Reinforcement Learning for Multi-Linked Manipulator 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 24, 1992

Technical Report
CUED/F-INFENG/TR 104

Abstract

We present an automatic trajectory planning and obstacle avoidance method for a multi-linked manipulator which uses position and velocity sensor information directly to produce the appropriate real-valued torques for each joint. The inputs are fed into a Cerebellar Model Arithmetic Computer (CMAC) [1] and in each state, the expected reward and torques for each joint are learnt through self-experimentation using a combination of the Temporal Difference (TD) technique [9] and stochastic hillclimbing [14]. Actions which cause the manipulator to reach the desired destination are rewarded whereas actions which lead to collisions with either joint limits or obstacles are punished by an amount proportional to the velocity before collision. After training, the manipulator is able to move along collision free paths from different start positions in the workspace to the destination.

Keywords: Reinforcement Learning; Machine Learning; Robotics; Connectionist Models

1 Introduction

The task of controlling a multi-linked manipulator can be broadly divided into three parts: path planning; coordinate transformation from task-oriented coordinates to body coordinates [1] as in inverse kinematics; and the generation of motor commands. Path planning involves finding the best path from a start position to the destination and usually obstacle avoidance, while the generation of motor commands involves trajectory tracking, i.e. applying the appropriate torques at each joint in order to follow some desired position and velocity profile. An accurate dynamical model of the manipulator is normally required.

This report proposes a method that handles these tasks without using a dynamical model. A simulation of a robot manipulator with two links is used to demonstrate the

*The authors thank Eli Tzirkel-Hancock, Tim Jervis and Chris Dance for their comments and help during the course of this work. Chen K. Tham is supported by a scholarship from the National University of Singapore.

[1] Task-oriented and body coordinates are sometimes referred to as distal and proximal coordinates, respectively.

ability of the learning controller in moving the end-effector of the manipulator from a start point to some desired destination. 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 position and velocity information from sensors are used as inputs to the system in our experiments, but the method can be modified to accept inputs from a range sensor or a vision system. The appropriate torques to each joint are obtained which not only enables the manipulator to reach its destination, but also avoid collisions with obstacles in the workspace.

This report expands on the findings described in [11] and is organized as follows. Related work are described briefly in Section 2. Section 3 describes the simulation of the robot and the task in greater detail. Section 4 contains details about the implementation of CMACs in the experiments. Section 5 provides an overview of reinforcement learning and Section 6 describes the theory of associative stochastic learning automata – the learning algorithm we employed is based on the theory explained in these two sections. Section 7 presents the reinforcement schedule used to facilitate learning of useful behaviours. Section 8 contains experimental results and a detailed study of two motions of the robot. Section 9 discusses several issues related to this work and finally, the conclusion is in Section 10. An appendix showing the derivation of the robot dynamics simulation model can be found at the end of the report.

2 Background

Barto, Sutton and Anderson [2] employed reinforcement learning in the control of a dynamical system in the form of a pole on a cart. The pole was free to move in a vertical plane and the cart horizontally in left and right directions. The objective was to balance the pole by applying a horizontal force to the cart. A reinforcement signal indicated when the pole had fallen over and learning was aided by the use of an Adaptive Critic Element (ACE) which tried to predict the reinforcement signal.

Nguyen and Widrow [7] employed two connectionist networks in the control of a simulated trailer truck backing into a loading dock, one to represent the kinematic model of the trailer truck and another the controller which had to generate the appropriate steering signal at different truck positions. This is a highly non-linear control problem and the controller was able to perform the task successfully even when the cab and trailer were initially 'jack-knifed'.

Mahadevan and Connell [4] described a subsumption approach for automatically programming a behaviour-based robot using reinforcement learning in a box-pushing task. The robot acquired several behaviours: (1) finding a box, (2) pushing a box across a room, and (3) recovering from stalled situations. Only one behaviour was active at any given time, selected by applicability functions and a priority-ordering scheme. These behaviours enabled the robot to find and push boxes around a room successfully.

More recently, Prescott and Mayhew [8] used reinforcement learning to enable a simulated mobile robot to acquire reflexive obstacle avoidance behaviour. The input to the mobile robot was from a primitive range sensor and after training, forward and angular velocities were obtained which caused it to move at an optimal speed along collision free paths in an environment containing obstacles.

The task we address here contains elements of the above. As in the truck backer-upper, the robot manipulator is a non-linear system; however, it is not necessary to have a two-stage learning process in our method since an emulator of the robot dynamics is

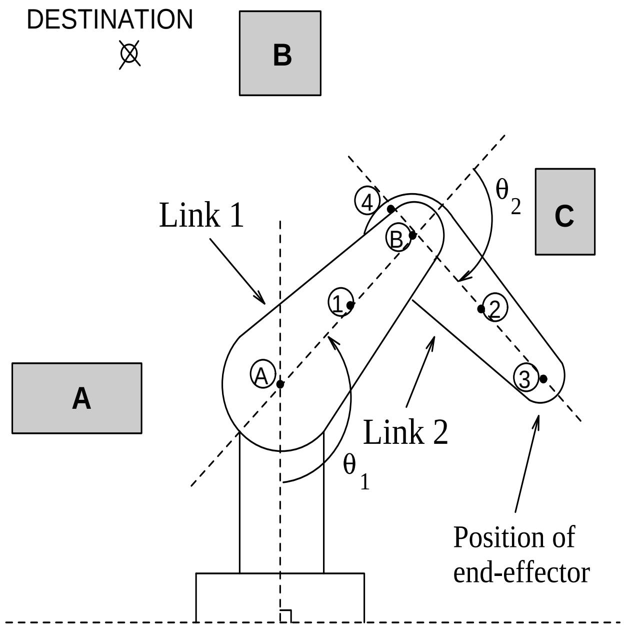

Figure 1: Robot manipulator with obstacles in the workspace.

not required. We also seek to control the manipulator through real-valued torques in a dynamical formulation of the problem.

In the discussion that follows, *robot* refers to the two-linked manipulator under consideration, shown in Figure 1; *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. $\boldsymbol{x} = [\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2]$. Figure 2 provides an overview of the proposed method. The various components and symbols will be explained in subsequent sections.

3 The Robot Simulation

The robot simulation is based on an actual robot and has two revolute joints, each free to rotate in the same plane through an angle of $\frac{6\pi}{4}$rads, i.e. $\frac{\pi}{4} < \theta_1 < \frac{7\pi}{4}$rads and $-\frac{3\pi}{4} < \theta_2 < \frac{3\pi}{4}$rads, where θ_1 and θ_2 are as shown in Figure 1. The equations of motion are derived from the Lagrangian in terms of the potential and kinetic energies. (see Appendix)

The objective is to get the end-effector at the end of link 2 to the destination from an arbitrary starting position in the workspace. A collision occurs when any part of the links hits an obstacle or when the robot attempts to move outside the allowed ranges of θ_1 and θ_2.

4 Cerebellar Model Arithmetic Computer (CMAC)

The joint angular positions and velocities are first fed into CMACs [1] which are used to represent several functions of the inputs described in later sections.

The CMAC is a coarse-coding structure where each region in the input space has a set of overlapping but offset tiles associated with it. Every tile is defined by quantizing functions

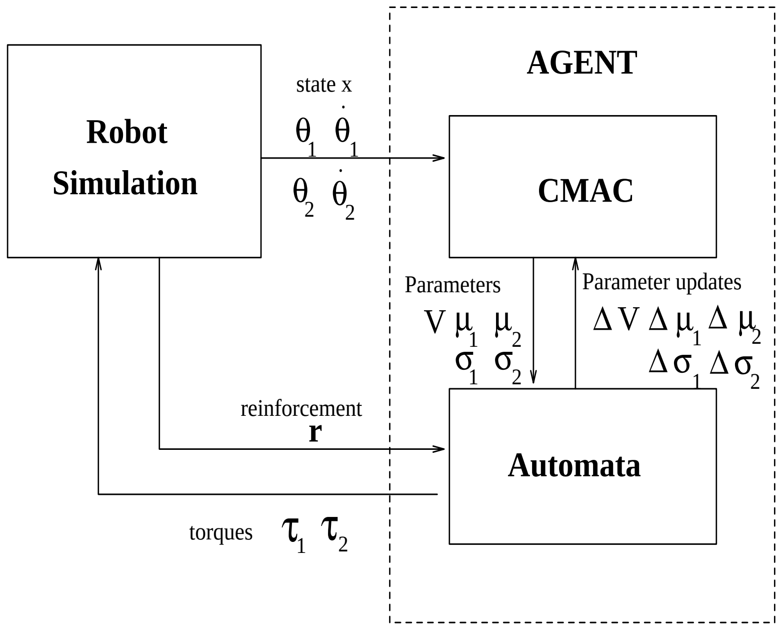

Figure 2: Interaction between the agent and robot.

operating on each input, with each quantizing function corresponding to a component of the desired output value. The sum of the weights indexed by these components makes up the output value. The CMAC performs fast function approximation and has good generalization properties which help learning since function values in neighbouring states are usually similar.

4.1 Implementation of a CMAC

The example shown in Figure 3 illustrates the method of implementation and the generalization ability of a CMAC. There are six resolution elements (in Albus' notation, $Q = 6$), four quantizing functions ($K = 4$) and two inputs which are the angular positions of each joint, θ_1 and θ_2 ($N = 2$).

Point A is a point in input space corresponding to a previously visited state $\boldsymbol{x}_A = [\theta_{A1}, \theta_{A2}] = [3.14, 0.00]$ with its output value p_A updated during the last visit and point B is a point nearby in input space with $\boldsymbol{x}_B = [\theta_{B1}, \theta_{B2}] = [3.44, 0.20]$ which has not been visited before.

The operations involved in order to arrive at the output value p_A are as follows. The quantizing functions quantize the input values $[\theta_{A1}, \theta_{A2}] = [3.14, 0.00]$ to yield sets of resolution elements $m^*_{A1} = \{2_1, 3_2, 2_3, 2_4\}$ and $m^*_{A2} = \{2_1, 3_2, 2_3, 2_4\}$, where the numbers in the sets and their subscripts are identifiers of the selected resolution elements and their quantizing functions, respectively. The resolution elements from corresponding quantizing functions relate to a component of the output p_A, hence the number of components forming the output value is equal to the number of quantizing functions for each input variable. The set of these components is $A^*_A = \{\{2_1, 2_1\}, \{3_2, 3_2\}, \{2_3, 2_3\}, \{2_4, 2_4\}\}$, and the sum of the weights indexed by these components gives the output p_A i.e. $p_A = w_1[\{2_1, 2_1\}] + w_2[\{3_2, 3_2\}] + w_3[\{2_3, 2_3\}] + w_4[\{2_4, 2_4\}]$.

Likewise, $A^*_B = \{\{2_1, 2_1\}, \{3_2, 3_2\}, \{3_3, 2_3\}, \{2_4, 2_4\}\}$ with all except the third compo-

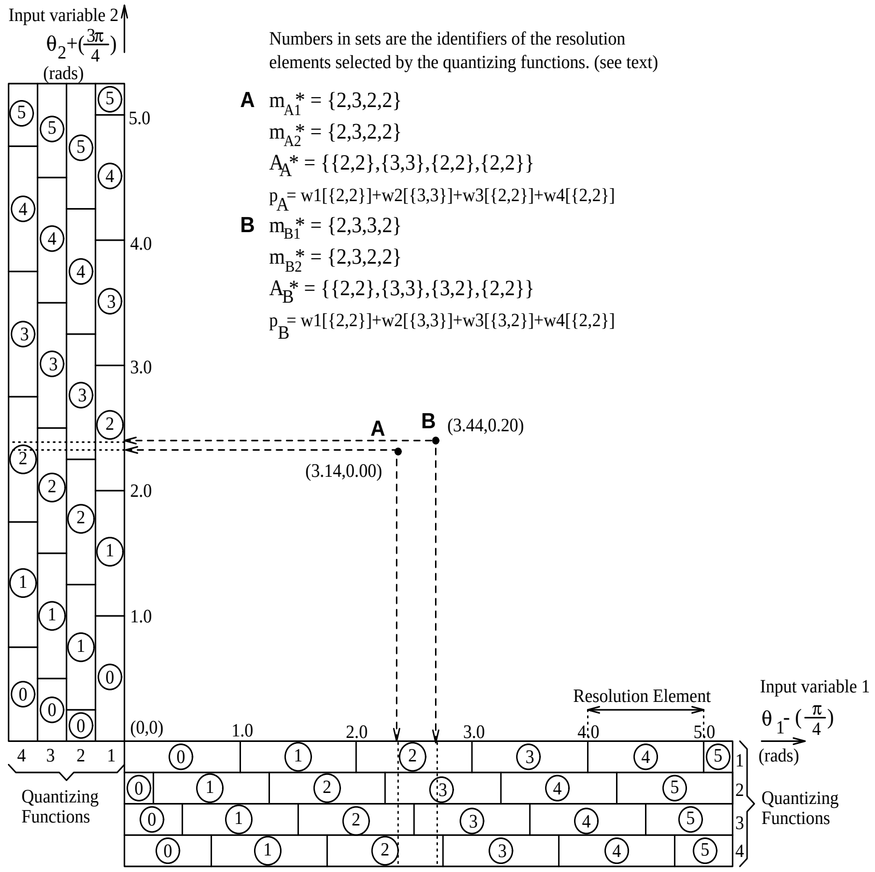

Figure 3: Example of CMAC usage.

nent being the same as in A_A^*. Thus, $p_B = w_1[\{2_1, 2_1\}] + w_2[\{3_2, 3_2\}] + w_3[\{3_3, 2_3\}] + w_4[\{2_4, 2_4\}]$. The result of this is generalization where the output value of p_B is similar to that of p_A (since three out of four indexed weights to be summed are the same for both p_A and p_B) eventhough only p_A had been updated previously.

The update procedure is as follows: if p is to be updated so that p^* is computed the next time the state is visited, an amount $\Delta = \frac{p^*-p}{|A^*|}$ is added to each weight, where $|A^*|$ is the number of components in set A^*.

The quantizing interval is the width of a resolution element. It can be seen that the quantizing functions are offset by $\frac{1}{K}$ of the quantizing interval. The offset corresponds to the quantizing resolution that can be achieved. Thus, if a bigger neighbourhood of generalization is required, the number of quantizing functions K can be reduced or the quantizing interval increased, while a greater resolution of the CMAC can be achieved by either increasing K or decreasing the quantizing interval. The number of resolution elements Q for each quantizing function is obtained by dividing the range of the input by the quantizing interval.

4.2 Experiment specific details

The accuracy of the functions to be approximated and represented by CMACs are limited by the quantizing resolution. A higher quantizing resolution requires a larger number of weights in each CMAC and more exploration has to be performed before the entire input space can be considered 'visited' due to the smaller neighbourhood of generalization. However, this does not mean that an agent with CMACs having higher quantizing resolutions need more training cycles to reach a particular level of performance as the higher accuracy of the functions contribute to improved performance. (see Section 8)

The CMACs that were implemented in the Case 1 experiment (see Section 8.1) are identical to the example shown in Figure 3, except that there are now four input variables, i.e. the input space is four-dimensional. The CMACs had six resolution elements ($Q = 6$) over the input range for each quantizing function, there were four quantizing functions for each input ($K = 4$) and four inputs ($N = 4$) i.e. angular positions and velocities of each joint, giving a total of $KQ^N = 5,184$ weights for each CMAC. This allowed a quantizing resolution of 0.25 rads for angular positions and 1.0 rad/s for angular velocities for the input ranges that we considered.

If the quantizing resolution is doubled for each of the four inputs, i.e. increase the number of resolution elements Q from 6 to 12 for each input, while keeping the number of quantizing functions K at 4 and the number of inputs N at 4, the number of weights in each CMAC would increase from 5,184 to $KQ^N = 82,944$, a sixteen-fold increase in memory requirements. A common technique used when there is a large number of weights is *hashing* where more than one weight is mapped to a particular location in memory.

In the Case 2 experiment, the quantizing resolution for angular positions was doubled to 0.125 rads while the quantizing resolution for angular velocities was maintained at 1.0 rad/s. The number of resolution elements for angular positions was increased to $Q_p = 12$ while for angular velocities, $Q_v = 6$ as before. The number of quantizing functions K and inputs N was the same as in Case 1. The number of weights needed to store each CMAC increased to $K(Q_p^{N_p} Q_v^{N_v}) = 4(12^2 \times 6^2) = 20,736$ i.e. a four-fold increase without hashing, where N_p and N_v refer to the number of inputs having Q_p and Q_v resolution elements respectively.

5 Reinforcement Learning

Reinforcement learning methods are based on a single scalar *reinforcement* signal from the environment that evaluates the performance of the learning system. It differs from supervised-learning methods since the reinforcement signal does not directly contain gradient or directional information i.e. it does not indicate whether improvement is possible and how (by how much and in which direction) the behaviour should be changed for improvement. The learning system has to infer this directional information from a collection of evaluation signals. Hence, reinforcement learning methods are able to overcome one of the limitations of supervised learning, the requirement of a 'teacher'.

The learning system has to explore and try different actions in order to discover actions that can lead to improved performance. In doing so, it encounters a conflict between performing actions that enable it to learn more about the environment (and hence take better actions in future) and actions that lead to payoffs based on the knowledge it currently has. Thrun [12] calls this the *exploration vs. exploitation* trade-off and suggests several directed exploration methods to minimize the costs of learning.

One of the earlier demonstrations of the potential of reinforcement learning for control tasks was the pole-balancing experiment of Barto, Sutton and Anderson [2] mentioned above. Barto, Sutton and Watkins [3] showed how a class of adaptive prediction methods that Sutton called temporal differences (TD) methods [9] can be used to solve *sequential* *decision* tasks in which an action selected at a given time will influence future actions and the final outcome and both short- and long-term consequences of decisions have to be considered.

5.1 Sequential decision tasks

These tasks can be formulated in terms of a stochastic dynamical system whose behaviour (which can be probabilistic) unfolds over time under the influence of a decision-maker's actions. The objective is to find strategies for selecting actions so as to maximize a measure of long term payoff gain.

Mathematically, the sequential decision task can be treated as a Markovian decision problem exhibiting the *Markov property* whereby the system *state* at any moment together with future input determine (the probabilities of) all aspects of the future behaviour of the system, regardless of how the the present state was reached.

In a typical sequential decision task, the mode of operation is as follows, where *agent* refers to the decision-maker:

1. The agent observes the environment and determines the current state.
2. The agent performs an action.
3. The environment delivers a payoff, which is a function of its state, the agent's action and possibly random disturbances.
4. The environment makes a transition to a new state, dependent on the same three factors above.
5. The agent observes the new state, performs another action, receives another payoff and the environment changes state again. This process is repeated for a number of time periods or until some terminating condition, e.g. goal is reached, is met.

The agent's task is to select actions that maximize the cumulative payoff over time. If the number of time periods, the *horizon*, of the operation is infinite, a *discount factor* γ, is introduced which allows payoffs to be weighted according to when they occur. Normally, the weights decrease with increasing temporal remoteness of the payoffs, thus the weighted sum will have a finite value. The discount factor adjusts the degree to which the long-term consequences of actions must be accounted for.

The agent uses a rule called a *policy* π to select actions depending on its state x; a policy's *return* is the weighted sum of the payoffs r the agent would receive if the policy π was used to select all the actions. When random factors are involved, this will be the *expected return* for state x:

$$E_\pi\left[\sum_{t=0}^{\infty}\gamma^t r_{t+1}\right] \tag{1}$$

where $0 \leq \gamma \leq 1$.

The expected return needs to be written as a function assigning to each state x the expected return if policy π is followed starting from state x:

$$V^\pi(x) = E_\pi\left[\sum_{t=0}^{\infty}\gamma^t r_{t+1} \mid x_0 = x\right] \tag{2}$$

The function $V^{\pi}(x)$ is the *evaluation function* of state x for policy π; it gives an immediately accessible prediction of return at any state. A policy that maximizes the expected return for all possible initial states is called an *optimal policy* π^*.

In cases where payoff is zero everywhere unless some goal state is reached, selecting actions to maximize return is the same as selecting actions that bring about the goal state and with the discount factor, the agent selects actions that bring about the goal state in the fewest time periods.

5.2 Solving sequential decision tasks

In order to estimate optimal policies in the absence of a complete model of the decision task, adaptive or learning methods which learn about the system underlying the task while interacting with it are used.

There are two general approaches in these methods. Firstly, the *model-based* approach involves constructing a model of the decision task in the form of estimates of the state-transition and payoff probabilities by keeping track of the frequencies with which various state transitions and payoffs occur while interacting with the system. Once an accurate model is formed, computational techniques such as dynamic programming can be applied to find an optimal policy.

Secondly, there is the *direct* approach where instead of learning a model of the decision task and estimating state-transition and payoff probabilities, the policy is adjusted directly as a result of the observed consequences of performing various actions.

Reinforcement learning using the TD procedure [3] is a direct and incremental approach for learning the evaluation function $V(x)$ in Equation 2 and an (optimal) decision policy. [2] The TD procedure works by using a parameter estimation method to obtain approximations to information that can be supplied by a model of the system underlying the decision task if such a model was available, and then applying stochastic dynamic programming methods on these approximations.

The TD error [3], which is the difference between the evaluation function value estimated by the model and the true value, is obtained by:

$$\varepsilon_{t+1} = r_{t+1} + \gamma V_t(x_{t+1}) - V_t(x_t) \tag{3}$$

with the quantities evaluated at time t and $t+1$.

The evaluation function is updated according to:

$$V_{t+1}(x_t) = V_t(x_t) + \alpha \varepsilon_{t+1} \tag{4}$$

where α is the learning rate parameter.

The policy is then updated to make an action that has led to a better-than-expected performance more likely and an action that has led to worse-than-expected performance less likely to be selected the next time state x_t is visited. The TD error ε_{t+1} provides a convenient utility measure of the last action taken with respect to the expected return of all actions in state x_t; a positive ε_{t+1} means that the last action led to better-than-expected performance and vice versa.

---

[2] It is not guaranteed that an optimal policy will always be found by the TD method and the degree of optimality achievable depends on many specific implementation details such as the manner of representing system states.

[3] In some implementations, the TD error is the output of a *critic*. It is also known as a *heuristic reinforcement signal.*

5.3 Experiment specific details

The learning procedure used is the on-line method described above in which the agent seeks to learn the evaluation function $V(\boldsymbol{x})$ of a policy and adjusts the policy incrementally for improved performance in every time step.

The TD (temporal difference) error is obtained by Equation 3, repeated here for emphasis:

$$\varepsilon_{t+1} = r_{t+1} + \gamma V_t(\boldsymbol{x}_{t+1}) - V_t(\boldsymbol{x}_t) \tag{5}$$

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

The evaluation function, represented by a CMAC, is updated according to:

$$V_{t+1}(\boldsymbol{x}) = V_t(\boldsymbol{x}) + \alpha\varepsilon_{t+1}c_t(\boldsymbol{x}) \tag{6}$$

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

This is similar to Equation 4 above with the addition of $c_t(\boldsymbol{x})$, a *stimulus trace* affecting the states through which the system passed in the time steps preceding t. This stimulus trace, which decays at a rate dependent on the parameter λ, can accelerate the learning process as shown by Sutton [9].

The policy updating process is based on the principle described above, but since the appropriate *real-valued* torques have to be determined at each state, the *stochastic hill-climbing* theory described in next section needs to be incorporated.

6 Stochastic Hillclimbing

Williams [14] proposed a method of combining theory from stochastic learning automata [6] with reinforcement learning in a connectionist network framework. Stochastic learning automata provide the stochastic behaviour necessary for exploration in reinforcement learning, while a connectionist network framework permits estimations of expected return to be learnt using an update rule similar to back-propagation [5].

6.1 Associative stochastic learning automata

A *stochastic learning automaton* is an abstract machine that selects actions and receives feedback from the environment evaluating those actions. It selects actions randomly based on some internally stored distribution over the set of possible actions. This distribution is updated on the basis of the evaluative feedback from the environment called *reinforcement*, the idea being that actions receiving favourable evaluation should become more likely choices by the automaton.

Williams [14] generalizes the stochastic learning automaton in two ways. The first concerns the choice of state space for the automaton, denoted by Θ, in which a function $\psi : \Theta \to \Delta^m$ maps it to the m-dimensional simplex of action probabilities, yielding a parameterized-state stochastic learning automaton.

Second, the automaton is allowed to have non-reinforcement input i.e. *context input*, so that it can learn a general input-output mapping. The preferred action may differ in differing contexts and the automaton is trying to learn which actions to associate with different context inputs. This can be achieved by having a bank of stochastic learning automata, one for each possible context input.

With these generalizations, the resulting automaton is called an *associative stochastic learning automaton* (ASLA).

Consider an ASLA which is the ith unit in a network. The output is denoted as y_i and the input $\boldsymbol{x}^i$. y_i is drawn from a distribution depending on $\boldsymbol{x}^i$ and a set of parameters $\boldsymbol{w}^i$ consisting of parameters w_{ij} denoting the strength of the connection from the jth unit to the ith unit. Let $\boldsymbol{W}$ denote all the parameters $\boldsymbol{w}^i$ of the network. A probability mass function $g_i(\xi, \boldsymbol{w}^i, \boldsymbol{x}^i) = Pr\{y_i = \xi \mid \boldsymbol{w}^i, \boldsymbol{x}^i\}$ determines the the value of y_i as a function of the parameters of the unit and its input. [4]

6.2 Stochastic hillclimbing algorithms

Learning procedures which 'stochastically hillclimb' in a measure of performance such as the expected value of the reinforcement signal $E\{r \mid \theta\}$, where $\theta \in \Theta$, can now be described [14]. These procedures must search the parameter space Θ for a point where $E\{r \mid \theta\}$ is maximum. In the case where a network of ASLA is considered and $\boldsymbol{W}$ denotes the parameters, the learning task becomes that of finding $\boldsymbol{W}$ which maximizes $E\{r \mid \boldsymbol{W}\}$.

Restricted REINFORCE Algorithm

In a *restricted* reinforcement learning task, the automaton makes exactly one action selection for each reinforcement value received. A network faces a such a task and at the end of each trial, reinforcement r is received and the parameters $\boldsymbol{W}$ are adjusted according to:

$$\Delta w_{ij} = \alpha_{ij}(r - b_{ij})e_{ij} \tag{7}$$

where α_{ij} is a *learning rate factor*, b_{ij} is a *reinforcement baseline*, and $e_{ij} = \frac{\partial \ln g_i}{\partial w_{ij}}$ is called the *characteristic eligibility* of w_{ij}. The factor $(r - b_{ij})$ is called the *reinforcement offset* and the reinforcement baseline b_{ij} is assumed to be conditionally independent of y_i, given $\boldsymbol{W}$ and $\boldsymbol{x}^i$. The rate factor α_{ij} is assumed to be non-negative and essentially constant, except for possible dependence on i and j but not on the input $\boldsymbol{x}^i$ to the unit. Any algorithm having this form, to be applied to the restricted reinforcement learning problem, is called a *restricted REINFORCE* algorithm.

Theorem 1 (Williams): For any restricted REINFORCE algorithm, the inner product of $E\{\Delta\boldsymbol{W} \mid \boldsymbol{W}\}$ and $\nabla_{\boldsymbol{W}} E\{r \mid \boldsymbol{W}\}$ is non-negative. Furthermore, if $\alpha_{ij} > 0$ for all i and j, then this inner product is zero only when $\nabla_{\boldsymbol{W}} E\{r \mid \boldsymbol{W}\} = 0$. Also, if $\alpha_{ij} = \alpha$ is independent of i and j, then $E\{\Delta\boldsymbol{W} \mid \boldsymbol{W}\} = \alpha\nabla_{\boldsymbol{W}} E\{r \mid \boldsymbol{W}\}$.

This means that the average update direction in parameter space lies in a direction in which the performance measure $E\{r \mid \boldsymbol{W}\}$ is increasing, thus performing a local optimization in the reinforcement learning task.

Theorem 1 also applies to stochastic learning automata with or without context input and no interconnections between units. Under these circumstances the restricted REINFORCE algorithm reduces to learning algorithms associated with stochastic learning automata e.g. linear reward-inaction L_{R-I} and associative reward-inaction A_{R-I}.

[4] An ASLA has two parts: deterministic and stochastic. The parameters w_{ij} and input x^i provide the parameter(s) of a random distribution deterministically and output y_i is then drawn from this distribution.

Extended REINFORCE Algorithm

The restricted REINFORCE algorithm needs to be extended in order to cater for learning problems having a temporal credit-assignment component where more than one action may be selected for a single reinforcement value. For example, a network may be trained on a episode-by-episode basis [5] where each episode consists of k time steps during which the units may recompute their outputs and the context input may change. At the end of an episode, a single reinforcement r is delivered and the parameters are incremented by:

$$\Delta w_{ij} = \alpha_{ij}(r - b_{ij}) \sum_{t=1}^{k} e_{ij}(t) \tag{8}$$

with similar conditions for the reinforce baseline b_{ij} and α_{ij} terms. This algorithm is known as the *extended REINFORCE* algorithm.

Theorem 2 (Williams): For any extended REINFORCE algorithm, the inner product of $E\{\Delta\boldsymbol{W} \mid \boldsymbol{W}\}$ and $\nabla_{\boldsymbol{W}} E\{r \mid \boldsymbol{W}\}$ is non-negative. Furthermore, if $\alpha_{ij} > 0$ for all i and j, then this inner product is zero only when $\nabla_{\boldsymbol{W}} E\{r \mid \boldsymbol{W}\} = 0$. Also, if $\alpha_{ij} = \alpha$ is independent of i and j, then $E\{\Delta\boldsymbol{W} \mid \boldsymbol{W}\} = \alpha\nabla_{\boldsymbol{W}} E\{r \mid \boldsymbol{W}\}$.

6.3 Experiment specific details

The theory described above can be used for a connectionist network consisting of associative stochastic learning automata where the weights connecting the ASLA are updated with the REINFORCE algorithm. We have used the stochastic hillclimbing approach and the REINFORCE algorithm eventhough we have not implemented such a connectionist network because it allows the agent to have real-valued outputs, reinforcement and context inputs.

A further extension to the theory is the use of a multi-parameter random distribution, in particular, the normal distribution with two parameters, its mean μ and standard deviation σ. Since temporal-difference techniques are employed in our approach, the extended REINFORCE algorithm is used to determine changes to the parameters μ and σ in order to maximize the performance measure $E\{r \mid \boldsymbol{x}\}$.

We implemented a large continuous bank of ASLA with the state $\boldsymbol{x} = [\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2]$ as the context input. The 'correct' ASLA is selected at each state since its parameters μ and σ are functions of the input state. 'Each' ASLA is simply a stochastic learning automaton with no context input which provided a real-valued action at its output when selected. There are no connections between the ASLAs.

In the task under consideration, the actions in each state are torques to each joint of the robot whose magnitudes and directions need to be determined stochastically. Each torque is chosen from a gaussian probability distribution with the parameters mean μ and standard deviation σ represented in input state space by CMACs [6]. The probability of applying torque τ in a joint is given by

$$g(\tau, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(\tau-\mu)^2}{2\sigma^2}} \tag{9}$$

[5] An episode corresponds to a trial in the experiments described in Section 8.

[6] Prescott and Mayhew [8] used a similar approach to obtain the forward and angular velocities for a mobile vehicle in the task described in Section 2.

The REINFORCE algorithm for updating the parameters μ and σ is as follows:

$$\begin{aligned}
\Delta\mu &= \alpha(r - b_\mu)e^{(\mu)} \quad \text{where } e^{(\mu)} = \frac{\partial \ln g}{\partial \mu} = \frac{(\tau-\mu)}{\sigma^2} \\
\text{and} \quad \Delta\sigma &= \alpha(r - b_\sigma)e^{(\sigma)} \quad \text{where } e^{(\sigma)} = \frac{\partial \ln g}{\partial \sigma} = \frac{(\tau-\mu)^2-\sigma^2}{\sigma^3}
\end{aligned}$$

The characteristic eligibilities of parameter updates are denoted by $e^{(\mu)}$ and $e^{(\sigma)}$. $(r - b_\mu)$ 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} [3] (see Section 5.2), which provides a utility measure of the last action taken with respect to the expected return of all actions performed when the agent is in a particular state, can be used as the reinforcement offset. As the TD error provides a reinforcement offset value at every time step, we have not adhered strictly to the requirement by the extended REINFORCE algorithm of a single reinforcement signal at the end of an episode. (see Section 7)

The characteristic eligibilities of the preceding states are weighted and summed (see Equation 8) to form stimulus traces $s_t^{(\mu)}(\boldsymbol{x})$ and $s_t^{(\sigma)}(\boldsymbol{x})$ as in Equation 6 for updating μ and σ respectively. Since the operations involved in updating μ and σ are the same, let

$$\boldsymbol{p} = \begin{bmatrix} \mu \\ \sigma \end{bmatrix} \quad ; \quad \boldsymbol{e} = \begin{bmatrix} e^{(\mu)} \\ e^{(\sigma)} \end{bmatrix} \quad ; \quad \boldsymbol{s} = \begin{bmatrix} s^{(\mu)} \\ s^{(\sigma)} \end{bmatrix}$$

Writing $\boldsymbol{p}_t$, $\boldsymbol{e}_t$ and $\boldsymbol{s}_t$ for the terms above evaluated at time t, the policy update rule is:

$$\boldsymbol{p}_{t+1}(\boldsymbol{x}) = \boldsymbol{p}_t(\boldsymbol{x}) + \beta\varepsilon_{t+1}\boldsymbol{s}_t(\boldsymbol{x}) \tag{10}$$

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

The learning rate parameter is denoted by β instead of α as in Equations 7 and 8 to distinguish it from α in Equation 6.

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

7 Reinforcement Schedule

The learning ability of the agent depends to a large extent on the magnitude of the reinforcement signal and the time at which it is received. The reinforcement schedule is essentially a reward function. Since collisions occur more frequently than successes, especially in the early stages of learning, the negative reinforcement that comes with collisions cannot be too large as to overwhelm any positive reinforcement previously acquired when the robot reached the destination. However, insufficient negative reinforcement decreases the agent's ability to avoid obstacles.

We implemented the following schedule, where r is the reinforcement:

1. If the destination is reached, $r = 50 + 50e^{-(|\dot{\theta}_1|+|\dot{\theta}_2|)}$. This gives a higher than normal payoff if the destination is approached with low velocities.

2. If a collision occurs, $r = -5\times|\dot{\theta}|$ of the link involved, subject to the constraint that the maximum negative reinforcement is $r = -50$.

3. If both $\dot{\theta}_1$ and $\dot{\theta}_2$ are greater than 10 rad/s, $r = -10$. This is to indicate that excessively high velocities are undesirable.

The last reinforcement rule violates to some extent the requirement of a single reinforcement signal at the end of an episode by the extended REINFORCE algorithm described in Section 6.2. Its inclusion was to prevent the selection of excessively high torques and velocities as the discount factor γ in Equation 5 encourages the agent to generate actions enabling the goal to be reached in as few time steps as possible. Mahadevan and Connell [4] also used differing reinforcement values in non-terminal states to encourage the learning of desirable behaviours.

8 Results

8.1 Initial values

The following values were used for the agent's parameters: $\alpha = 0.1$, $\beta = 0.1$, $\lambda = 0.5$ and $\gamma = 0.95$ [7]. We set the standard deviation of torques $\sigma_1(\boldsymbol{x})$ and $\sigma_2(\boldsymbol{x})$, for joints 1 and 2 respectively, to 10.0 Nm for all states at the beginning so that the agent starts with exploratory behaviour and does not get stuck at local maxima in the early stages. The mean torques $\mu_1(\boldsymbol{x})$ and $\mu_2(\boldsymbol{x})$ and the evaluation function $V(\boldsymbol{x})$ are set to zero throughout at the beginning. Since the stimulus traces decay rapidly, we need only to maintain a queue of 5 previous states and characteristic eligibilities and update the function values of these states according to Equations 6 and 10.

We present results for two cases:

Case 1 The quantizing resolution of the CMACs for joint positions θ_1 and θ_2 is 0.25 rad.

Case 2 The quantizing resolution of the CMACs for joint positions θ_1 and θ_2 is 0.125 rad.

The quantizing resolution for joint velocities $\dot{\theta}_1$ and $\dot{\theta}_2$ is 1.0 rad/s for both cases. As described in Section 4, the amount of memory required to store each CMAC for Case 2 is four times that of Case 1.

For each case, the learning curves, evaluation function and policy are examined. (References to figures are given in Case 1 followed by Case 2 order.)

8.2 Training

Training is conducted in cycles of 5,000 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 from a random position in the workspace. This

[7] The value of $\gamma < 1.0$ is required for the stability of the difference equations formed by Equations 5, 6 and 10 if the training operation has an infinite-horizon. In the experiments described in Section 8, training was conducted in trials consisting of at most 100 steps, i.e. the horizon is finite, and the values of V, μ and σ did not grow without bound even when $\gamma = 1.0$. However, performance in terms of the number of successes per cycle was better when $\gamma = 0.95$.

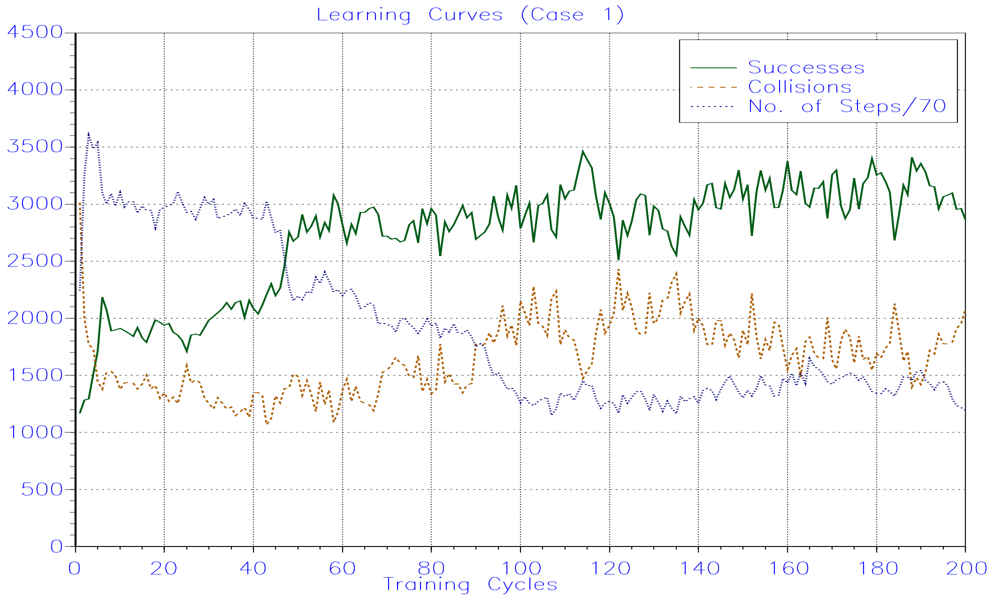

Figure 4: Case 1 Learning curves for 200 cycles

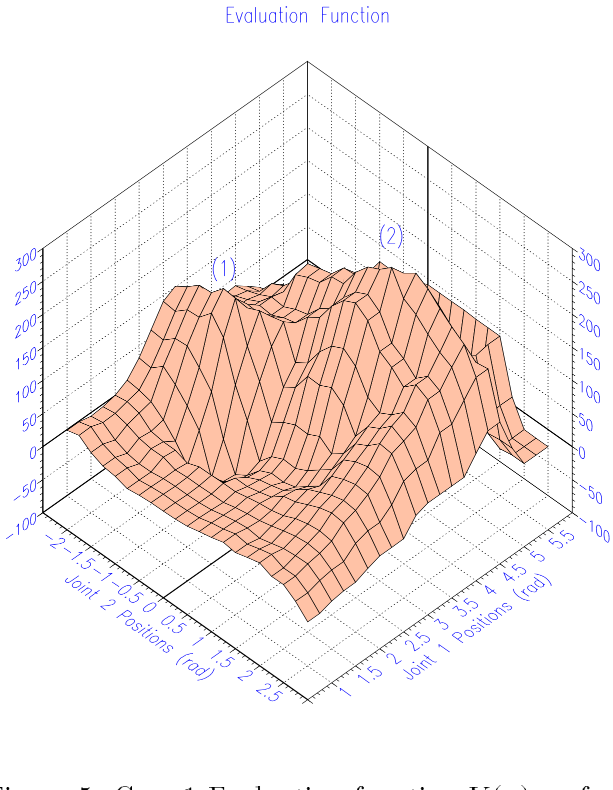

Figure 5: Case 1 Evaluation function $V(x)$ surface

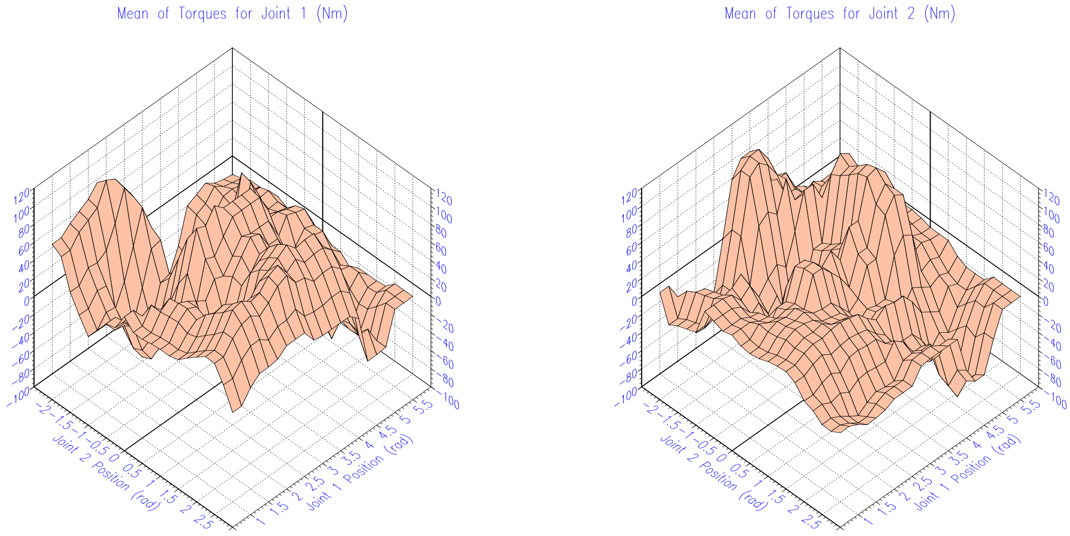

Figure 6: Case 1 Policy - Mean $\mu_1(x)$ and $\mu_2(x)$ of gaussian pdfs for joints 1 and 2

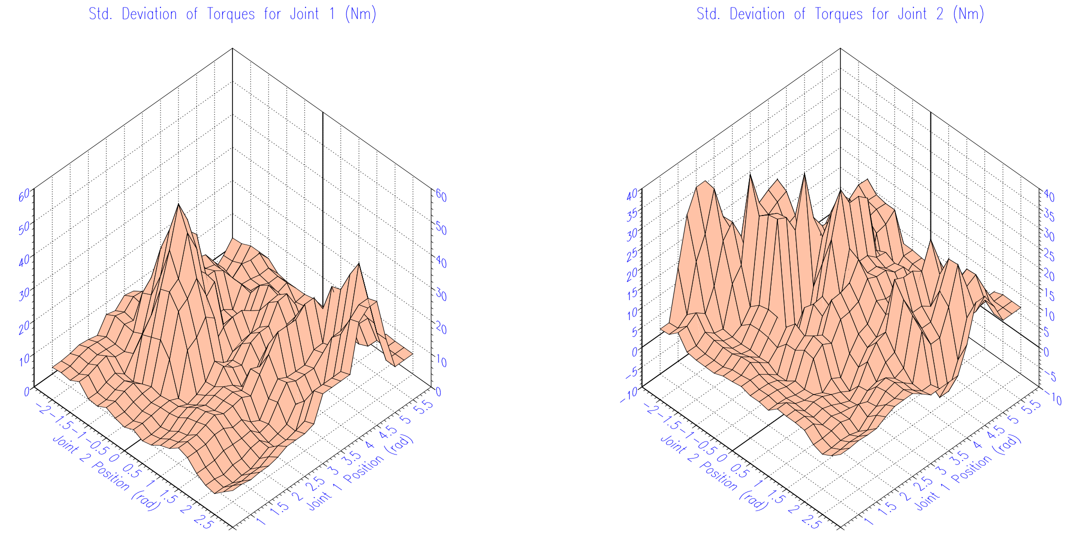

Figure 7: Case 1 Policy - Std deviation $\sigma_1(x)$ and $\sigma_2(x)$ of gaussian pdfs for joints 1 and 2

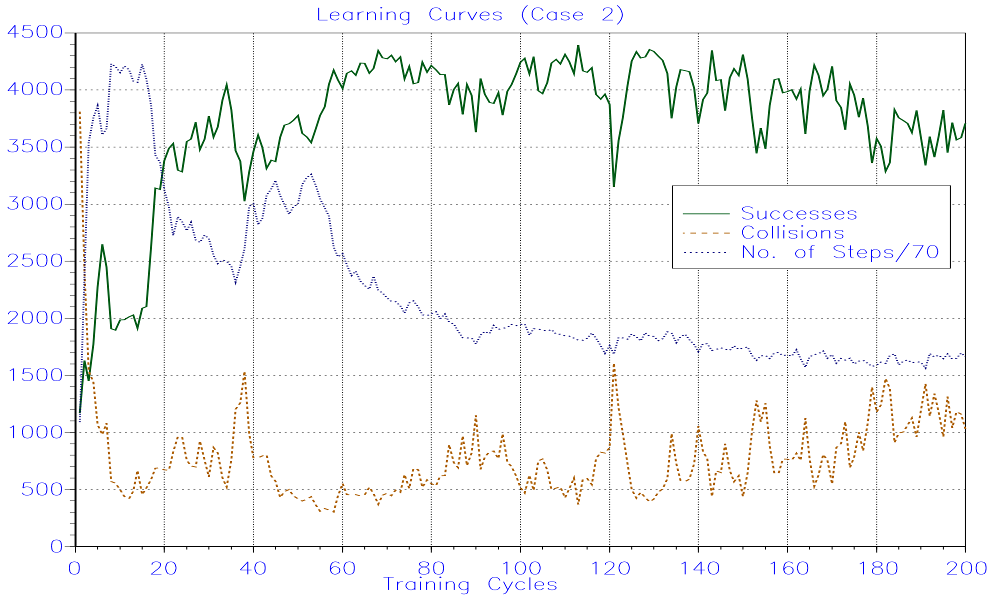

Figure 8: Case 2 Learning curves for 200 cycles

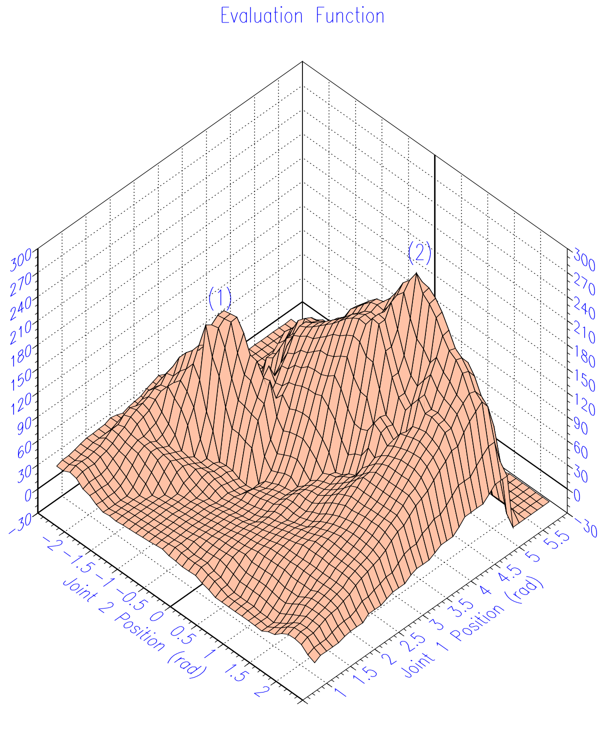

Figure 9: Case 2 Evaluation function $V(x)$ surface

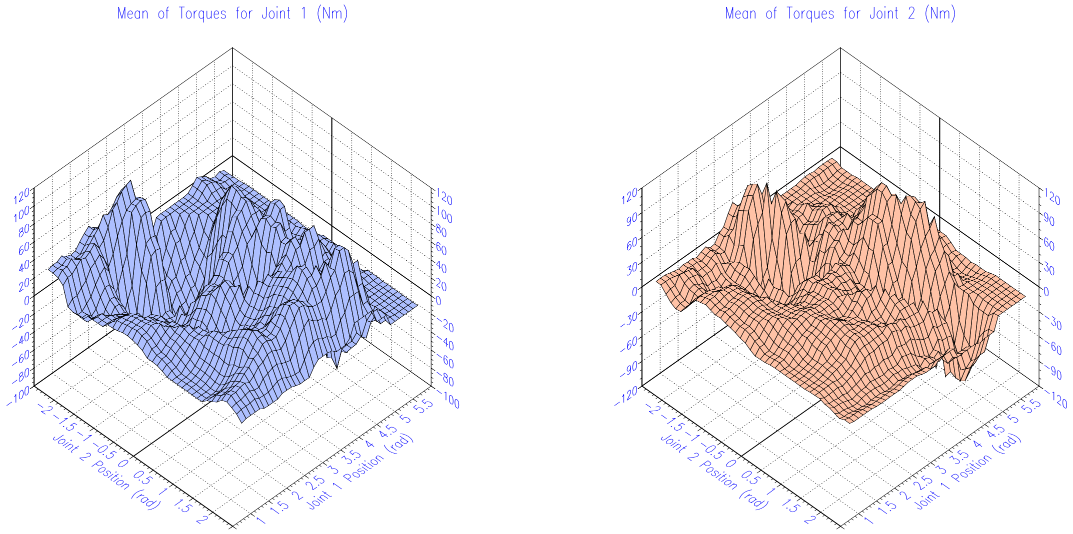

Figure 10: Case 2 Policy - Mean $\mu_1(x)$ and $\mu_2(x)$ of gaussian pdfs for joints 1 and 2

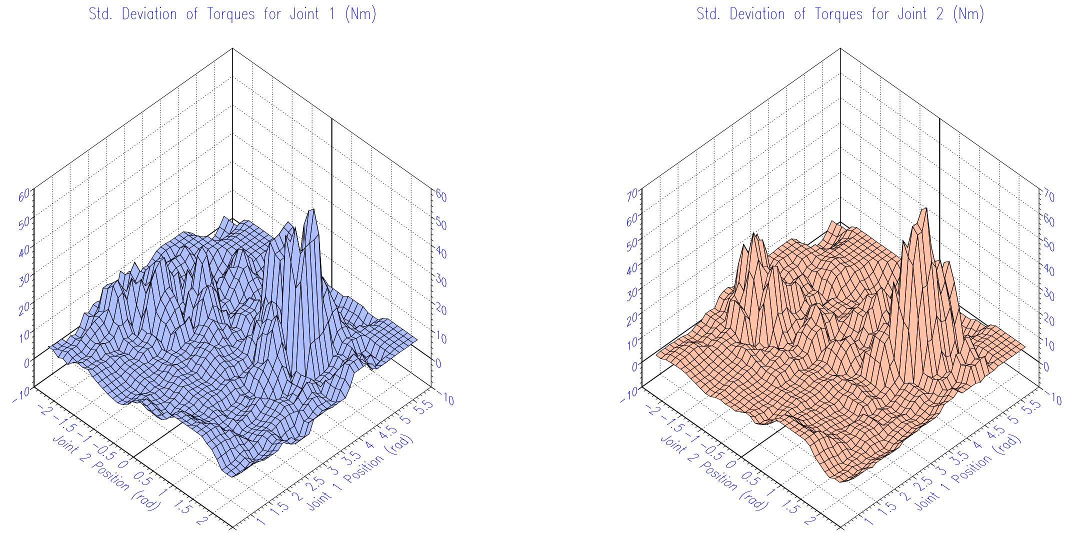

Figure 11: Case 2 Policy - Std deviation $\sigma_1(x)$ and $\sigma_2(x)$ of gaussian pdfs for joints 1 and 2

means that the agent is trained so as to enable the end-effector of the robot to reach the destination from different start positions in the workspace. [8]

Figures 4 and 8 show how the performance of the agent improves with training in terms of the increasing number of successes and decreasing number of collisions in each cycle for 200 training cycles. [9] The total number of steps taken in each cycle (divided by 70 to fit into the graph) is also indicated.

With a lower quantizing resolution (Case 1), the maximum performance level of about 3,000 successes and 1,600 collisions was reached after 60 cycles. The best performance was obtained at the 114th cycle with 3,461 successes and 1,473 collisions. The decreasing number of steps taken even after the number of successes and collisions have stabilized indicate an ongoing effort by the agent to generate actions that bring the robot to the destination in as few steps as possible. At the end of the 200 cycles, the average number of steps per cycle is about 100,000 steps, i.e. 20 steps per trial.

With a higher quantizing resolution (Case 2), the maximum performance level of about 4,100 successes and 700 collisions was also reached after 60 cycles. The best performance was obtained at the 113th cycle with 4394 successes and 369 collisions. The number of steps per cycle decreases until about 115,000 at the end of the 200 cycles, i.e. 23 steps per trial.

It can be seen that the learning rate is not affected by the quantizing resolution of the CMACs. However, the maximum performance level was greatly enhanced by the increased quantizing resolution in Case 2, resulting in a more than 20% increase in the number of successes per cycle.

8.3 Evaluation function and policy learnt

The evaluation function $V(\boldsymbol{x})$ learnt are shown in Figures 5 and 9. Since the input state-space is four-dimensional, the graphs show the expected return as a function of joint positions θ_1 and θ_2 with $\dot{\theta}_1$ and $\dot{\theta}_2$ both fixed at 0.5 rad/s. There are two possible joint configurations by which the destination can be reached, i.e. two solutions to the inverse kinematics, when $\theta_1 = 3.14$, $\theta_2 = -0.87$ and $\theta_1 = 4.01$, $\theta_2 = 0.79$. These correspond to the two peaks indicated by (1) and (2) respectively in the evaluation function surfaces shown. Peak (2) is higher than peak (1), indicating that the agent was able to reach the destination with the joint configuration corresponding to peak (2) more often (thus receiving more reward) during training.

The learnt policy can be seen from the mean $\mu_1(\boldsymbol{x})$ and $\mu_2(\boldsymbol{x})$ functions in Figures 6 and 10 and standard deviation $\sigma_1(\boldsymbol{x})$ and $\sigma_2(\boldsymbol{x})$ functions in Figures 7 and 11. As in the case of $V(\boldsymbol{x})$, these functions are shown against joint positions θ_1 and θ_2, with $\dot{\theta}_1$ and $\dot{\theta}_2$ both fixed at 0.5 rad/s.

It is not obvious from these graphs that a sensible policy had been acquired by the agent. Detailed analysis of two motions with different starting positions and different final joint configurations presented in the next sub-section show that a good policy had indeed been learnt. In general, $\mu_2(\boldsymbol{x})$ and $\sigma_2(\boldsymbol{x})$ can be seen to quite high near the inverse kinematic solutions. In most parts of the state-space, the final values of the four parameter

[8] This task is more difficult than those finding a solution from a fixed start position as described in Sutton [10] and Thrun [12].

[9] The sum of the number of successes and collisions does not equal 5,000 since trials which exceed 100 steps are terminated, e.g. trials which begin with both links below obstacle A from where it is impossible to reach the destination.

functions are different from their starting values, indicating that a big proportion of the input space had been explored.

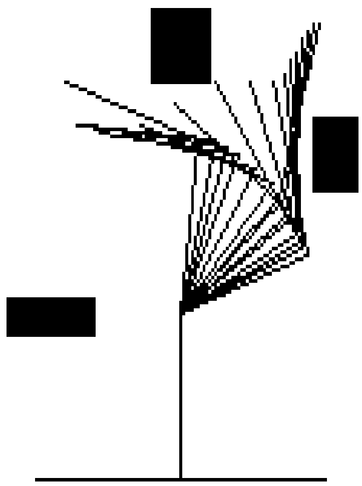

Figure 12: The path taken by the robot in a 33-step sequence from a start point to the destination.

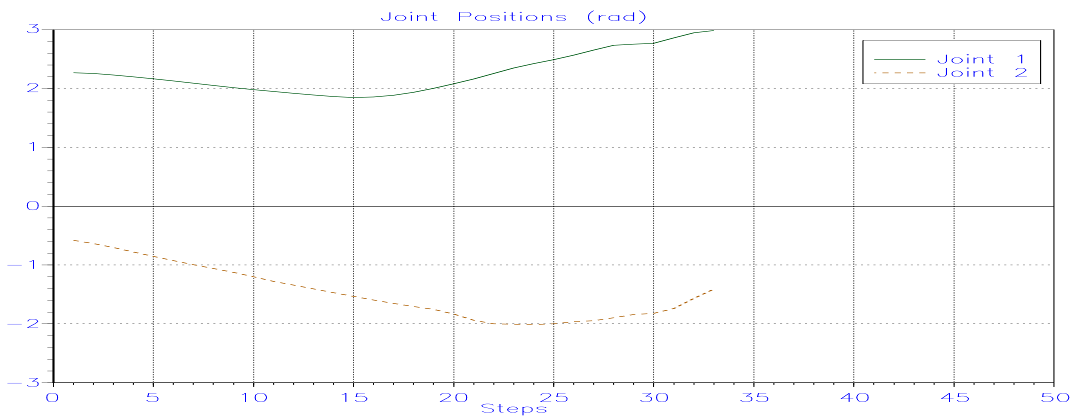

Figure 13: Variation in joint positions θ_1 and θ_2 during the 33-step sequence.

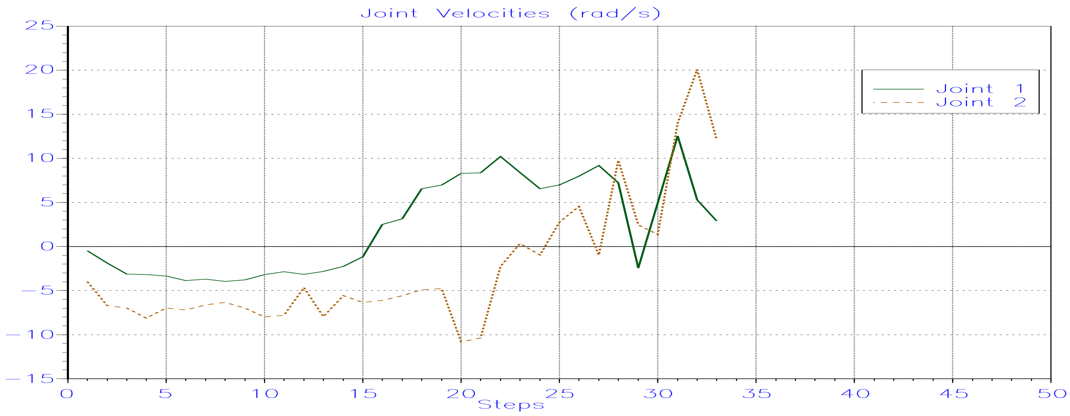

Figure 14: Variation in joint velocities $\dot{\theta}_1$ and $\dot{\theta}_2$ during the 33-step sequence.

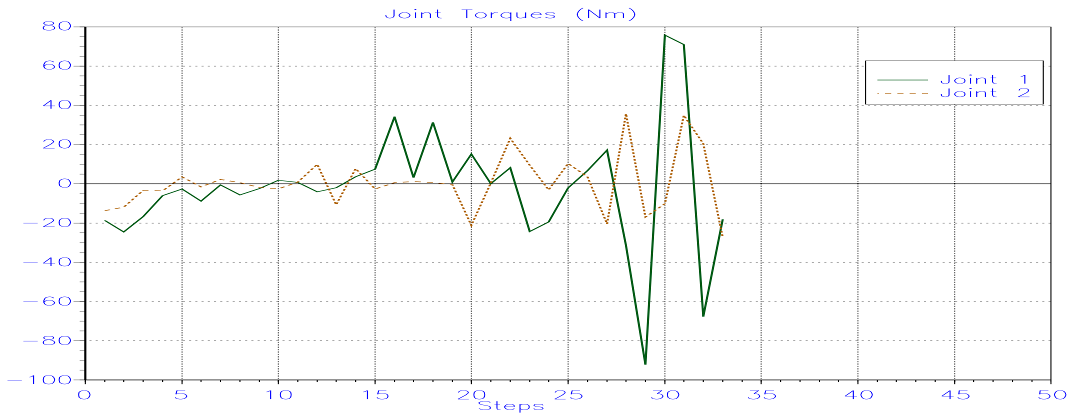

Figure 15: Variation in joint torques τ_1 and τ_2 during the 33-step sequence.

Figure 16: Variation in expected return V during the 33-step sequence.

8.4 Detailed analysis of two motions

Two detailed studies of the ability of the robot to move from different start positions in the workspace to the destination are now presented. In these examples, the quantizing resolution of joint positions in the CMACs is 0.125 rad, i.e. Case 2 above. The strategies learnt are quite sophisticated and frequently involve backing-up motions which position the joints away from the obstacles before an advance towards the destination is made. Although there are two joint configurations by which the destination can be reached, the agent was able to select the final configuration which required fewer steps from the start position.

8.4.1 Motion 1

Figure 12 shows the path taken by the robot from a start position between obstacles B and C in the top right area of the workspace. The changes in the positions, velocities and torques at the joints are shown in Figures 13, 14 and 15.

Angular positions The position profile that is achieved is smooth. Both joint angles θ_1 and θ_2 (see Figure 1 for a description of the angles referred to by θ_1 and θ_2) decrease in the first 15 steps in a backing-up motion, with θ_2 decreasing faster than θ_1. Once the end of link 2 has moved below obstacle B, θ_2 remains constant between the 22nd and 25th steps while θ_1 is increased. Finally, when the destination is within reach, θ_2 increases and the end of link 2 reaches the destination at the end of the 33rd step.

Angular velocities The joint velocities are consistent with the the behaviour described above. Both joint 1 velocity $\dot{\theta}_1$ and joint 2 velocity $\dot{\theta}_2$ are negative in the first 15 steps. Then $\dot{\theta}_1$ becomes positive and $\dot{\theta}_2$ is nearly zero between the 22nd and 25th steps. Finally, both $\dot{\theta}_1$ and $\dot{\theta}_2$ are strongly positive just before the destination is reached despite the first reinforcement rule in the reinforcement schedule described in Section 7. (see Section 9.3 for a discussion)

Torques Let τ_1 and τ_2 refer to the torques applied in joints 1 and 2 respectively. Both τ_1 and τ_2 are negative at the start to cause the robot to move from rest. After that, τ_1 and τ_2 are nearly zero between the 5th and 15th steps in order to maintain fairly constant joint velocities. Then, τ_1 becomes positive to accelerate link 1 towards the destination, but in the 29th step τ_1 becomes strongly negative to decelerate link 1 to rest and prevent overshoot. Meanwhile, τ_2 has become positive to cause θ_2 to increase slightly towards the destination. In the last few steps before the destination is reached, τ_1 oscillates quickly between strongly positive and strongly negative to hasten arrival at the destination while preventing overshoot which would cause the end-effector to miss the destination completely. [10]

Expected return The expected return (Figure 16) increases steadily in the first 20 steps. The dip in expected return at the 22nd step corresponds to the moment when the end of joint 2 nearly touches obstacle B, indicating 'danger' of collision. Subsequently, the expected return increases until a second dip occurs in the 28th and 29th steps. This

[10] The torques in the final step should be ignored because no action is initiated and the current trial ends once the destination is reached.

dip reflects the 'danger' of missing the destination through overshoot and prompted a corrective action in the form of a strongly negative τ_1 to prevent a further increase in θ_1. Finally, the lower of the two peaks, i.e. peak (1), in the evaluation function surface shown in Figure 9 is reached at the destination. [11]

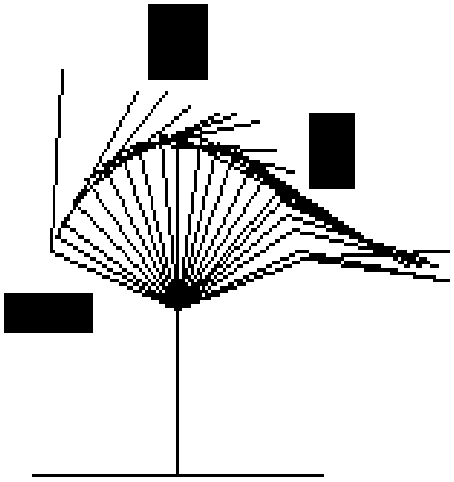

Figure 17: The path taken by the robot in a 44-step sequence from a start point to the destination.

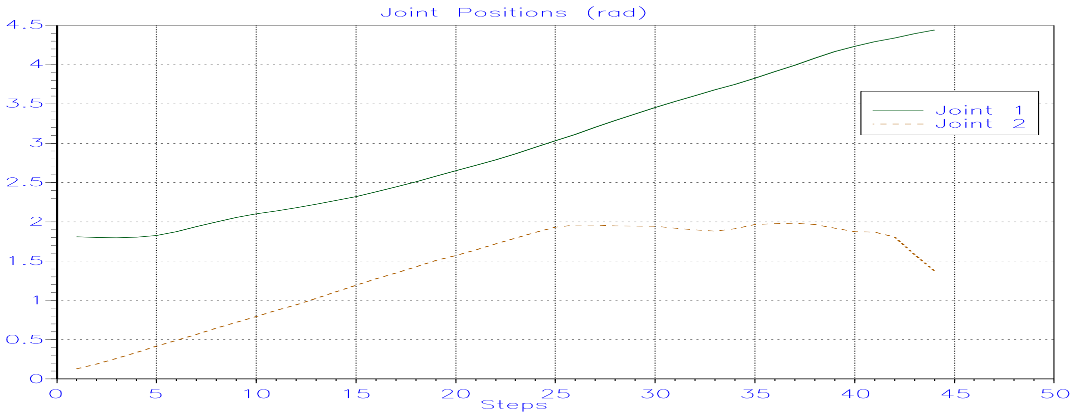

Figure 18: Variation in joint positions θ_1 and θ_2 during the 44-step sequence.

[11] The last dip in V should be ignored as reward is awarded to states from which an action leads to the goal. The goal state itself does not initiate any action and is never awarded a reward. However, V is not zero at the goal state due to its proximity to states with high expected return - this is a result of generalization in the CMAC.

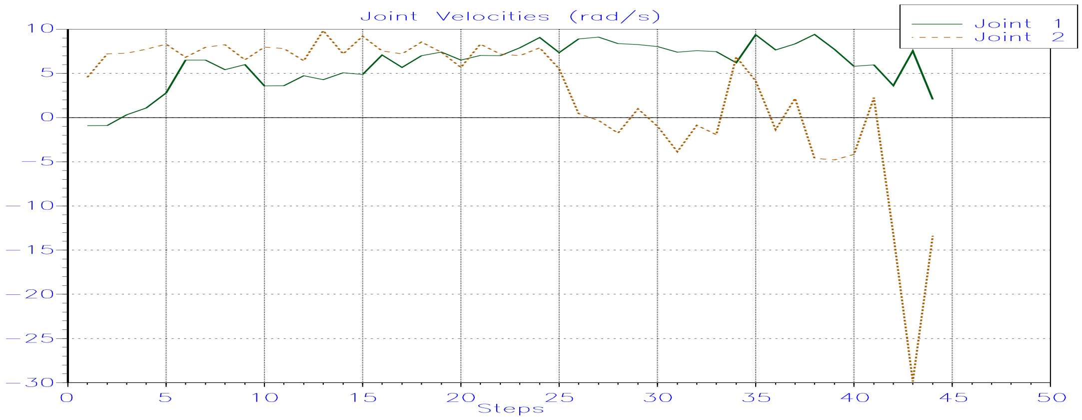

Figure 19: Variation in joint velocities $\dot{\theta}_1$ and $\dot{\theta}_2$ during the 44-step sequence.

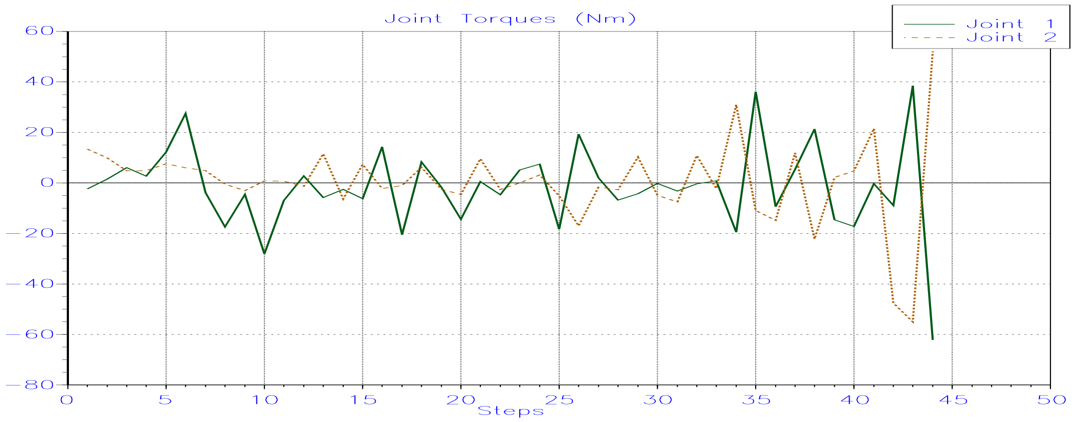

Figure 20: Variation in joint torques τ_1 and τ_2 during the 44-step sequence.

Figure 21: Variation in expected return V during the 44-step sequence.

8.4.2 Motion 2

Figure 17 shows the path taken by the robot from a start position below obstacle C in the right area of the workspace. The changes in the positions, velocities and torques at the joints are shown in Figures 18, 19 and 20.

Angular positions As in the previous example, the position profile that is achieved is smooth. This task does not require a backing-up motion and advancement towards the destination is achieved with joint 1 angle θ_1 increasing steadily most of the way after staying constant for several steps in the beginning to allow joint 2 angle θ_2 to increase so that link 2 will not collide with obstacle C. θ_2 increased steadily in the first 25 steps and remained nearly constant from the 26th to the 40th step, although it increased slightly in the 34th step to avoid collision with the bottom left corner of obstacle B. From the 41st step till the end, θ_2 decreased rapidly to enable the end-effector to reach the destination, once it had passed obstacle B completely.

Angular velocities Joint 1 velocity $\dot{\theta}_1$ starts off from zero and then remains positive throughout the motion. Joint 2 velocity $\dot{\theta}_2$ is positive from the 1st to 25th step before decreasing to zero for several steps and then increasing to a positive velocity again in the 34th step to move further away from the bottom left edge of obstacle B. Once clear, $\dot{\theta}_2$ becomes negative so that the end-effector can reach the destination. The magnitude of the joint velocities just before reaching the destination are again undesirably high. (see Section 9.3 for a discussion)

Torques The torque in joint 2 τ_2 starts positive to drive link 2 away from the bottom of obstacle C. The torque in joint 1 τ_1 becomes strongly positive to drive link 1 in the 6th step; however, τ_1 is quickly reduced between the 7th and 11th steps to moderate the motion of link 1 since link 2 is still not completely clear of a collision with obstacle C. τ_1 and τ_2 then oscillate about zero to maintain $\dot{\theta}_1$ and $\dot{\theta}_2$ generally constant before τ_2 becomes negative in the 26th step to decelerate link 2 to zero velocity. In the 34th step, τ_2 becomes positive momentarily to avoid obstacle B while in the 35th step, τ_1 becomes positive to quickly get past obstacle B. Finally, τ_1 becomes strongly positive in the 43rd step and τ_2 becomes strongly negative in the 42nd and 43rd steps to hasten arrival at the destination.

Expected return The expected return (Figure 21) increases steadily all the way without decreasing unlike the previous example. This is due to the fact that this motion is relatively straightforward and no 'dangerous' state was entered. The motion can be viewed as climbing from the start position to the higher peak of the evaluation function surface shown in Figure 9.

9 Discussion

9.1 Exploration strategies

There is a trade-off between exploiting the existing policy to maximize known rewards or experimenting with untried actions that have potentially negative consequences but may eventually lead to a better policy.

In the experiments, Williams' [14] stochastic hillclimbing approach was used with gaussian probability density functions to perform exploration in order to find the appropriate torques at each state. However, this method only performs exploration in the early stages of learning when the standard deviations in all states are large. Once the appropriate torques that lead to increasing expected return have been found, the standard deviations in those states decrease and further exploration is discouraged. In a static and time-invariant environment, this is desirable in order to preserve a good learnt policy. However, it is a severe limitation in changing environments which require continuous exploration and reevaluation of existing policies. Thrun [12] proposed several directed exploration strategies including a recency-based technique in which states which have not been visited recently are selected.

9.2 Positions of destination and obstacles

In the experiments, the destination and obstacles were fixed and the agent had to learn how to reach the destination from different starting positions. If the positions of the destination and obstacles are changed, the policy that the agent had acquired needs to be modified.

Several simulations were run in which the destination was changed after the agent had learnt a policy which enabled it to reach a particular destination. Results showed that adapting to the new circumstances was either impossible or would take a very long time.

The agent can also be made more versatile if it acquired the ability for local navigation. The choice of joint positions and velocities as inputs to the agent was to enable the performance of the learning algorithm in solving the task to be investigated without depending too much on the source and representation of the inputs. However, this means that the evaluation function and policy learnt are only valid for the size, orientation and positions of objects that existed during training. This limitation can be removed if inputs are taken from a rangefinder or camera so that when the exact location of objects are changed but perceptual input is similar, the agent can generate the same action.

9.3 Torques and velocities

The graphs in Figures 14, 15, 19 and 20 show that the velocities and torques of the joints were very high just before the destination was reached compared to earlier parts of the motion. This is undesirable in a real physical system and a smooth velocity profile that reaches zero at the destination is preferred.

Another point of concern is the oscillatory nature of the torques produced even when the motion was in parts of the workspace having no obstacles nearby. This wastes energy and can cause 'chattering' in a real physical system.

The first effect can be attributed to the TD procedure itself. Although the exponential component of the first reinforcement rule in the reinforcement schedule awarded a higher than normal payoff if the destination was approached with moderate velocities, its effect was swamped by the contribution of the discount factor γ. The presence of the discount factor encouraged the agent to generate actions which enabled it to reach the destination in as few steps as possible so that a higher return can be attained.

It may be possible to reduce the second effect by having additional reinforcement based on the minimization of the following quadratic measure, known as the *minimum torque* *change* criterion [13]:

$$C_T = \int_0^{t_f} \sum_{i=1}^{n} \left(\frac{dT_i}{dt} \right)^2 dt$$

An improved reinforcement schedule that takes into account stability and energy considerations can also be incorporated.

9.4 Monolithic vs Subsumption architecture

The architecture of the agent described in this report is *monolithic* in the sense that the task was not decomposed into simpler behaviours. Mahadevan and Connell [4] present results which indicate that a *subsumption* approach, where the agent is organized as a set of task-achieving modules and each behaviour is learnt separately, was superior to learning the entire task as one behaviour.

The results described in this report show that good performance in a moderately difficult task can be obtained with a monolithic architecture. However, a subsumption approach may be required in a more complicated robot control task, e.g. locating objects and obstacles or maintaining a smooth velocity profile and stability of the system, in addition to obstacle avoidance and moving to a destination.

10 Conclusion

We have demonstrated how reinforcement learning can be used in an integrated method for controlling a multi-linked manipulator. The agent was able to learn the appropriate torques that should be applied at each joint to enable the end-effector of the manipulator to reach the destination, starting from an arbitrary position in the workspace. The manipulator was also able to avoid collisions with obstacles.

The values of the evaluation function and policy parameters in different parts of the input space were approximated and stored using CMACs. It was seen that the quantizing resolution of the input variables by the CMACs significantly affected the level of performance of the agent.

The learning algorithm in the form of a combination of reinforcement learning and stochastic hillclimbing algorithms was effective, enabling the agent to generate the sequence of real-valued torques necessary to perform the required task.

A Derivation of Robot Dynamics Simulation Model

The dynamics model of the two-linked manipulator used in the simulations described in this report are derived in this section.

A.1 Lagrangian approach

The equations of motion of the robot were derived using a Lagrangian in terms of the kinetic and potential energies.

$$\frac{d}{dt}\left(\frac{\partial L(\dot{\theta}, \theta)}{\partial \dot{\theta}} \right) - \frac{\partial L(\dot{\theta}, \theta)}{\partial \theta} = \tau \tag{11}$$

$$L = K(\dot{\theta}, \theta) - V(\theta) \tag{12}$$

where

$K(\dot{\theta}, \theta)$	is the system kinetic energy
$V(\theta)$	is the system potential energy
$\theta_{2\times1}$	is the position of the robot in terms of the 2 joint angles
$\dot{\theta}_{2\times1}$	is the velocity of the robot in terms of the 2 joint velocities
$\tau_{2\times1}$	are the torques required in the 2 joints for equilibrium

A symmetric matrix $M(\theta)$ i.e. $M(\theta) = M^T(\theta)$ which is a function of θ can be found such that the kinetic energy

$$K(\dot{\theta}, \theta) = \dot{\theta}^T M(\theta)\dot{\theta} \tag{13}$$

giving

$$\frac{\partial L(\dot{\theta}, \theta)}{\partial \dot{\theta}} = 2M(\theta)\dot{\theta}$$

Hence, Equation 11 degenerates to

$$\underbrace{\frac{d}{dt}\left(2M(\theta)\dot{\theta}\right)}_{2M(\theta)\ddot{\theta}+f_1(\dot{\theta},\theta)} - \underbrace{\frac{\partial L(\dot{\theta}, \theta)}{\partial \theta}}_{f_2(\dot{\theta},\theta)} = \tau \tag{14}$$

where $f_1(\dot{\theta}, \theta) = 2\dot{\theta}\frac{d}{dt}(M(\theta))$ and $f_2(\dot{\theta}, \theta) = \frac{\partial L(\dot{\theta},\theta)}{\partial \theta}$, giving

$$2M(\theta)\ddot{\theta} + f(\dot{\theta}, \theta) = \tau \tag{15}$$

where $f(\dot{\theta}, \theta) = f_1(\dot{\theta}, \theta) - f_2(\dot{\theta}, \theta)$. Equation 15 can be rearranged such that the angular acceleration $\ddot{\theta}$ at each joint is obtained when the applied torques and joint positions and velocities are known.

A.2 Model of robot with two joints

The two joints i.e. shoulder and elbow joints are revolute joints and the links move in the same plane as shown in Figure 1. The links of the robot were modelled as two simple beams with uniformly distributed mass and centers of gravity 1 and 2. Two point masses were also considered: 3 to account for the mass of the end-effector and 4 for the mass of the motor driving the elbow joint.

In the following discussion, distance l_1 refers to A–B, l_2 to B–3 and l_3 to B–4; $\underline{v}_B$ refers to the velocity of joint B and $\hat{n}$ is the unit vector $\perp$ to B–3.

Kinetic energy of link 1:

$$\begin{aligned} k_1 &= \frac{m_1}{2l_1}\int_0^{l_1}(\dot{\theta}_1 l)^2 dl \\ &= \frac{1}{6}m_1 l_1^2 \dot{\theta}_1^2 \end{aligned}$$

Kinetic energy of link 2:

$$\begin{aligned} k_2 &= \frac{m_2}{2l_2}\int_0^{l_2} \mid \underline{v}_B + l\dot{\theta}_2\hat{n} \mid^2 dl \\ &= \frac{1}{2}m_2 l_1^2\dot{\theta}_1^2 + \frac{1}{6}m_2 l_2^2\dot{\theta}_2^2 + \frac{1}{2}m_2 l_1 l_2 \dot{\theta}_1\dot{\theta}_2\cos\theta_2 \end{aligned}$$

Kinetic energy of point mass ③:

$$
\begin{aligned}
k_3 &= \frac{1}{2}m_3 \mid \underline{v}_B + l_2\dot{\theta}_2\hat{n} \mid^2 \\
&= \frac{1}{2}m_3 l_1^2\dot{\theta}_1^2 + \frac{1}{2}m_3 l_2^2\dot{\theta}_2^2 + m_3 l_1 l_2 \dot{\theta}_1\dot{\theta}_2\cos\theta_2
\end{aligned}
$$

Kinetic energy of point mass ④:

$$
\begin{aligned}
k_4 &= \frac{1}{2}m_4 \mid \underline{v}_B - l_3\dot{\theta}_2\hat{n} \mid^2 \\
&= \frac{1}{2}m_4 l_1^2\dot{\theta}_1^2 + \frac{1}{2}m_4 l_3\dot{\theta}_2^2 - m_4 l_1 l_3 \dot{\theta}_1\dot{\theta}_2\cos\theta_2
\end{aligned}
$$

Putting into the form of Equation 13, kinetic energy

$$
\begin{aligned}
K(\theta,\dot{\theta}) &= \dot{\theta}^T \begin{bmatrix} m_{11} & m_{12}\cos\theta_2 \\ m_{21}\cos\theta_2 & m_{22} \end{bmatrix} \dot{\theta} \\
\text{where } m_{11} &= \frac{1}{6}m_1 l_1^2 + \frac{1}{2}m_2 l_1^2 + \frac{1}{2}m_3 l_1^2 + \frac{1}{2}m_4 l_1^2 \\
m_{12} &= \frac{1}{2}\left(\frac{1}{2}m_2 l_1 l_2 + m_3 l_1 l_2 - m_4 l_1 l_3\right) \\
m_{21} &= m_{12} \\
m_{22} &= \frac{1}{6}m_2 l_2^2 + \frac{1}{2}m_3 l_2^2 + \frac{1}{2}m_4 l_3^2
\end{aligned}
$$

The potential energy $V(\theta)$ is calculated as follows

$$
\begin{aligned}
V(\theta) &= -\frac{1}{2}m_1 l_1 g\cos\theta_1 - m_2 g(l_1\cos\theta_1 + \frac{1}{2}l_2\cos(\theta_1-\theta_2) \\
&\quad - m_3 g(l_1\cos\theta_1 + l_2\cos(\theta_1-\theta_2)) - m_4 g(l_1\cos\theta_1\cos\theta_1 - l_3\cos(\theta_1-\theta_2)) \\
&= -n_1\cos\theta_1 - n_2\cos(\theta_1-\theta_2) \\
\text{where } n_1 &= l_1 g(\frac{1}{2}m_1 + m_2 + m_3 + m_4) \\
n_2 &= l_2 g(\frac{1}{2}m_2 + m_3) - m_4 l_3 g
\end{aligned}
$$

Thus, the Lagrangian L can be evaluated from Equation 12.

A.3 Equations of motion

The functions $f_1(\dot{\theta},\theta)$ and $f_2(\dot{\theta},\theta)$ were evaluated by differentiating $M(\theta)$ and the Lagrangian as shown in Equation 14.

Finally, rearranging Equation 15 gives:

$$
\ddot{\theta}_1 = \frac{m_{22}[\tau_1 + n_1\sin\theta_1 + n_2\sin(\theta_1-\theta_2) + 2m_{12}\dot{\theta}_2^2\sin\theta_2] - m_{12}\cos\theta_2[\tau_2 - n_2\sin(\theta_1-\theta_2)]}{2[m_{11}m_{22} - (m_{12}\cos\theta_2)^2]} \tag{16}
$$

$$
\ddot{\theta}_2 = \frac{m_{12}\cos\theta_2[\tau_1 + n_1\sin\theta_1 + n_2\sin(\theta_1-\theta_2) + 2m_{12}\dot{\theta}_2^2\sin\theta_2] - m_{11}[\tau_2 - n_2\sin(\theta_1-\theta_2)]}{2[(m_{12}\cos\theta_2)^2 - m_{11}m_{22}]} \tag{17}
$$

The masses and lengths used were:

$$\begin{array}{rclrcl} m_1 &=& 1.6\,\text{kg} & l_1 &=& 0.23\,\text{m} \\ m_2 &=& 0.6\,\text{kg} & l_2 &=& 0.25\,\text{m} \\ m_3 &=& 0.3\,\text{kg} & l_3 &=& 0.05\,\text{m} \\ m_4 &=& 0.5\,\text{kg} & g &=& 9.81\,\text{m/s}^2 \end{array}$$

References

[1] J. S. Albus. *Brains, Behaviour and Robotics*, chapter 6, pages 139–179. BYTE Books, McGraw-Hill, 1981.

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

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

[4] S. Mahadevan and J. Connell. Automatic programming of behaviour-based robots using reinforcement learning. Research Report RC 16359 # 72625, IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, Jul 1990.

[5] J. L. McClelland and D. E. Rumelhart, editors. *Parallel Distributed Processing : Explorations in the Microstructure of Cognition*, volume 1 : Foundations. Bradford Books/MIT Press, Cambridge, MA, 1986.

[6] K. Narendra and M. A. L. Thathachar. *Learning Automata: An Introduction*. Englewood Cliffs, NJ, USA, 1989.

[7] D. Nguyen and B. Widrow. The truck backer-upper : An example of self-learning in neural networks. In *International Joint Conference on Neural Networks*, 1989.

[8] T. J. Prescott and J. E. W. Mayhew. Obstacle avoidance through reinforcement learning. In J.E. Moody, S. J. Hanson, and R. P. Lippman, editors, *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann, San Mateo, CA, 1991.

[9] R. S. Sutton. Learning to predict by the methods of temporal differences. Technical Report TR 87–509.1, GTE Laboratories Inc., 1987.

[10] R. S. Sutton. First results with DYNA, an integrated architecture for learning, planning and reacting. In *Proceedings of the 1990 AAAI Spring Symposium on Planning in Uncertain, Unpredictable or Changing Environments*, 1990.

[11] C. K. Tham and R. W. Prager. Reinforcement learning for multi-linked manipulator control. In *6th International Conference on Systems Research Informatics and Cybernetics*, 1992. To appear.

[12] S. B. Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, School of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, USA, January 1992.

[13] Y Uno, M. Kawato, and R. Suzuki. Formation of optimum trajectory in control of arm movement: minimum torque change criterion. Technical Report MBE86-79, Japan IEICE, 1987.

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