Reinforcement Learning Methods for Multi-Linked Manipulator Obstacle Avoidance and Control

Chen K. Tham, Richard W. Prager *
Cambridge University Engineering Department,
Trumpington Street, Cambridge CB2 1PZ, UK.
e-mail : ckt@eng.cam.ac.uk, rwp@eng.cam.ac.uk

June 3, 1993

Abstract

This paper treats the multi-linked manipulator obstacle avoidance and control task as the interaction between a learning agent and an unknown environment. The role of the agent is to generate actions that maximises the reward that it receives from the environment, i.e. when the goal of reaching the destination without collision with obstacles is achieved. We demonstrate how two learning algorithms common in reinforcement learning literature: Adaptive Heuristic Critic (AHC) (Barto et al., 1983) and Q-Learning (Watkins, 1989), can be used to solve the task successfully in two different ways: firstly, through the generation of position commands to a PD controller which produces torque commands to drive the manipulator, and secondly, through the direct generation of torque commands, removing the need for a PD controller. During the process, the inverse kinematics problem for multi-linked manipulators is automatically solved. Fast function approximation is achieved through the use of an array of Cerebellar Model Arithmetic Computers (CMAC) (Albus, 1975). The generation of both discrete and continuous actions are investigated and the performance of the algorithms in terms of learning rates, efficiency of solutions, and memory requirements are evaluated.
Keywords: Reinforcement Learning; Machine Learning; Adaptive Systems; Neural Control

1 Introduction

We describe methods to handle the task of multi-linked manipulator obstacle avoidance and control without a known dynamical or world model. A simulated robot manipulator with two links is used and the objective is to move from a start point to some desired destination in the workspace. A reinforcement signal which indicates when a collision with an obstacle or joint limit has occurred or when the destination has been reached is required. Joint positions and velocities are used as inputs to the system. The appropriate actions for each joint are obtained which not only enables the manipulator to reach the destination, but also avoid collisions with obstacles in the workspace. This a non-trivial task involving large input and action spaces. (see Figure 1)

The performance of two reinforcement learning algorithms for solving this task are compared. The first set of experiments described in section 3 uses the Adaptive Heuristic Critic (AHC) (Barto et al., 1983) algorithm together with Williams' stochastic hillclimbing algorithm (Williams, 1988) in order to produce real-valued actions. In the remaining experiments, the Q-Learning algorithm and a discrete action space are used. We evaluate how well these algorithms perform in two methods of controlling the manipulator: (1) through position commands to a PD controller, and (2) through direct torque commands.

In the following discussion, *robot* refers to the two-linked manipulator under consideration; *agent*, the controller which has to learn the correct actions to take; and *state* $\boldsymbol{x}$, the angular positions and velocities of the joints, i.e. $\boldsymbol{x} = [\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2]$.

*This paper was presented at the IEEE Asia-Pacific Workshop on Advances in Motion Control, Singapore, July 15-16, 1993.

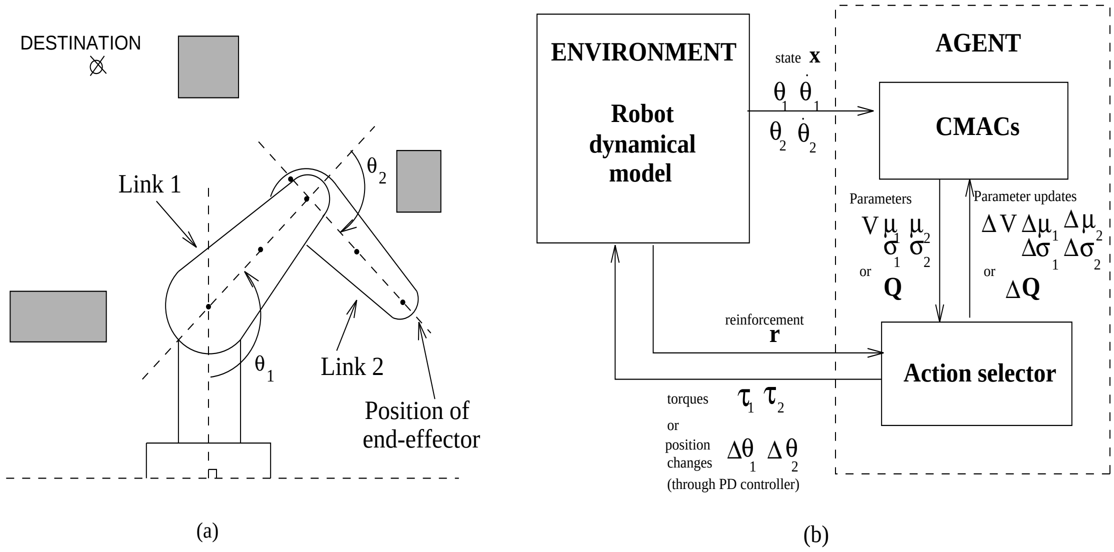


Figure 1: (a) Robot manipulator with obstacles in the workspace. (b) Interaction between the agent and environment.

2 Reinforcement Learning

Reinforcement learning is a direct learning method where the performance of a learning agent is judged on the basis of a single scalar signal, the *reinforcement* r received from the environment. The objective is to determine the appropriate *policy* (which specifies the agent's actions in each state) in order to maximise the *return*, the expected long term discounted sum of reinforcement.

2.1 Adaptive Heuristic Critic (AHC) and Stochastic Hillclimbing

Adaptive Heuristic Critic

The learning procedure is an on-line method in which the agent seeks to learn the *evaluation function* $V(\boldsymbol{x})$ of a policy and adjusts the policy for improved performance in every time step (Barto et al., 1989). The evaluation function gives a prediction of return at any state.

The TD (temporal difference) error[1] is obtained by :

$$\varepsilon_{t+1} = r_{t+1} + \gamma V_t(\boldsymbol{x}_{t+1}) - V_t(\boldsymbol{x}_t) \tag{1}$$

where $\boldsymbol{x}_t$ is the state of the system at time t and γ is the *discount factor*.

The evaluation function, approximated by a Cerebellar Model Arithmetic Computer (CMAC) (Albus, 1975), is updated according to :

$$V_{t+1}(\boldsymbol{x}) = V_t(\boldsymbol{x}) + \alpha \varepsilon_{t+1} c_t(\boldsymbol{x}) \tag{2}$$

$$\text{where} \quad \begin{array}{rcll} c_t(\boldsymbol{x}) & = & 1 + \lambda c_{t-1}(\boldsymbol{x}) & \text{if } \boldsymbol{x} = \boldsymbol{x}_t \\ c_t(\boldsymbol{x}) & = & \lambda c_{t-1}(\boldsymbol{x}) & \text{if } \boldsymbol{x} \neq \boldsymbol{x}_t \end{array}$$

α is the learning rate parameter and $c_t(\boldsymbol{x})$, an *eligibility trace* affecting the states through which the system passed in the time steps preceding t. This eligibility trace, which decays at a rate dependent on the parameter λ, accelerates the learning process as shown by Sutton (Sutton, 1984).

[1]The TD error is also known as the *heuristic reinforcement signal*.

Stochastic Hillclimbing

In order to achieve real-valued outputs, actions are determined stochastically by *associative stochastic learning automata* (Williams, 1988). Each action a is chosen from a Gaussian probability distribution with two parameters, the mean μ and standard deviation σ, which are approximated as functions of the inputs by CMACs.

Williams proposed a method to update the parameters μ and σ in the direction of increasing expected return $E(r \mid \boldsymbol{x})$ in a stochastic hillclimbing manner :

$$\begin{array}{rrcll} & \Delta\mu & = & \beta(r-b_\mu)e^{(\mu)} & \text{where } e^{(\mu)} = \frac{\partial \ln g}{\partial \mu} = \frac{(a-\mu)}{\sigma^2} \\ \text{and} & \Delta\sigma & = & \beta(r-b_\sigma)e^{(\sigma)} & \text{where } e^{(\sigma)} = \frac{\partial \ln g}{\partial \sigma} = \frac{(a-\mu)^2-\sigma^2}{\sigma^3} \end{array} \tag{3}$$

The terms $e^{(\mu)}$ and $e^{(\sigma)}$ are the *characteristic eligibilities* of parameter updates; $(r-b_\mu)$ and $(r-b_\sigma)$ are the *reinforcement offsets* indicating how good the last action was with respect to *reinforcement baselines* b_μ and b_σ. The TD error ε_{t+1}, which provides a utility measure of the last action taken with respect to the expected return in the previous state (Barto et al., 1989), can be used as the reinforcement offset. As in Equation 2, we employ eligibility traces $s_t^{(\mu)}$ and $s_t^{(\sigma)}$ when updating μ and σ respectively.

Since the operations involved in updating μ and σ are the same, let $\boldsymbol{p} = [\mu, \sigma]$, $\boldsymbol{e} = [e^{(\mu)}, e^{(\sigma)}]$ and $\boldsymbol{s} = [s^{(\mu)}, s^{(\sigma)}]$. Writing $\boldsymbol{p}_t$, $\boldsymbol{e}_t$ and $\boldsymbol{s}_t$ for these terms evaluated at time t, the policy update rule is :

$$\boldsymbol{p}_{t+1}(\boldsymbol{x}) = \boldsymbol{p}_t(\boldsymbol{x}) + \beta\varepsilon_{t+1}\boldsymbol{s}_t(\boldsymbol{x}) \tag{4}$$

$$\begin{array}{rrcll} \text{where} & \boldsymbol{s}_t(\boldsymbol{x}) & = & \boldsymbol{e}_t(\boldsymbol{x}) + \lambda\boldsymbol{s}_{t-1}(\boldsymbol{x}) & \text{if } \boldsymbol{x} = \boldsymbol{x}_t \\ & \boldsymbol{s}_t(\boldsymbol{x}) & = & \lambda\boldsymbol{s}_{t-1}(\boldsymbol{x}) & \text{if } \boldsymbol{x} \neq \boldsymbol{x}_t \end{array}$$

This update rule allows μ to increase towards a if $a > \mu$ and the expected return from taking action a is better than the average action μ, as indicated by positive ε_{t+1} (and vice versa). The standard deviation σ is decreased if $|a-\mu| < \sigma$ and ε_{t+1} is positive, leading to convergence towards a locally optimum action. The opposite case causes σ to increase and allows more exploratory behaviour.

2.2 Q-Learning

In Q-Learning (Watkins, 1989), the *state-action value* $Q(\boldsymbol{x}, a)$, which is the return in state $\boldsymbol{x}$ when action a is performed, is estimated. The action space is discrete and a separate $Q(\boldsymbol{x}, a)$ exists for each action a. Each time the agent takes an action a from state x at time t, the current state-action value estimate for $\boldsymbol{x}$ and a denoted $\hat{Q}_t(\boldsymbol{x}, a)$ is updated as follows :

$$\hat{Q}_{t+1}(\boldsymbol{x}, a) = (1-\beta)\hat{Q}_t(\boldsymbol{x}, a) + \beta[r(\boldsymbol{x}, \boldsymbol{y}, a) + \gamma \max_{l \in A} \hat{Q}_t(\boldsymbol{y}, l)] \tag{5}$$

where $\boldsymbol{y}$ is the actual next state, γ is the discount factor, β is a step-size parameter, A is the set of possible actions and $r(\boldsymbol{x}, \boldsymbol{y}, a)$ is the reinforcement that is received in the transition from state $\boldsymbol{x}$ to state $\boldsymbol{y}$ by taking action a. As in Equations 2 and 4, the state-action value estimates for the preceding state-action pairs were also updated by amounts determined by the decaying eligibility trace (not shown in Equation 5 for clarity). The state-action value estimates for other states and actions remain unchanged.

During learning, the agent has to explore by performing different actions in order to discover the best one. As in Sutton (Sutton, 1990), the method we have used is to select actions according to the Boltzmann distribution

$$P(a \mid \boldsymbol{x}) = e^{\hat{Q}(\boldsymbol{x}, a)} / \sum_l e^{\hat{Q}(\boldsymbol{x}, l)} \text{ for all } l \in A \tag{6}$$

This allows the agent to bias its policy toward the estimated optimal actions while permitting sufficient exploration.

Unlike the AHC approach which has separate update rules for the evaluation function $V(\boldsymbol{x})$ and the policy $\boldsymbol{p}(\boldsymbol{x})$, Q-Learning requires only one update rule for the state-action values $Q(\boldsymbol{x}, a)$. However, this does not imply reduced computation as the maximum $Q(\boldsymbol{x}, a)$ across actions has to be found and the action probabilities calculated for state $\boldsymbol{x}$ before an action is selected.

Since $Q(\boldsymbol{x}, a)$ is stored for all actions in each state, memory requirements in Q-Learning are greater than that of AHC if the action space is large. In AHC, the policy $\boldsymbol{p}(\boldsymbol{x})$ has to be stored whereas in Q-Learning, the policy is deduced from the Q-values.

3 Experiment Details

Four experiments were carried out :

AHC-POS: the AHC-Williams algorithms are used and the outputs are real-valued position change commands.

AHC-TOR: the AHC-Williams algorithms are used and the outputs are real-valued torque commands.

Q-POS: the Q-Learning algorithm is used and the outputs are discrete position change commands.

Q-TOR: the Q-Learning algorithm is used and the outputs are discrete torque commands.

In the AHC-POS and AHC-TOR cases, there was no need to pre-determine the range of action values for the agent; the action values were initialized to zero at the beginning and converged to the appropriate actions in each state. In the Q-POS and Q-TOR cases, the actions are discrete and the magnitudes of position changes and torques corresponding to each action are set to reasonable values for the task at hand. For Q-POS, the actions are { -0.4, -0.2, -0.1, -0.05, -0.025, 0.0, 0.025, 0.05, 0.1, 0.2, 0.4 } and for Q-TOR, the actions are { -10.0, -5.0, -2.0, -1.0, -0.5, 0.0, 0.5, 1.0, 2.0, 5.0, 10.0 }, in units of rads and Nm respectively. In our experiments, Q-POS and Q-TOR used an array of 22 CMACs (one for each action in each joint) while AHC-POS and AHC-TOR used 5 CMACs (one for each of $V, \mu_1, \sigma_1, \mu_2, \sigma_2$).

The reinforcement r is awarded according to :

1. If the destination is reached, $r = 0.5 + 0.5e^{-(|\dot{\theta}_1|+|\dot{\theta}_2|)}$. This gives a higher than normal reinforcement if the destination is approached with low velocities.
2. If a collision occurs, $r = -0.025 \times |\dot{\theta}|$ of the link involved, subject to the constraint that the maximum negative reinforcement is $r = -0.30$.
3. If either $|\dot{\theta}_1|$ or $|\dot{\theta}_2|$ is greater than 10.0 rad/s, $r = -0.05$. This is to indicate that excessively high velocities are undesirable.

The following were the same for all four experiments: the dynamical model of the robot, the reinforcement schedule and the CMAC structure. The value of γ in Equations 1 and 5 was 0.95 and the TD(λ) method (Sutton, 1984) of using an eligibility trace in parameter updates for temporal credit assignment was employed in all cases. With $\lambda = 0.5$, the eligibility traces decay rapidly; thus, we only had to maintain a queue of 5 previous states (and actions for Q-POS and Q-TOR) and update the function values of these states or state-action pairs. For the AHC-POS and Q-POS cases, the PD controller used had the form:

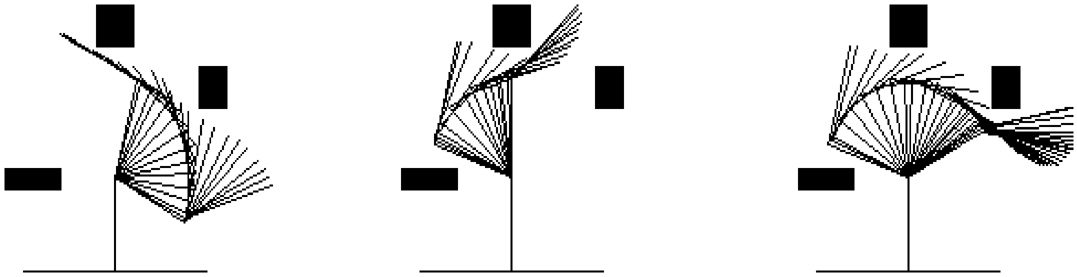
$$\tau = k_1 \times \Delta\theta - k_2 \times \dot{\theta} \tag{7}$$

where $k_1 = 100.0$ and $k_2 = 3.0$. Parameters β and σ_{start}, however, had different values in the experiments. These are summarized in Table 1.

Experiment	γ	λ	α	β	μ_{start}	σ_{start}	Memory req'mt
AHC-POS	0.95	0.5	0.1	0.005	0.0 rad[a]	0.1 rad	672K
AHC-TOR	0.95	0.5	0.1	0.5	0.0 Nm[a]	10.0 Nm	668K
Q-POS	0.95	0.5	–	0.5	random[b]	n/a	2048K
Q-TOR	0.95	0.5	–	0.5	random[b]	n/a	2044K

Table 1: Parameter values and memory requirements during execution in the 4 experiments. *Notes*: (*a*) actions are selected according to the Gaussian distribution; (*b*) actions are selected according to the Boltzmann distribution across possible actions.

Training was conducted in cycles of 5000 trials where each trial consists of at most 100 steps. If the destination is reached or a collision occurs, the current trial ends and the next one begins with the robot starting at a random position in the workspace.

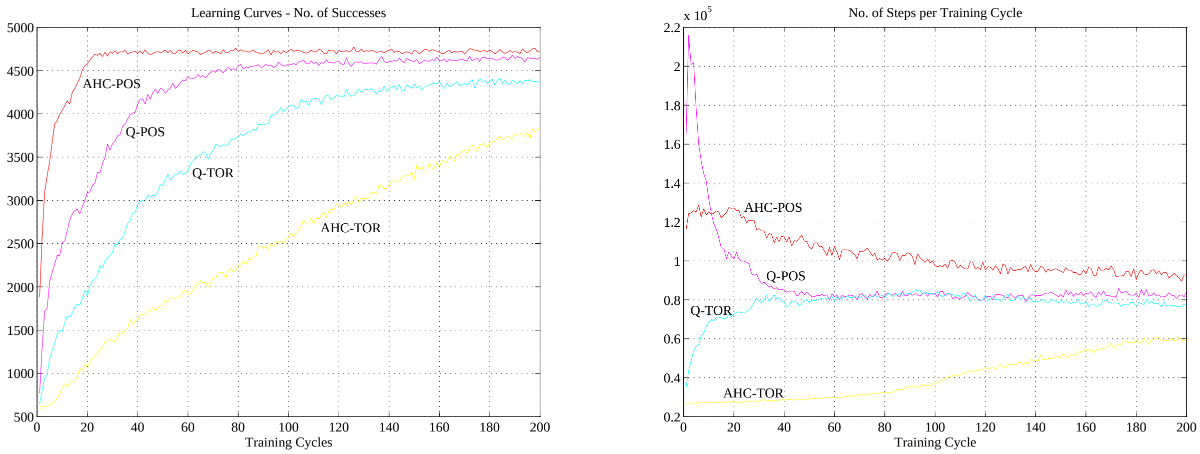


(a) (b) (c)

Figure 2: Trajectories followed by the robot from different start positions to the destination.

4 Results

Figure 2 shows that the robot is able to reach the destination from different start positions in the workspace. The motion is smooth and the robot is able to back out of tight situations before moving towards the destination.



(a) (b)

Figure 3: (a) Performance improvement with training. (b) Policy optimization with training.

Figure 3(a) shows how the performance of the agent improved with training in terms of the increasing number of successes in each cycle. For the four implementations, the agent required different amounts of training before reaching its best level of performance. AHC-POS learnt fastest, followed by Q-POS and Q-TOR. AHC-TOR was the slowest.

The final performance figures for the four experiments are summarized in Table 2. AHC-TOR reached its best level of performance after 443 training cycles (not shown in Figure 3(a)) and both were better than their Q-Learning counterparts. This is due to the more precise control actions possible with real-valued outputs. The best performance achieved are in AHC-POS, AHC-TOR, Q-POS and Q-TOR order.

In all four cases, there are two distinct phases during learning as seen in Figure 3(b). In the beginning, the number of steps taken per training cycle increases as the agent discovers how to avoid collisions. This is then followed by a policy optimization phase where the agent attempts to perform actions to reach the destination in as few time steps as possible. In the end, the efficiency of the learnt policies are in AHC-TOR, Q-TOR, Q-POS and AHC-POS order.

The evaluation function $V(\boldsymbol{x})$ learnt by AHC-TOR is shown in Figure 4. The AHC-POS evaluation function had the same shape although the policy learnt was different. There are two possible robot configurations in which the end-effector reaches the destination, i.e. two solutions

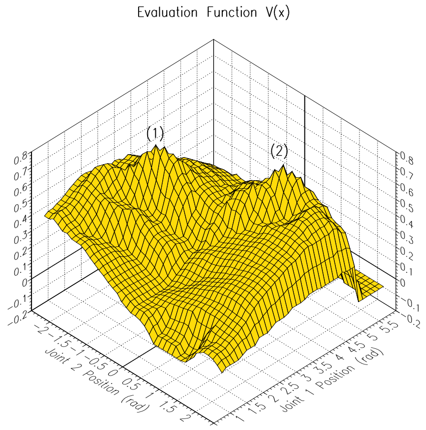


Figure 4: Evaluation function surface at $\dot{\theta}_1 = 0.0$rad/s, $\dot{\theta}_2 = 0.0$rad/s.

Experiment	Final # Success	Final Average # Steps/Trial
AHC-POS	4720	17.99
AHC-TOR	4701	12.72
Q-POS	4650	16.56
Q-TOR	4416	15.41

Table 2: Summary of final performance figures.

to the inverse kinematics. These correspond to two peaks in the evaluation function surface as indicated by (1) and (2) in the graph. In the Q-Learning experiments, 22 different Q-functions were learnt which gave the Q-value for each of the possible actions in every state.

5 Conclusion

We have demonstrated how reinforcement learning can be used in an integrated method for obstacle avoidance and control of a multi-linked manipulator. Our results indicate that the AHC-Williams and Q-Learning algorithms are effective in learning how to solve a complicated task involving large input and action spaces. The algorithms required different amounts of training and achieved different levels of performance. We found that the AHC-TOR approach required a lot more training than the other methods, but converged to the best policy in the end.

References

Albus, J. S. (1975). A new approach to manipulator control : The cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement and Control*, 97(3):220–227.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-13(5):835–846.

Barto, A. G., Sutton, R. S., and Watkins, C. J. C. H. (1989). Learning and sequential decision making. Technical Report COINS 89–95, Dept. of Computer and Information Science, University of Massachusetts, Amherst.

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, Amherst, MA.

Sutton, R. S. (1990). Integrated architectures for learning, planning and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, Cambridge, UK.

Williams, R. J. (1988). Toward a theory of reinforcement-learning connectionist systems. Technical Report NU-CCS-88-3, College of Computer Science, Northeastern University, Boston, MA.