Reinforcement Learning applied to the control of an Autonomous Underwater Vehicle

Chris Gaskett **David Wettergreen** **Alexander Zelinsky**
Robotic Systems Laboratory
Department of Systems Engineering, RSISE
The Australian National University
Canberra, ACT 0200 Australia
[cg|dsw|alex]@syseng.anu.edu.au

Abstract

At the Australian National University we are developing an autonomous underwater vehicle for exploration and inspection. Our aim is to develop on-board intelligent control. We intend that the vehicle will learn to control its thrusters in response to command and sensor inputs. Algorithms based on reinforcement learning with continuous state and actions are being developed for this purpose.

Figure 1: Kambara

1 Introduction

Australia's extensive coastline and near-shore waters contain vast biological and mineralogical resources. Only a small fraction of these areas have been explored in detail. Those undersea areas which are developed require continuous monitoring. This motivates us to develop the tools needed for exploration and inspection.

At the Australian National University we developing an autonomous underwater vehicle (AUV) for these tasks [Wettergreen *et al.*, 1998]. We wish to enable submersible robots to autonomously search in regular patterns, follow along fixed natural and artificial features, and swim after dynamic targets. These capabilities are essential to tasks like cataloguing reefs, exploring geologic features, and studying marine creatures, as well as inspecting pipes and cables and assisting divers.

1.1 AUV Control

There have been many approaches to the problem of control of underwater vehicles, ranging from traditional control techniques [Yoerger and Slotine, 1985] to several different artificial network-based control architectures [Lorentz and Yuh, 1996].

Most existing systems control the vehicle in only one or two dimensions, for example yaw and surge, and assume motion along other dimensions can be controlled independently. The implementation of these controllers usually requires a dynamic model of the vehicle. Movement between two points is typically considered a navigation problem, separate to the control problem.

We are developing a system which learns to control an AUV through experience of the real world. No explicit model of the vehicle is given. Our current control approach is a connectionist implementation of model-free reinforcement learning. The controller generates continuous outputs based on continuous state information; this is achieved with the help of an unusual interpolator. The controller learns in response to a scalar reward signal, attempting to maximise the total reward over time. We are currently testing the controller in simulation.

1.2 Kambara

Our AUV is named Kambara, an Australian Aboriginal word for crocodile. Kambara's mechanical structure was designed and fabricated by the University of Sydney. It is a simple, low-cost underwater vehicle suitable as a test-bed for research in underwater robot autonomy. At the Australian National University we have undertaken the task of equipping Kambara with power, electronics, computing and sensing.

Kambara's mechanical structure is an open frame which rigidly supports five thrusters and two watertight enclosures. Kambara's five thrusters enable roll, pitch, yaw, heave, and surge maneuvers. Hence, is is underactuated and not able to perform direct sway (lateral) motion; it is non-holonomic.

Mounted in the upper enclosure are computers, electronics and a sensor package consisting of a triaxial accelerometers, rate gyros, inclinometers and a magnetic heading compass. The lower enclosure, connected to the upper by a flexible coupling, contains batteries as well as depth, temperature and leakage sensors. Kambara carries all the sensors and computers it needs for autonomy. In operation, we envision Kambara will receive only occasional supervisory commands, and control its actions with its on-board resources.

The following section contains a discussion of the control problem and some previous approaches. Section 3 describes our approach to the problem. Section 4 describes some preliminary experimental results gained from simulation.

2 Approaches to AUV Control

In developing a control system for an AUV, the aim is for the vehicle to be able to accurately follow a desired trajectory regardless of the complexities of its own dynamics or the disturbances it experiences.

Traditional approaches to the control of such systems proceed from dynamics modelling, to design of a feedback control law that produces control inputs to compensate for deviation from the desired motion. This is predicated on the assumption that the system can be well-modelled and that specific desired motions can be determined.

Non-traditional, specifically connectionist (artificial neural network), approaches to motion control, can avoid much of the modelling difficulty. Instead, networks are constructed without any model of system dynamics. An appropriate controller is developed through training and the appropriate actions to move the vehicle along the desired path slowly emerge.

2.1 Traditional Control of AUVs

Small, slow-moving underwater vehicles present a particularly challenging control problem. The dynamics of such vehicles are nonlinear because of inertial, buoyancy and hydrodynamic effects [Yoerger and Slotine, 1985]. Linear approximations are insufficient and nonlinear modelling and control techniques are needed to obtain high performance [Fossen, 1995].

Nonlinear models of underwater vehicles have many coefficients, for example those characterising vehicle hydrodynamics, that must be identified. Some model parameters remain unknown either because they are unobservable or because they vary with un-modelled conditions. To date, most stable systems are developed in simulation and only with considerable effort and expense are applied to a specific vehicle with restrictions on its operating regime [Goheen, 1995].

Kambara has five thrusters, none of which are alone capable of moving along or rotating the vehicle through any single dimension. They are unmatched and uncalibrated. Considerable effort has been made in recent years to developing accurate models of thrusters [Yoerger *et al.*, 1990; Bachmayer *et al.*, 1998]. The reason is that thrusters are a dominant source of nonlinearity in vehicle motion.

Kambara has only one plane of symmetry (vertical), although center-of-mass and -buoyancy are not coplanar with the vertical and the motors are slightly mis-aligned. Vehicle asymmetry and thruster interaction during manoeuvring lead to coupling between control parameters which complicates high-performance control.

Yoerger and Slotine proposed a series of single-input/single-output continuous-time controllers by using sliding mode techniques and demonstrated the robustness of these systems in the presence of uncertainties [Yoerger and Slotine, 1985]. Sliding mode techniques enable stable control of the system over a wide operating regime, as required for an AUV. Another advantage is that adaptation can be incorporated to modify the control law as it reaches the limits of its operating regime. Cristi proposed an adaptive sliding mode controller based on a primary linear model and bounds on nonlinear disturbances [Cristi *et al.*, 1990].

Refinements to sliding mode controllers continue to produce one or two-dimensional controllers [Healey and Lienard, 1993; Rodriques *et al.*, 1997; Bartolini *et al.*, 1998]. A general full degrees of freedom solution to control of freely moving underwater vehicle remains elusive.

2.2 Connectionist Control of AUVs

Control using artificial neural networks offers a promising method of designing a nonlinear controller with less reliance on developing accurate models. Controllers implemented as neural networks can be more flexible and are suitable for dealing with multi-variable problems. Many approaches to neuro-control including reinforcement learning are described in [White and Sofge, 1992]. Several different neural network based controllers for AUVs have been developed [Lorentz and Yuh, 1996].

Sanner and Akin [Sanner and Akin, 1990] developed a pitch controller trained by back-propagation. Training of the controller was done off-line in with a fixed system model. Output error at the single output node was estimated by a critic equation based on the pitch error.

Ishii, Fujii and Ura [Ishii *et al.*, 1995] developed a heading controller based on indirect inverse modelling. The model was implemented as a recursive neural network which was trained offline using data acquired by experimentation with the vehicle. The controller was trained on-line. Error at the output of the controller was estimated by propagating through the model to the

single output node which drove the steering thrusters differentially.

Yuh [Lorentz and Yuh, 1996] has developed several neural network based AUV controllers. Error at the output of the controller was based on a critic equation which uses an estimate of the upper bounds of the vehicle inertia matrix to assign error to individual outputs. The controller learned in simulation.

Venugopal [Venugopal *et al.*, 1992] used a similar arrangement to Yuh except that a gain matrix was inserted between the controller and the system model. It reduced the reliance on known parameters of the vehicle but made assumptions about the interactions between various directions of motion [Lorentz and Yuh, 1996].

None of these systems appear to incorporate any planning ability at the controller level. These systems deal with proscribed trajectories and set points, rather than reaching a state through a sequence of steps. They still require at least a partial model of the vehicle.

The resulting performance of these controllers has been promising. The ability to learn or at least refine the controller on-line in real time has been demonstrated [Ishii *et al.*, 1995] as has the ability to cope with changing system parameters [Lorentz and Yuh, 1996].

We are aware of no AUV neurocontrollers that are trained through reinforcement learning.

3 Reinforcement Learning for Control

For Kambara we are developing a model-free reinforcement learning system with multiple continuous states and multiple continuous actions. The lack of an explicit a priori model makes the system adaptable and reduces reliance on knowledge of the system to be controlled.

Our current implementation approach is a connectionist version of Watkins's one step $\mathcal{Q}$-learning algorithm [Watkins, 1989] coupled with Baird and Klopf's wire-fitting interpolation method [Baird and Klopf, 1993].

3.1 Reinforcement Learning

The distinguishing characteristic of reinforcement learning systems is that they receive feedback from a scalar reward which may be temporally distant from the actions carried out.

Reinforcement learning receives feedback from a critic which evaluates the ongoing progress (unlike supervised learning systems in which training requires the correct output). In the case of an AUV, an extended sequence of thruster commands is required to reach a goal. At any instant it is difficult to determine whether individual thrusters are behaving correctly. Only after a period of time can their collective performance be evaluated. The reward follows, often with some delay, an action or sequence of actions. Reward could be based on distance from a target, roll relative from vertical or any other measure of performance. The controller learns to choose actions which, over time, will give the greatest total reward.

The delay before reward leads to the "temporal credit assignment" problem, identifying which parts of a composite action caused the reward is the "structural credit assignment" problem. The scalar reward value on its own does not give enough information to determine what part of a composite action was beneficial, or what part of the state information was important for determining the choice of this action. Thus reinforcement learning systems require time for exploration of the state and action spaces. An introduction to reinforcement learning is given in [Sutton and Barto, 1998].

In many systems a model is used to ease the structural credit assignment problem [White and Sofge, 1992]. These methods could be described as indirect reinforcement learning. This can increase the speed of learning, but are only plausible if a fairly accurate dynamic model is available. In situations where the complexity of finding a model is higher than the complexity of solving the control problem, the direct (model free) method may be more appropriate.

3.2 Continuous States and Actions

Many real world control problems require actions of a continuous nature, in response to continuous state measurements. But most learning systems, indeed most classical AI techniques, are designed to operate in discrete (or symbolic) domains.

It is possible to control small, lightweight robots using discrete actions. These robots have less momentum due to reduced mass and are less easily damaged by step changes in requested motor speeds or oscillations in the requested speed. An example of a successful discrete state and action learning scheme for a small robot is the genetic programming approach described in [Nordin and Banzhaf, 1995].

Fine control of a larger robot (such as Kambara), as needed for station keeping, cannot be carried out with a few coarsely coded outputs. The commands to the motor need to vary smoothly.

Systems in which state and action are discretised scale poorly as the number of state and action variables increases. Accurate control would require each variable to be discretised to many levels.

It is possible to discretise the action in a different way, by having an output which describes whether a variable should be increased, decreased or left as it is. This reduces the number of outputs but increases the need to store state information.

As these discrete systems may fail to generalise between similar actions they require larger quantities of

training data as the number of quantisation levels increases.

A continuous variable should not be considered as equivalent to an infinite number of discrete states, each separated by an infinitely small difference from its neighbour.

3.3 Continuous State $\mathcal{Q}$-learning

$\mathcal{Q}$-learning [Watkins, 1989] is an implementation method for reinforcement learning method in which a mapping is learned from a state-action pair to a value called $\mathcal{Q}$. The mapping eventually represents the reward (in the long run) of performing an action in a state. A controller then measures the state, chooses the action which has the highest $\mathcal{Q}$ value and executes it. The $\mathcal{Q}$ function is updated according to equation 1.

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

Where $\mathcal{Q}$ is the value of performing action $\mathbf{u}$ in state $\mathbf{x}$; $\mathbf{x}$ is the state vector; $\mathbf{u}$ is the action vector; R is the immediate reinforcement; α is a learning rate and γ is the discount factor.

$\mathcal{Q}$-learning (and many other reinforcement learning algorithms) are normally considered in a discrete sense. This allows implementation in a simple lookup table. When states are continuous a possible implementation is to use an artificial neural network (or several artificial neural networks) as an interpolator across states.

By using a neural network for each of the discrete actions, each network can output the $\mathcal{Q}$ value of performing that particular action. An example of this approach is [Lin, 1992].

This approach does not address the problem of continuous actions. If a neural network had action and state as inputs, and $\mathcal{Q}$ as an output, it would be necessary to perform a guided search to find the optimal action, an unattractive prospect. There is also no ability to generalise between similar actions.

3.4 Interpolating Continuous Actions by Wire-fitting

Baird and Klopf's wire-fitting scheme offers a way to implement reinforcement learning with continuous actions [Baird and Klopf, 1993]. The wire-fitting function is a moving least squares interpolator, closely related to Shepard's function [Lancaster and Šalkauskas, 1986]. The modification to Shepard's function is a smoothing factor which increases the influence of the point at which $f(\mathbf{x},\mathbf{u})$ is maximum. Equation 2 describes the wire-fitting function. $f(\mathbf{x},\mathbf{u})$ is the interpolated value of action $\mathbf{u}$ in state $\mathbf{x}$ given the set of points $\mathbf{u}_{0\ldots n}(\mathbf{x}), y_{0\ldots n}(\mathbf{x})$. For $\mathcal{Q}$ learning, $\mathcal{Q} = f(\mathbf{x},\mathbf{u})$.

$$f(\mathbf{x},\mathbf{u}) = \lim_{\epsilon\to 0^+} \frac{\sum_i^n \frac{y_i(\mathbf{x})}{\|\mathbf{u}-\mathbf{u}_i(\mathbf{x})\|^2 + c_i(y_{\max}(\mathbf{x}) - y_i(\mathbf{x})) + \epsilon}}{\sum_i^n \frac{1}{\|\mathbf{u}-\mathbf{u}_i(\mathbf{x})\|^2 + c_i(y_{\max}(\mathbf{x}) - y_i(\mathbf{x})) + \epsilon}} \tag{2}$$

Where i is the wire number, or point number; n is the total number of wires; $\mathbf{x}$ is the state vector; $\mathbf{u}_i(\mathbf{x})$ is the ith action vector; $y_i(\mathbf{x})$ is the value of the ith action vector; $\mathbf{u}$ 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 $\mathbf{u}$ and $\mathbf{u}_i$ is the number of continuous variables in the action. For a five motor AUV the dimensionality of these vectors would be five. The number of wires, n, is currently chosen empirically and reflects the number of fundamental actions.

The wire-fitting function has several properties which make it a useful interpolator for implementing $\mathcal{Q}$-learning. Updates to the $\mathcal{Q}$-function require $\max f(\mathbf{x},\mathbf{u})$ which can be calculated quickly with the wire-fitting function. $\mathbf{u}$ for $\max f(\mathbf{x},\mathbf{u})$ can also be calculated quickly. This is needed when choosing an action to carry out. 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 $\mathcal{Q}$. It does not suffer from oscillations, unlike most polynomial schemes. Importantly, partial derivatives in terms of each y and $\mathbf{u}$ 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.

3.5 Continuous State and Action $\mathcal{Q}$-learning

The purpose of the wire-fitting function in the combined wire-fitting/connectionist approach to $\mathcal{Q}$-learning is to assist in solving the structural credit assignment problem.

Figure 2 shows the structure of the learning system. The state $\mathbf{x}$ is the input to a feed-forward neural network. The output of this network is a number of points $(\mathbf{u}_k, y_k)$ which represent some possible action which could be carried out, and its expected total reward.

When the network is fully trained, the best action to carry out is simply that given by the $\mathbf{u}_k$ with the highest y value. During training the y_i value will not reflect the value of executing the corresponding $\mathbf{u}_i$ action. It is likely than none of the wires will be representing the most beneficial action. The training procedure is shown in algorithm 1.

Partial Derivatives

In the training procedure (algorithm 1) partial derivatives of $\mathcal{Q}$ (the output of the wire-fitting function) for

$$\frac{\partial \mathcal{Q}}{\partial y_k} = \lim_{\epsilon \to 0^+} \frac{\sum_i^n \frac{1}{\|\mathbf{u}-\mathbf{u}_i(\mathbf{x})\|^2 + c_i(y_{\max}(\mathbf{x}) - y_i(\mathbf{x})) + \epsilon} \cdot \left[\|\mathbf{u} - \mathbf{u}_i(\mathbf{x})\|^2 + c_i(y_{\max}(\mathbf{x}) - y_i(\mathbf{x})) + \epsilon + y_k \cdot c\right]}{\left[\sum_i^n \frac{y_i(\mathbf{x})}{\|\mathbf{u}-\mathbf{u}_i(\mathbf{x})\|^2 + c_i(y_{\max}(\mathbf{x}) - y_i(\mathbf{x})) + \epsilon} \cdot \|\mathbf{u} - \mathbf{u}_i(\mathbf{x})\|^2 + c_i(y_{\max}(\mathbf{x}) - y_i(\mathbf{x})) + \epsilon\right]^2} \tag{3}$$

$$\frac{\partial \mathcal{Q}}{\partial u_{k,j}} = \lim_{\epsilon \to 0^+} \frac{\left[\sum_i^n \frac{y_i(\mathbf{x})}{\|\mathbf{u}-\mathbf{u}_i(\mathbf{x})\|^2 + c_i(y_{\max}(\mathbf{x}) - y_i(\mathbf{x})) + \epsilon} - \sum_i^n \frac{1}{\|\mathbf{u}-\mathbf{u}_i(\mathbf{x})\|^2 + c_i(y_{\max}(\mathbf{x}) - y_i(\mathbf{x})) + \epsilon} \cdot y_k\right] \cdot 2 \cdot (u_{k,j} - u_j)}{\left[\sum_i^n \frac{y_i(\mathbf{x})}{\|\mathbf{u}-\mathbf{u}_i(\mathbf{x})\|^2 + c_i(y_{\max}(\mathbf{x}) - y_i(\mathbf{x})) + \epsilon} \cdot \|\mathbf{u} - \mathbf{u}_i(\mathbf{x})\|^2 + c_i(y_{\max}(\mathbf{x}) - y_i(\mathbf{x})) + \epsilon\right]^2} \tag{4}$$

1. Feed the current state ($\mathbf{x}$) into the neural network.
2. Obtain the set of suggested actions and their values ($y_k, \mathbf{u}_k$) from the network output.
3. Choose an action to execute ($\mathbf{u}$), usually by choosing the wire with the highest y value, sometimes by exploring other options.
4. Execute the action and store the next state and a scalar reward from the environment.
5. Calculate the expected value $\mathcal{Q}$ of carrying out the action $\mathbf{u}$ with the wire-fitting function.
6. Calculate a new value for $\mathcal{Q}$ based on the current value of $\mathcal{Q}$, the instantaneous reward R and the expected reward from the next state using the 1 step $\mathcal{Q}$-learning rule (equation 1).
7. Calculate new values for all wires $\mathbf{u}_{0\ldots n}$ and $y_{0\ldots n}$ through the partial derivatives of the wire-fitting function.
8. Train the neural network using back-propagation with the new values of $\mathbf{u}_{0\ldots n}$ and $y_{0\ldots n}$. Then repeat ...

Algorithm 1: Learning system training procedure

all $\mathbf{u}_{0\ldots n}$ and $y_{0\ldots n}$ are needed (Step 7). Baird and Klopf state that the wire-fitting function has partial derivatives but do not describe them [Baird and Klopf, 1993].

Equation 3 is the partial derivative of $\mathcal{Q}$ in terms of $y(\mathbf{x})_k$. This equation is inexact when $y_k = y_{\max}$. In this case the partial derivative can be calculated but it is more complicated and time consuming. This partial derivative in the current implementation is instead calculated by finite differences.

Equation 4 gives the partial derivative of $\mathcal{Q}$ in terms of $u(\mathbf{x})_{k,j}$, where j selects a term of the action vector (u_j is a term of the chosen action). Fortunately, the summation terms in equations 3 and 4 have already been found in the calculation of $\mathcal{Q}$ with equation 2 in step 5.

Change in Wire Values

With partial derivatives known it is possible to calculate new values for all the wires $\mathbf{u}_{0\ldots n}$ and $y_{0\ldots n}$. As a result of this change the $\mathcal{Q}$ output from the wire-fitter should move close to the new target $\mathcal{Q}$.

If it was only possible to change one of the inputs (designated z_k, which could be any y_k or $u_{k,j}$) we would use

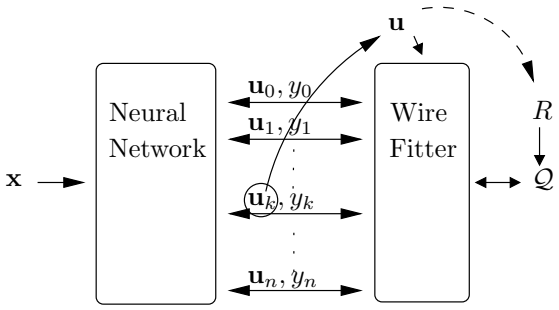

Figure 2: Structure of the learning system

the definition of the derivative in equation 5 to calculate this change according to equation 7.

$$\frac{\partial \mathcal{Q}}{\partial z_k} = \lim_{\Delta \to 0} \frac{\Delta \mathcal{Q}}{\Delta z_k} \tag{5}$$

$$\Delta \mathcal{Q} = \frac{\partial \mathcal{Q}}{\partial z_k} \cdot \Delta z_k \quad \text{for small } \Delta \tag{6}$$

$$\Delta z_k = \frac{\Delta \mathcal{Q}}{\frac{\partial \mathcal{Q}}{\partial z_k}} \quad \text{for small } \Delta \tag{7}$$

However, as there are multiple inputs to the wire-fitter that can be altered, a rule must be devised to change these inputs by the minimum amount to change $\mathcal{Q}$ to the required value. It is desirable to obtain most of the change in $\mathcal{Q}$ by altering the inputs with the highest magnitude partial derivatives, resulting in a rule given by equation 8.

$$\Delta z_k = a(\mathbf{z}) \cdot \frac{\partial \mathcal{Q}}{\partial z_k} \cdot \Delta \mathcal{Q} \tag{8}$$

Scaling factor a must vary to change $\mathcal{Q}$ without over or under correcting to the target $\mathcal{Q}$ value. By combining equations 5,6 and 8 it is possible to solve for a, giving the solution shown in equation 9.

$$a = \frac{1}{\sum_i \left(\frac{\partial \mathcal{Q}}{\partial z_i}\right)^2} \tag{9}$$

Combining equations 8 and 9 yields equation 10, the final update rule.

$$\Delta z_k = \frac{1}{\sum_i \left(\frac{\partial \mathcal{Q}}{\partial z_i}\right)^2} \cdot \frac{\partial \mathcal{Q}}{\partial z_k} \cdot \Delta \mathcal{Q} \tag{10}$$

Empirical testing of this rule indicates that the change in $\mathcal{Q}$ is very close to the desired change in $\mathcal{Q}$.

4 Experimental Results

This learning method is currently being tested in a simulated two dimensional environment in which actions occur accurately and observably. Soon it will be implemented on Kambara to learn from scratch in the water.

4.1 Simulation Description

A simulated non-holonomic, two-dimensional AUV has been devised with motors on the left and right sides. The simulation includes momentum, angular momentum and frictional effects. Sensors give the location of the target in body coordinates as well as linear and angular velocity. Reward is given after after each time step based on the change in distance to the target. Moving toward the target provokes a positive reward, conversely moving away gives negative reward. A reward function which gives more guidance, for example incorporating angle to the target, should improve learning speed.

An image of the simulation is shown in figure 3.

4.2 Evaluation of performance

In repeated trials, simulated AUVs reach their first (random) target location (within small bounds) about 70% of the time. If more that 2000 time steps are taken without reaching the target this is regarded as a failure. The AUVs initially take a very round about route, moving off in the wrong direction and spiralling around. If the first target is reached, a second target is generated. This target is reached by 88% of AUVs which reach the first target with fewer steps taken.

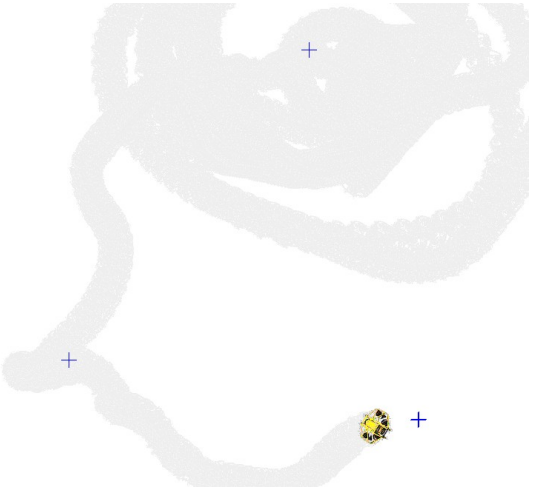

Figure 3: An image from the simulation showing the AUV approaching its third target

Figure 4 shows the percentage of AUVs reaching the n^{th} target having reached the $(n-1)^{th}$ target and the absolute percentage of AUVs reaching the n^{th} target. Figure 5 shows the number of steps taken for successful AUVs to reach the n^{th} target. These results from 300 simulation runs show that the performance of the system is improving with time.

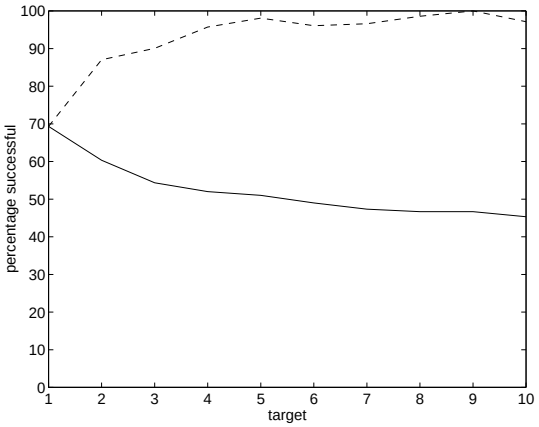

Figure 4: Dashed line is percentage of simulated AUVs reaching target n after reaching target $n-1$, solid line is percentage of AUVs reaching target n

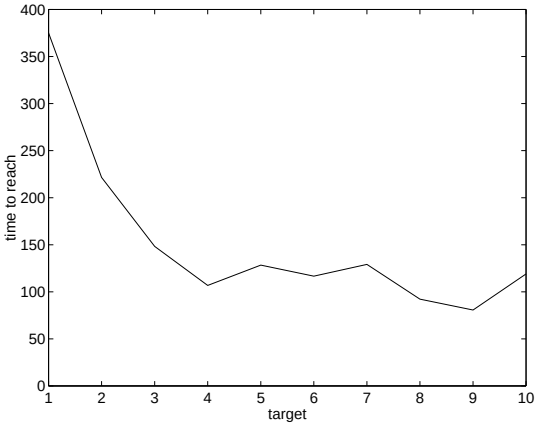

Figure 5: Number of steps taken to reach target n after reaching target $n-1$

5 Conclusion

There are many unresolved issues still be addressed in implementing this control method on-board Kambara.

Further experiments are needed to identify the best neural network structure and learning parameters. Also we will need to scale up to the full degrees of freedom and test the robustness of this approach in the real world. The promising first result is that with reinforcement learning and without any model of system dynamics we can develop a controller that will autonomously guide an underwater vehicle to its target.

Acknowledgements

We wish to thank the members of the Robotic Systems Laboratory for their support and encouragement and WindRiver Systems for the contribution of VxWorks, Kambara's real-time operating system.

References

[Bachmayer *et al.*, 1998] R. Bachmayer, L. Whitcomb, and M. Grosenbach. Four quadrant finite dimensional thruster model. In *Proceedings of OCEANS98*. IEEE, 1998.

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

[Bartolini *et al.*, 1998] G. Bartolini, E. Punta, and E. Usai. Tracking control of underwater vehicles including thruster dynamics by second order sliding modes. In *Proceedings of OCEANS98*. IEEE, 1998.

[Cristi *et al.*, 1990] R. Cristi, F. Papoulias, and A. Healey. Adaptive sliding mode control of autonomous underwater vehicles in the dive plane. *IEEE Journal of Oceanic Engineering*, 15(3):152–159, July 1990.

[Fossen, 1995] Thor I. Fossen. Underwater vehicle dynamics. In J. Yuh, editor, *Underwater Robotic Vehicles: Design and Control*. TSI Press, 1995.

[Goheen, 1995] Kevin R. Goheen. Techniques for URV modeling. In J. Yuh, editor, *Underwater Robotic Vehicles: Design and Control*. TSI Press, 1995.

[Healey and Lienard, 1993] A. Healey and D. Lienard. Multivariable sliding mode control for autonomous diving and steering of unmanned underwater vehicles. *IEEE Journal of Oceanic Engineering*, 18(3):327–338, July 1993.

[Ishii *et al.*, 1995] Kazuo Ishii, Teruo Fujii, and Tamaki Ura. An on-line adaption method in a neural network based control system for AUV's. *IEEE Journal of Oceanic Engineering*, 20(3), July 1995.

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

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

[Lorentz and Yuh, 1996] Jørgen Lorentz and J. Yuh. A survey and experimental study of neural network AUV control. In *Proceedings of the Symposium on Autonomous Underwater Vehicle Technology*. IEEE, 1996.

[Nordin and Banzhaf, 1995] Peter Nordin and Wolfgang Banzhaf. Genetic programming controlling a miniature robot. In E. V. Siegel and J. R. Koza, editors, *Working Notes for the AAAI Symposium on Genetic Programming*, pages 61–67, MIT, Cambridge, MA, USA, 1995. AAAI.

[Rodriques *et al.*, 1997] L. Rodriques, P. Tavares, and M. Prado. Sliding mode control of an AUV in diving and steering planes. In *Proceedings of OCEANS97*, pages 576–583. IEEE, 1997.

[Sanner and Akin, 1990] R. M. Sanner and D. L. Akin. Neuromorphic pitch attitude regulation of an underwater telerobot. *IEEE Control Systems Magazine*, April 1990.

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

[Venugopal *et al.*, 1992] K. P. Venugopal, R. Sudhakar, and A. S. Pandya. On-line learning control of autonomous underwater vehicles using feedforward neural networks. *IEEE Journal of Oceanic Engineering*, 17(4):308–318, October 1992.

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

[Wettergreen *et al.*, 1998] David Wettergreen, Chris Gaskett, and Alexander Zelinsky. Development of a visually-guided autonomous underwater vehicle. In *Proceedings of OCEANS98*. IEEE, 1998.

[White and Sofge, 1992] D. A. White and D. A. Sofge, editors. *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*. Van Nostrand Reinhold, 1992.

[Yoerger and Slotine, 1985] D. Yoerger and J-J. Slotine. Robust trajectory control of underwater vehicles. *IEEE Journal of Oceanic Engineering*, OE-10(4):462–470, October 1985.

[Yoerger *et al.*, 1990] D. Yoerger, J. Cooke, and J-J. Slotine. The influence of thruster dynamics on underwater vehicle behavior and their incorporation into control system design. *IEEE Journal of Oceanic Engineering*, 15(3):167–178, July 1990.