos describes a system based on radial basis functions [13]. It is very similar to the $\mathcal{Q}$-Kohonen system in that each radial basis neuron's holds a center vector like the Kohonen feature vector. The number of possible actions is equal to the number of radial basis neurons, so actions are piecewise constant (not fulfilling the *Continuity* criteria). It does not meet the *$\mathcal{Q}$ Evaluation* criteria as only those actions described by the radial basis neurons have an associated value.

4.6 Neural Field $\mathcal{Q}$-learning

Gross, Stephan and Krabbes have implemented a $\mathcal{Q}$-learning system based on dynamic neural fields [9]. A neural vector quantiser (Neural Gas) clusters similar states. A neural field encodes the values of actions so that selecting the action with the highest $\mathcal{Q}$ requires iterative evaluation of the neural field dynamics. This limits the speed with which actions can be selected (the *Action Selection* criteria) and values of states found (the *State Evaluation* criteria). The system fulfils the *State Generalisation* and *Action Generalisation* criteria.

4.7 Our Approach

We seek a method of learning the control for a continuously acting agent functioning in the real world, for example a mobile robot travelling to goal location. For this application of reinforcement learning, the existing approaches have shortcomings that make them inappropriate for controlling this type of system. Many can't adequately generalise between states and/or actions. Others can't produce smoothly varying control actions or can't generate actions quickly enough for operation in real time. For these reasons we propose a scheme for reinforcement learning that uses a neural network and an interpolator to approximate the $\mathcal{Q}$-function.

5 Wire-fitted Neural Network $\mathcal{Q}$-Learning

Wire-fitted Neural Network $\mathcal{Q}$-Learning is a continuous state, continuous action $\mathcal{Q}$-learning method. It couples a single feedforward artificial neural network with an interpolator ("wire-fitter") to fulfil all the criteria in Fig. 1.

Feedforward Artificial Neural networks have been used successfully to generalise between similar states in $\mathcal{Q}$-learning systems where actions are discrete [11, 7]. If the output from the neural network describes (non-fixed) actions and their expected values, an interpolator can be used to generalise between them. This would fulfil the *State Generalisation* and *Action Generalisation* criteria.

Baird and Klopf [14] describe a suitable interpolation scheme called "wire-fitting". The wire-fitting function is a moving least squares interpolator, closely related to Shepard's function [15]. Each "wire" is a combination of an action vector, $\boldsymbol{u}$, and its expected value, q, which is a sample of the $\mathcal{Q}$-function. Baird and Klopf used the wire-fitting function in a memory based reinforcement learning scheme. In our system these parameters describing wire positions are the output of a neural network, whose input is the state vector, $\boldsymbol{x}$.

Figure 2 is an example of wire-fitting. The action is this case is one dimensional, but the system supports many dimensional actions. The example shows the graph of action versus value ($\mathcal{Q}$) *for a particular state*. The number of wires is fixed, the position of the wires changes to fit new data. Required changes are calculated using the partial derivatives of the wire-fitting function. Once new wire positions have been calculated the neural network is trained to output these new positions.

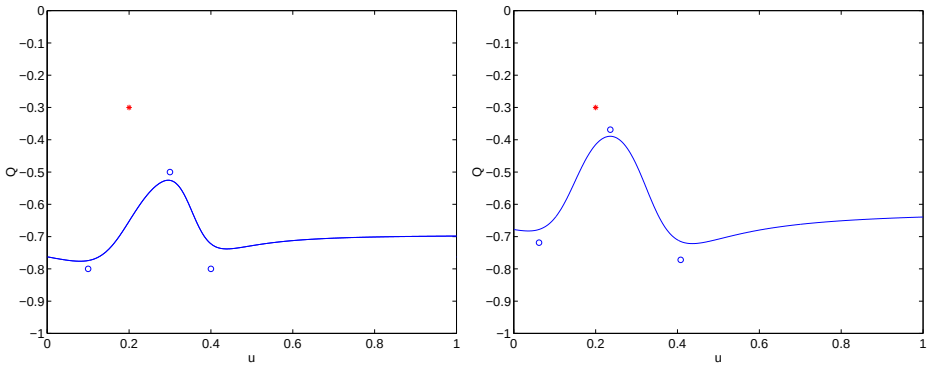


Fig. 2. The wire-fitting process. The action (u) is one dimensional in this case. Three wires (shown as $\circ$), this is the output from the neural network for a particular state. The wire-fitting function interpolates between the wires to calculate $\mathcal{Q}$ for every u. The new data ($*$) does not fit the curve well (*left*), so the wires are moved according to partial derivatives (*right*). In other states the wires would be in different positions.

The wire-fitting function has several properties which make it a useful interpolator for implementing $\mathcal{Q}$-learning.

Updates to the $\mathcal{Q}$-value (1) require $\mathcal{Q}_{\max}(\boldsymbol{x}, \boldsymbol{u})$, which can be calculated quickly with the wire-fitting function (the *State Evaluation* criteria).

The action $\boldsymbol{u}$ for $\mathcal{Q}_{\max}(\boldsymbol{x}, \boldsymbol{u})$ can also be calculated quickly (the *Action Selection* criteria). This is needed when choosing an action to carry out. A property of this interpolator is that the highest interpolated value always coincides with the highest valued interpolation point, so the action with the highest value is always one of the the input actions. When choosing an action it is sufficient to propagate the state through the neural network, then compare the output q to find the best action. The wire-fitter is not required at this stage, the only calculation is the forward pass through the neural network.

Wire-fitting also works with many dimensional scattered data while remaining computationally tractable; no inversion of matrices is required. Interpolation is local, only points nearby influence the value of $\mathcal{Q}$. Areas far from all wires have a value which is the average of $\boldsymbol{q}$, wild extrapolations do not occur (see Fig. 2). It does not suffer from oscillations, unlike most polynomial schemes.

Importantly, partial derivatives in terms of each q and $\boldsymbol{u}$ of each point can be calculated quickly. These partial derivatives allow error in the output of the $\mathcal{Q}$-function to be propagated to the neural network according to the chain rule.

This combination of neural network and interpolator stores the entire $\mathcal{Q}$ function (the *$\mathcal{Q}$ Evaluation* criteria). It represents policies in a very flexible way; it allows sudden changes in action in response to a change in state by changing wires, while also allowing actions to change smoothly in response to changes in state (the *Continuity* and *Flexible Policy* criteria).

The training algorithm is shown in figure 3.

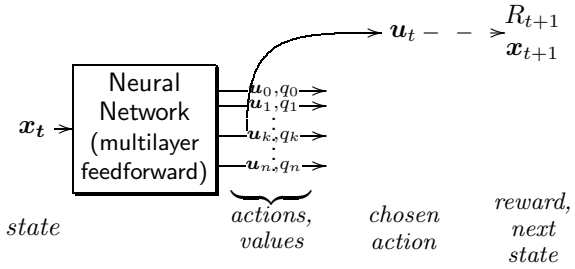
Training of the single hidden layer, feedforward neural network is by incremental backpropagation. The learning rate is kept constant throughout. Tan-sigmoidal neurons are used, restricting the magnitude of actions and values to between 1 and -1.

The wire-fitting function is

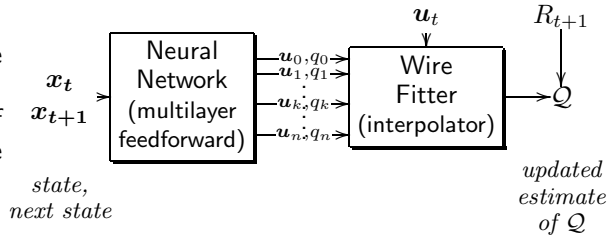
$$\begin{aligned}
\mathcal{Q}(\boldsymbol{x}, \boldsymbol{u}) &= \lim_{\epsilon \to 0^+} \frac{\sum_{i=0}^{n} \frac{q_i(\boldsymbol{x})}{\|\boldsymbol{u}-\hat{\boldsymbol{u}}_i(\boldsymbol{x})\|^2 + c(q_{\max}(\boldsymbol{x}) - q_i(\boldsymbol{x})) + \epsilon}}{\sum_{i=0}^{n} \frac{1}{\|\boldsymbol{u}-\hat{\boldsymbol{u}}_i(\boldsymbol{x})\|^2 + c(q_{\max}(\boldsymbol{x}) - q_i(\boldsymbol{x})) + \epsilon}} \\
&= \lim_{\epsilon \to 0^+} \frac{\sum_{i=0}^{n} \frac{q_i(\boldsymbol{x})}{distance(\boldsymbol{x}, \boldsymbol{u})}}{\sum_{i=0}^{n} \frac{1}{distance(\boldsymbol{x}, \boldsymbol{u})}} \\
&= \lim_{\epsilon \to 0^+} \frac{wsum(\boldsymbol{x}, \boldsymbol{u})}{norm(\boldsymbol{x}, \boldsymbol{u})} \quad ,
\end{aligned} \tag{2}$$

where i is the wire number; n is the total number of wires; $\boldsymbol{x}$ is the state vector; $\boldsymbol{u}_i(\boldsymbol{x})$ is the ith action vector; $q_i(\boldsymbol{x})$ is the value of the ith action vector; $\boldsymbol{u}$ is the action vector to be evaluated, c is a small smoothing factor and ϵ avoids division by zero. The dimensionality of the action vectors $\boldsymbol{u}$ and $\boldsymbol{u}_i$ is the number of continuous variables in the action. The two simplified forms shown simplify

1. In real time, feed the state into the neural network. Carry out the action with the highest q. Store the resulting change in state.



2. Calculate a new estimate of $\mathcal{Q}$ from the current value, the reward and the value of the next state. This can be done when convenient.



3. From the new value of $\mathcal{Q}$ calculate new values for $\boldsymbol{u}$ and $\boldsymbol{q}$ using the wire-fitter partial derivatives. Train the neural network to output the new $\boldsymbol{u}$ and $\boldsymbol{q}$. This can be done when convenient.

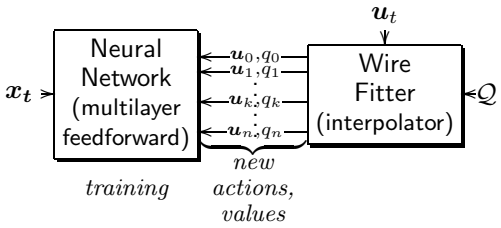


Fig. 3. Wire-fitted neural network training algorithm.

description of the partial derivatives. The partial derivative of $\mathcal{Q}$ from (2) in terms of $q\left(\boldsymbol{x}\right)_k$ is

$$\frac{\partial \mathcal{Q}}{\partial q_k} = \lim_{\epsilon \to 0^+} \frac{norm\left(\boldsymbol{x}, \boldsymbol{u}\right) \cdot \left(distance\left(\boldsymbol{x}, \boldsymbol{u}\right) + q_k \cdot c\right) - wsum\left(\boldsymbol{x}, \boldsymbol{u}\right) \cdot c}{\left[norm\left(\boldsymbol{x}, \boldsymbol{u}\right) \cdot distance\left(\boldsymbol{x}, \boldsymbol{u}\right)\right]^2} \quad . \tag{3}$$

Equation (3) is inexact when $q_k = q_{\max}$. The partial derivative of $\mathcal{Q}$ in terms of $u\left(\boldsymbol{x}\right)_{k,j}$ is

$$\frac{\partial \mathcal{Q}}{\partial u_{k,j}} = \lim_{\epsilon \to 0^+} \frac{\left[wsum\left(\boldsymbol{x}, \boldsymbol{u}\right) - norm\left(\boldsymbol{x}, \boldsymbol{u}\right) \cdot q_k\right] \cdot 2 \cdot \left(u_{k,j} - u_j\right)}{\left[norm\left(\boldsymbol{x}, \boldsymbol{u}\right) \cdot distance\left(\boldsymbol{x}, \boldsymbol{u}\right)\right]^2} \quad , \tag{4}$$

where j selects a term of the action vector (u_j is a term of the chosen action). The summation terms in (3) and (4) have already been found in the calculation of $\mathcal{Q}$ with (2).

With partial derivatives known it is possible to calculate new positions for all the wires $\boldsymbol{u}_{0\ldots n}$ and $q_{0\ldots n}$ by gradient descent. As a result of this change the $\mathcal{Q}$ output from the wire-fitter should move closer to the new target $\mathcal{Q}$.

6 Practical Issues

When the input to a neural network changes slowly a problem known as unlearning or interference can cause the network to unlearn the correct output for other inputs because recent experience dominates the training data [16]. We cope with this problem by storing examples of state, action and next state transitions and replaying them as if they are being re-experienced. This creates a constantly changing input to the neural network, known as a persistent excitation. We do not store target outputs for the network as these would become incorrect through the learning process described in Sect. 5. Instead the wire-fitter is used to calculate new neural network output targets. This method makes efficient use of data gathered from the world without relying on extrapolation. A disadvantage is that if conditions change the stored data could become misleading.

One problem with applying $\mathcal{Q}$-learning to continuous problems is that a single suboptimal action will not prevent a high value action from being carried out at the next time step. Thus the value of actions in a particular state can be very similar, as the value of the action in the next time step will be carried back. As the $\mathcal{Q}$-value is only approximated for continuous states and actions it is likely that most of the approximation power will be used representing the values of the states rather than actions in states. The relative values of actions will be poorly represented, resulting in an unsatisfactory policy. The problem is compounded as the time intervals between control actions get smaller.

Advantage Learning [4] addresses this problem by emphasising the differences in value between the actions. In Advantage Learning the value of the optimal action is the same as for $\mathcal{Q}$-learning, but the lesser value of non-optimal actions is emphasised by a scaling factor ($k \propto \Delta t$). This makes a more efficient use of the approximation resources available. The Advantage Learning update is

$$\begin{aligned}\mathcal{A}(\boldsymbol{x},\boldsymbol{u}) := (1-\alpha)\,\mathcal{A}(\boldsymbol{x},\boldsymbol{u}) \\ + \alpha\left[\tfrac{1}{k}\left(R+\gamma^k \max \mathcal{A}(\boldsymbol{x}_{t+1},\boldsymbol{u}_{t+1})\right) + \left(1-\tfrac{1}{k}\right)\max \mathcal{A}(\boldsymbol{x}_t,\boldsymbol{u}_t)\right] \quad , \end{aligned} \tag{5}$$

where $\mathcal{A}$ is analogous to $\mathcal{Q}$ in (1). The results in Sect. 7 show that Advantage Learning does make a difference in our learning task.

7 Simulation Results

We apply our learning algorithm to a simulation task. The task involves guiding a submersible vehicle to a target position by firing thrusters located on either side. The thrusters produce continuously variable thrust ranging from full forward to full backward. As there are only two thrusters (left, right) but three degrees on freedom (x, y, rotation) the submersible is non-holonomic in its planar world. The simulation includes momentum and friction effects in both angular and linear displacement. The controller must learn to slow the submersible and hold position as it reaches the target. Reward is the negative of the distance to the target (this is not a *pure* delayed reward problem).

Fig. 4. Appearance of the simulator for one run. The submersible gradually learns to control its motion to reach targets

Figure 4 shows a simulation run with hand placed targets. At the point marked zero the learning system does not have any knowledge of the effects of its actuators, the meaning of its sensors, or even the task to be achieved. After some initial wandering the controller gradually learns to guide the submersible directly toward the target and come to a near stop.

In earlier results using $\mathcal{Q}$-learning alone [17], the controller learned to direct the submersible to the first randomly placed target about 70% of the time. Less than half of the controllers could reach all in series of 10 targets. Observation of $\mathcal{Q}$-values showed that the value varied only slightly between actions, making it difficult to learn a stable policy. In our current implementation we use Advantage Learning (see Sect. 6) to emphasise the differences between actions. We now report that 100% of the controllers converge to acceptable performance.

To test this, we placed random targets at a distance of 1 unit, in a random direction, from a simulated submersible robot and allowed a period of 200 time steps for it to approach and hold station on the target. For a series of targets, the average distance over the time period, was recorded. A random motion achieves an average distance of 1 unit (no progress) while a hand coded controller can achieve 0.25. The learning algorithm reduces the average distance with time, eventually approaching hand coded controller performance. Recording distance rather than just ability to reach the target ensures that controllers which fail to hold station don't receive a high rating.

Graphs comparing 140 controllers trained with $\mathcal{Q}$-learning and 140 trained with Advantage Learning are shown in the box-and-whisker plots in Fig. 5. The median distance to the target is the horizontal line in the middle of the box. The upper and lower bounds of the box show where 25% of the data above and below

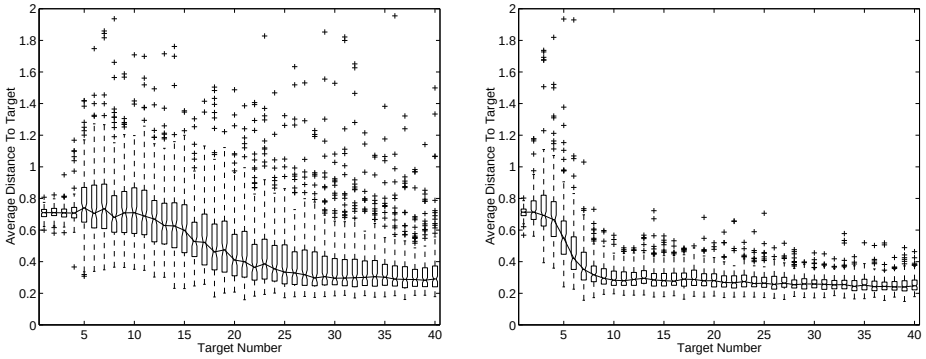


Fig. 5. Performance of 140 learning controllers using $\mathcal{Q}$-learning (*left*) and Advantage Learning (*right*) which attempt to reach 40 targets each placed one distance unit away

the median lie, so the box contains the middle 50% of the data. Outliers, which are outside 1.5 times the range between the upper and lower ends of the box from the median, are shown by a "+" sign. The whiskers show the range of the data, excluding outliers. Advantage Learning converges to good performance more quickly and reliably than $\mathcal{Q}$-learning and with many fewer and smaller magnitude spurious actions. Gradual improvement is still taking place at the 40th target. The quantity of outliers on the graph for $\mathcal{Q}$-learning show that the policy continues to produce erratic behaviour in about 10% of cases.

When reward is based only on distance to the target (as in the experiment above) the actions are somewhat step like. To promote smooth control it is necessary to punish for both energy use and sudden changes in commanded action. Such penalties encouraged smoothness and confirmed that the system is capable of responding to continuous changes in state with continuous changes in action. A side effect of punishing for consuming energy is an improved ability to maintain position.

8 Conclusion

A practical continuous state, continuous action $\mathcal{Q}$-learning system has been described and tested. It was found to converge quickly and reliably on a simulated control task. Advantage Learning was found to be an important tool in overcoming the problem of similarity in value between actions.

Acknowledgements

We thank WindRiver Systems and BEI Systron Donner for their support of the Kambara AUV project. We also thank the Underwater Robotics project team: Samer Abdallah, Terence Betlehem, Wayne Dunston, Ian Fitzgerald, Chris McPherson, Chanop Silpa-Anan and Harley Truong for their contributions.

References

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Bradford Books, MIT, 1998.

[2] Christopher J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.

[3] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans on systems, man and cybernetics*, SMC-13:834–846, 1983.

[4] Mance E. Harmon and Leemon C. Baird. Residual advantage learning applied to a differential game. In *Proceedings of the International Conference on Neural Networks*, Washington D.C, 1995.

[5] Paul J. Werbos. Approximate dynamic programming for real-time control and neural modeling. In D. A. White and D. A. Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*. Van Nostrand Reinhold, 1992.

[6] Juan C. Santamaria, Richard S. Sutton, and Ashwin Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behaviour*, 6(2):163–218, 1998.

[7] Gavin Adrian Rummery. *Problem solving with reinforcement learning*. PhD thesis, Cambridge University, 1995.

[8] Claude F. Touzet. Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems*, 22(3-4):251–81, 1997.

[9] H.-M. Gross, V. Stephan, and M. Krabbes. A neural field approach to topological reinforcement learning in continuous action spaces. In *Proc. 1998 IEEE World Congress on Computational Intelligence, WCCI'98 and International Joint Conference on Neural Networks, IJCNN'98*, Anchorage, Alaska, 1998.

[10] J. S. Albus. A new approach to manipulator control: the cerebrellar model articulated controller (CMAC). *J. Dynamic Systems, Measurement and Control*, 97:220–227, 1975.

[11] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning Journal*, 8(3/4), 1992.

[12] T. Kohonen. *Self-Organization and Associative Memory*. Springer, Berlin, third edition, 1989.

[13] Juan Miguel Santos. *Contribution to the study and design of reinforcement functions*. PhD thesis, Universidad de Buenos Aires, Universite d'Aix-Marseille III, 1999.

[14] Leemon C. Baird and A. Harry Klopf. Reinforcement learning with high-dimensional, continuous actions. Technical Report WL-TR-93-1147, Wright Laboratory, 1993.

[15] Peter Lancaster and Kęstutis Šalkauskas. *Curve and Surface Fitting, an Introduction*. Academic Press, 1986.

[16] W. Baker and J. Farrel. An introduction to connectionist learning control systems. In D. A. White and D. A. Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*. Van Nostrand Reinhold, 1992.

[17] Chris Gaskett, David Wettergreen, and Alexander Zelinsky. Reinforcement learning applied to the control of an autonomous underwater vehicle. In *Proceedings of the Australian Conference on Robotics and Automation (AuCRA99)*, 1999.