UNIVERSITY of LIVERPOOL

Reinforcement Learning For The Control of Large-Scale Systems

Thesis submitted in accordance with the
requirements of the University of Liverpool
for the degree of Doctor of Philosophy

in

Electrical Engineering and Electronics

by

K.H. Chan , B.Eng, M.Sc.(Eng)

December 2001

Reinforcement Learning For The Control of Large-Scale Systems

by

K.H. Chan

To my Father, whom I owe much to.

A Kind, Devoted and Dear person.

Acknowledgements

I thank my supervisor Professor Q.H. Wu for his intellectual guidance and support throughout this research and for his help when organising this Thesis. I am grateful to The Department of Electrical Engineering and Electronics, The University of Liverpool for providing the research facilities, making it possible for me to conduct this research. Thanks to Lin Jiang for helping me debug and tidy up my programs and Mohamad Sedaaghi and David Stamp for providing the basic latex thesis format. Many thanks to EPSRC for providing the funding enabling this research to be possible from the beginning. Much appreciation goes to my family for their support in everyway possible, my Mum for being so patient with me during the period of my University life. Special thanks to my Father for showing by example that learning new things and accomplishing them against any adversity make us better individuals, it maybe hard work but we will have benefited from it when it is done.

Abstract

Reinforcement Learning For The
Control of Large-Scale Systems

by

K.H. Chan

Learning through repeated trials and updating the desirability of repeating a certain trial is a fundamental learning process in all animals. This type of learning or reinforcement learning can be adopted for use in intelligent engineering control and applications which contain high degrees of complexity, uncertainty and have non-linear characteristics. Reinforcement learning systems have the property of progressively improving their performance based on past experience and require little analytical knowledge of the system for optimisation.

The two main types of reinforcement learning are investigated. The *Learning Automata* (LA), which originates from psychology, and *Temporal Difference* (TD) learning, which owes it's origins to artificial neural networks and hence biology. Initially studies of a TD learning agent for PID control of a third order dynamic system were investigated, then a *Team of Learning Automata* (TLA) was used on the same control problem as a comparison test. The third order system was replaced with a single-machine-infinite-busbar power system and the analysis repeated for the TLA. The single agent learning method was then extended to multi-agent learning control of a multi-machine power system with three synchronous generators as the basis of a large scale system to be investigated. TD learning was then used to evaluate multi-agent PID and fuzzy logic control of the multi-machine power system. To conclude some comparisons between the two different types of reinforcement learning were made to highlight their advantages and limitations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Reinforcement learning enables machines to exhibit learning behavior in new/unknown or partially known environments. The principle idea is that for any system, learning can be achieved by a process of trial and error. Good actions are rewarded and are likely to be tried in future trials, while bad actions are penalised and are likely to be avoided in future trials. The constant trying of actions and discovering their consequences, or exploring, theoretically allows an agent to learn without any prior knowledge of that particular environment or system.

Currently many problems are unsolvable simply because it is too difficult to determine what the learning agent should do (e.g. in a stochastic environment [1] [2]), or the state space is very large that it would be impractical to map concisely. If the learning agent could solve problems through trial and error, then that would be of great practical value. It is on this basis that the many reinforcement learning schemes are able to learn and hence solve problems. Learning deals with the ability of systems to improve their responses based on past experience. Since the ability of living organisms to cope with uncertainty is well known, it is only natural that a lot of effort has been made to incorporate similar features into artificial intelligence systems which can then be

applied to engineering problems. Many subjects can relate to the development of reinforcement learning as well as artificial intelligence (AI) in general. Subjects such as computer science, mathematics, biology, psychology, cybernetics and now even extending to engineering with current emphasis on pragmatic problem solving. It is also interesting to know that reinforcement learning has managed to play an important role in all the above mentioned fields.

1.1 Learning

Learning is the acquisition of useful information or knowledge and can develop in many ways in artificial intelligence (as well as in natural intelligence). Although many learning algorithms have been used to successfully solve various problems, historically, the most important learning paradigm has been that of *supervised learning* [3]. In supervised learning the learner is required to associate pairs of items presented by a "teacher" during training. When later presented with just the first item of the pair, the learner is supposed to recall the second. Two other types of learning have also been essential in the evolution of biological intelligence: *unsupervised learning* and *reinforcement learning*. In unsupervised learning, a system is only presented with a set of examples as inputs. The system is not given any external indication as to what the correct responses should be nor whether the generated responses are right or wrong. Statistical clustering models such as the Kohonen self-organising map [4] have no knowledge of the number of clusters and are examples of unsupervised learning. Basically, unsupervised learning aims at finding a certain kind of regularity in the data represented by the examples. To find meaningful regularity, there must be some redundancy in the input data to describe or classify the examples.

Reinforcement learning is somewhere between supervised learning, in which the system is provided with the desired output, and unsupervised learning, in

which the system gets no feedback at all on how it is doing. In reinforcement learning, the system receives an evaluative feedback, the scalar reinforcement signal $r(t)$ that tells the system whether its output response is right or wrong, but no information on what the right output should be is provided. Since little or no information is given on what the right output should be, the system must employ some random (or structured) search strategy so that the space of plausible and rational choices is searched until a correct answer is found. The reinforcement learning paradigm can further be classified as *associative* and *nonassociative* reinforcement learning.

An associative reinforcement learning scheme requires information in the form of an environmental scalar reinforcement signal as well as a context or input vector. It attempts to form associations between the input and output vector from stimulus-action training pairs (comparable to supervised learning). Thathachar, Phansalker [5] and Barto, Anandan [6], have successfully employed these methods using stochastic learning automata to solve pattern-classification problems. The reinforcement scheme Barto and Anandan used was called an A_{R-P} or *associative reward-penalty* scheme. The algorithm is a combination of a stochastic learning automata algorithm and a pattern classification algorithm based on stochastic approximation [6].

A nonassociative reinforcement learning scheme is one which receives a scalar reinforcement signal from the environment in response to the application of an action selected from an admissible finite set (and no context vector). The objective is to use the reinforcement scalar to guide the choice of action selection towards that action which is deemed optimal in some sense and is comparable to unsupervised learning.

1.2 Supervision or Criticism

Reinforcement learning is an approach to machine intelligence that originates from psychology and contain elements of *Dynamic Programming* [7] [8] [9] [10] [11] and Supervised learning [4] [12] to solve problems that either discipline could not otherwise have solved on their own. Dynamic programming has traditionally been applied to problems involving optimisation and control [13] [9], however dynamic programming is limited in the size and complexity of the problems it can solve. Supervised learning is a general method used for training a parameterised function approximator, such as a neural network. In order to learn, supervised learning requires input-output data pairs from the function to be learned and form a mapping between input-output pairs through training. In a way the learning agent tries to behave as instructed by the environment.

Reinforcement learning on the other hand uses a general evaluative feedback, the reinforcement signal $r(t)$, this gives an overall indication if a chosen action was good or bad, usually to a certain degree based on a probability measure. Thus the reinforcement signal $r(t)$ behaves like a guide or critic and provides criticism as opposed to supervision. Reinforcement learning as a learning control method tries to make the environment behave in a way desired by the learning agent. The goal directed way reinforcement learning behaves is more intuitive and flexible and the advantage of reinforcement learning becomes more apparent when *uncertainty* exists [14] in the system such as in a stochastic system, or when the system is too complex and hence mapping such a large state space completely becomes prohibitively impractical. Other practical concerns are for stochastic systems where the possible output mappings to any given input will not be a simple one to one relation and obtaining data for training (learning) becomes un-feasible, in these situations simple supervised learning can not help.

Reinforcement learning appeals to many researchers because of this generality by learning from trial and error for any given goal. The process of trial and error learning is fundamental to all living animals which reinforcement learning has borrowed from in order to emulate and understand this aspect of animal learning behaviour. From the biological and cognitive points of view, reinforcement learning is much closer to the modern animal learning theory than is supervised learning. Moreover animals can learn extensively about their environments using just external reinforcement signals from the world or other animals, Lin and Lin [15] describes the situation as very similar to learning many high-level intelligent actions such as how to drive a car. It can also be argued that most real life systems are complex and/or stochastic (noisy/random) in nature. The use of reinforcement learning can overcome these obstacles and achieve good performance where other learning methods fail or achieve poor results. Therefore reinforcement learning should be the one of the first considerations in learning control.

1.3 Various Learning Techniques

Throughout its development much insightful research has evolved the basic reinforcement learning paradigm from its roots in psychology and animal learning behaviour. Some of the techniques arise from a need to solve particular classes of problem but many seek to improve the efficiency at which a solution can be found. The main developments in reinforcement learning will be described to give a flavour of these techniques. The learning automata and temporal difference learning are described in more detail in later chapters but are mentioned here for perspective.

1.3.1 Learning Automata

From the very beginnings of reinforcement learning based on psychology the learning automaton was both unique and revolutionary at the time. Introduced by Tsetlin (1973) [16] [17], the learning automaton is a low memory machine for solving selection learning problems due to its nature of trial and error learning which is the heart of all reinforcement learning. The basic learning architecture consists of the learning automaton interacting with an environment which returns a reward signal based on the action selected and its outcome on the current state of the environment. Good actions are rewarded by returning a positive scalar value to reinforce the probability of selecting that good action in future while punishing bad actions by returning a negative scalar value to diminish the probability of selecting that bad action in future. The net result is that the learning automaton learns to select good actions that are beneficial to the environment while avoiding bad actions, the environment being a generic term used to describe any problem space, such as the control of a generator output in a power system network. The learning automaton is a simple concept that so far seems to closely model the fundamental way animals learn when placed in new and unknown environments. Apart from applications of learning automata a lot of work has been done to improve and compare various types of learning schemes used in the updating of the internal states of a learning automata. Narendra and Thathachar [17], Thathachar and Phansalker [5] study the convergence of the learning automata either individually or in structures of teams and hierarchies, Lanctot and Oommen [18] compare different types of variable structure stochastic automata to see which ones converge fastest. In a similar piece of work Rajaraman and Sastry [19] compare the rates of convergence of the pursuit algorithm using finite time analysis for both the continuous and discretised forms of the pursuit algorithm. The pursuit algorithm being a popular update scheme used to adjust the states of a learning

automaton.

1.3.2 Dynamic Programming

Bellman [7] coined the term *dynamic programming* (DP) as a substitute for describing the mathematical theory of multi-stage decision processes. Dynamic programming is a collection of mathematical method for optimising sequential-decisions and shares many principles with other reinforcement learning methods, it is mainly used in stochastic control [8] and problems in which a *Markov Decision Process* (MDP) [10] [20] can be formulated. Common objectives, such as the consequence of long-term and short-term effects when choosing a particular action in a given state need to be addressed, since a "bad" action may be required in the short-term to reach the optimal state in the long-term (*temporal credit assignment* [21]). The primary objective of learning is to construct an optimal action selection policy or simply policy that will maximise the agents performance. DP methods can compute optimal policies given a perfect model of the environment as a MDP [16].

A problem that exists for conventional DP methods occur when the state space is infinite or relatively large. The most difficult aspect of applying DP is often the modelling of the decision task, learning methods particularly reinforcement learning methods have great practical importance. DP methods can solve these problems theoretically, but the computational solution may not be obtainable due to memory limitations, especially when the number of variables is large. The term used to describe this problem by Bellman [7] and others is - "the curse of dimensionality". In order for a learning method to be useful in improving a decision policy it must converge sufficiently rapidly so that the amount of computation required is less than would be required to find an explicit solution using DP. This is the problem that *Incremental Dynamic Programming* IDP methods such as Q-learning and TD try to resolve.

Within DP *policy iteration* and *value iteration* are the most often used methods.

Policy iteration manipulates a *policy* directly, rather than finding it indirectly from the optimal value function. The policy determines which action should be performed in each state and is a mapping from states to actions. The *value function* is a mapping from states to state values and can be approximated using a function approximator (e.g. a multi-layered perceptron trained using backpropagation). The *value* of a state being defined as the sum (or expected sum) of the reinforcements received when starting in that state and following some policy to a terminal state. The value of a state can be changed by selecting different actions. If the state values improve then the policy is adjusted to select that new action when in that state. When no improvements are possible, then the policy is guaranteed to be optimal. The basic method is described below,

1. choose an arbitrary policy

2. compute the value function for the given policy

3. improve the policy at each state

4. if policy is optimal then stop, otherwise go to 1

Another way of finding an optimal policy is to find the optimal value function. The optimal value function can be found using an iterative algorithm called value iteration. If $V(x_t)$ is an approximation of the value function with x_t a state vector, then the approximation of the optimal value function in a given state is equal to the true value of that state plus some error in the approximation, given by

$$V(x_t) = e(x_t) + V^*(x_t) \tag{1.3.1}$$

where $e(x_t)$ is the error in the approximation of the value of the state occupied at time t and $V^*(x_t)$ is the optimal value in that state. Similarly the update approximation at time $t+1$ can be expressed by

$$V(x_{t+1}) = e(x_{t+1}) + V^*(x_{t+1}) \tag{1.3.2}$$

By this definition a relationship exists between the values of successive states, x_t and x_{t+1}. This relationship can be described by the following equations (1.3.3)

$$\begin{aligned} V^*(x_t+1) &= r(x_t) + \gamma V^*(x_t) \\ V(x_t+1) &= r(x_t) + \gamma V(x_t) \end{aligned} \tag{1.3.3}$$

In which the new or updated value is determined by the previous γ discounted value plus some reward (sometimes penalty) feedback response from the environment. The errors of successive states can be described using equation (1.3.4)

$$e(x_t) = \gamma e(x_{t+1}) \tag{1.3.4}$$

$r(x_t)$ is the immediate reinforcement and γ is known as the *discount factor.* The discount factor is used to exponentially decrease the weight of reinforcements received in the future, γ is a number in the range of 0...1 and is used to weight near term reinforcement more heavily than distant future reinforcement where the closer γ is to 1 the greater the the weight of future reinforcements.

In describing the way learning experience is accumulated by the system the term *agent* or *learning agent* is often used. The term agent is best described by Sutton and Barto [16] in which the learner and decision maker are known as the agent. also everything external to the agent in which it interacts with is known as the *environment.* This definition of agent and environment is used in the general literature and also used here.

The total amount of payoff received by the agent over time depends on the number of time periods over which this total is determined, the sequence of actions and states that occurred and the outcomes of any random factors

affecting the payoffs and state transitions. The number of time periods over which the total amount of payoff is determined is called the *horizon* of the decision task. If the horizon is finite (e.g. the state space is small and/or deterministic), then the total payoff is simply the sum of the individual payoffs received at each time period until the task's horizon is reached. If the horizon is infinite (e.g the state space is large and/or stochastic), then this sum may not be finite, a problem solved by using a discount factor γ that allows payoffs to be weighted according to when they occur. The choice of an appropriate discount factor ensures that the weighted sum is finite even for infinite horizon tasks, Sutton and Barto [22] call this *imminence weighting.*

For value iteration, if the function approximator, $V(x_t)$ used to represent $V^*(x_t)$ is a lookup table, then one can find the optimal value function by performing sweeps through state space, using an updating algorithm given by

$$\Delta W_t = \max_u(r(x_t, u) + \gamma V(x_{t+1})) - V(x_t) \tag{1.3.5}$$

Where ΔW_t is the updated reinforcement weight, u is the action performed in state x_t and causes a transition to state x_{t+1} and $r(x_t, u)$ is the reinforcement received when performing action u in state x_t. This assumes that the function approximator is a lookup table, which in many practical problems with large or continuous state space poses problems. Hence one extension to the the basic value iteration method is to use gradient descent on the error function for every update of ΔW_t. The value iteration algorithm can be described by the following

1. Set x to be the current state

2. If $x \in \overline{G}$, then stop

3. Select an action $u \in \overline{A}(x)$

4. Execute action u, as a consequence, the agent receives reward $r(x, u)$

and is in the current state, then the number of steps is incremented i.e. $t = t + 1$

5. Set $\Delta W_t = \max_u(r(x_t, u) + \gamma V(x_{t+1})) - V(x_t)$

6. Go to 1.

Where $\overline{G}$ is the set of goal states and $\overline{A}$ is the set of actions in any state.

1.3.3 Temporal Difference Learning

Temporal Difference (TD) learning was formulated by Sutton (1988) [3] to improve conventional DP methods. Sutton also found links with reinforcement learning and neural networks from research involving ADALINES (adaptive linear elements). Apart from the learning automaton, temporal difference learning is the other major development in reinforcement learning. From the basic TD algorithm Sutton produced a general form for TD by introducing a continuous factor $\lambda \in [0, 1]$ to produce TD(λ), the factor λ is used to give relative weight to the importance of actions in a action sequence model by modelling a forgetting factor when learning.

Two fundamental reasons for the development of TD learning exist, the first is due to its relation with DP and the problems associated with conventional DP methods, mainly the need for a perfect model and thus for large environments the computational requirements become impractical which limits the use of DP. The side effect of this also means that TD learning is suitable for on-line model free learning. The second important point that TD learning addresses is the problem of credit assignment, after a long sequence of actions which ones contribute to the final solution more than the others or relative to the other actions taken in the sequence of choices.

1.3.4 Q-Learning

Another important reinforcement learning method found in the literature is, *Q-learning* developed by Watkins [21] [23] [24] [16] [25] and viewed as a form of model free dynamic programming that allows an agent to act optimally in Markov decision processes. In Q-learning a value function of the form $\mathcal{Q}(s_t, a_t)$ is used and known as the state-action function or simply the Q-function, one step Q-learning can be calculated using equation (1.3.6).

$$\mathcal{Q}(s_t, a_t) \leftarrow \mathcal{Q}(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a \mathcal{Q}(s_{t+1}, a_{t+1}) - \mathcal{Q}(s_t, a_t)] \quad (1.3.6)$$

Q-learning learns the state (s_t) action (a_t) function by following a greedy policy that chooses actions that return the highest value for any current state. Q-learning is guaranteed to converge as long as all state-action pairs are continually updated [16]. From Watkins pioneering work much research has followed in trying to improve and apply Q-learning. Examples such as *Sarsa* [23] [16], *Summation Q-Learning Q(λ)* [23] try to improve on the basic Q-learning method. The term Sarsa is derived from the quintuple $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ that makes up a transition from one state-action pair to the next based on equation (1.3.6) with the assumption that if s_{t+1} is terminal, then $\mathcal{Q}(s_{t+1}, a_{t+1})$ is defined as zero [16]. Summation Q-learning updates the Q-function based on the expected return given the current action probability instead of the maximum value of $\mathcal{Q}(s_{t+1}, a_{t+1})$ [23]. Q(λ) is a method of combining Q-learning and TD(λ) by Peng and Williams [23]. One-step Q-learning makes minimal use of information received by the system since only a single prediction of the Q-function value is updated for a single state-action pair per time step. TD(λ) methods offer a way of allowing multiple predictions to be updated per time step and hence speeds up convergence. The TD(λ) method uses a standard one-step Q-learning update to improve the current prediction of the Q-function and then using the temporal differences between successive greedy predictions to update it from there on.

1.3.5 W-Learning

W-learning consists of a hierarchical structure of learning agents, each specialised to optimise/control one aspect of the complete learning system. A simple example to explain W-learning is to use a two-link robot arm with a manipulator performing a pick up task in an environment with obstacles. The objective of the robot arm is thus to successfully pick up the target object while avoiding the obstacles. For W-learning in this case there are two main tasks to optimise/control, first the robot arm is required to reach the object then pick it up and secondly the arm has to avoid the obstacles along the way. There will be two learning agents in this problem, one of the agents has the goal, reach the target and pick up the object, the other learning agent has the goal, avoid the obstacles. At each time step the competing agents each suggests an action, but only one action is executed. Which agent is obeyed changes dynamically. Each agent can function in the absence of the others and will try to reach its own goal, but each is also frustrated by the presence of the other agents. The switching between agents can be complex and a better solution would be to have a simple switch which the agents can use to organise themselves from.

W-learning was developed from Q-learning by Humphrys [26]. The simplest scheme involves each action having a measure of importance or weight $\mathcal{W}$ (hence W-learning) when in a given state and then executing the action with the largest $\mathcal{W}$. To be exact the learning agents are $A_1, ..., A_n$, at each discrete time step all the agents observe the world to be in some state x. Each agent A_i then suggests an action a_i with weight $\mathcal{W}$, the action $a_k(x)$ is then executed where

$$\mathcal{W}_k(x) = \max_{i \in 1,...,n} \mathcal{W}_i(x) \tag{1.3.7}$$

The agent A_k is called the *leader* [26] in the competition for state x at the moment. The actions $a_i(x)$ are those that the agent has learnt to take to pursue its goals. This is where the W-values $\mathcal{W}_i(x)$ come from and how they

change in response to not being obeyed. Schemes using "importance" values are common in multi-behavior models but are normally hand designed, generating them continuously and on-line is a simple task for reinforcement learning.

1.4 Applications of Learning Methodologies

The development of reinforcement learning has provided many ideas, new algorithms and applications. The flexibility of reinforcement learning has meant that a lot of previous work and applications are very diverse and can sometimes seem to be unrelated. The use of reinforcement learning can cover aspects such as process control to higher cognitive action processes and even emotional states of animal psychology. The broad range of examples available that use reinforcement learning means that reinforcement learning is not a specialisation in any one area of research. Recently a lot of work in diverse fields have now been bridged by reinforcement learning.

In the following examples a trend from simple single learning agents towards more complex distributed network or hierarchy of learning agents is evident and seems to be the way future reinforcement learning control is heading.

1.4.1 Applications of Learning Automata

Being the earliest reinforcement learning method, the learning automaton has been applied to a wide range of control problems. The original reason by Tsetlin for developing the learning automaton was in order to solve the *n-armed bandit* problem [16] [17]. The number of learning systems applications has also increased with the advent of highly integrated computers which makes technology cost-effective. Applications vary from the simple maze problems in which a 'mouse' learns to find the 'cheese' situated in the maze to playing complex sequential two player games, such as chess. Narendra and Thathachar

[17] provides extensive theory for using learning automata to game playing complex sequential games either using single learning automaton or interconnected learning automata and even developing the theory to include games with N-players, where N> 2. It is also interesting to note that Narendra and Thathachar [17] mentions potential applications regarding information routing in networks, priority assignment in a queuing system (similar to task scheduling) and image compression.

When the number of actions increases, the behaviour of a single automaton will be slow. This problem can be avoided by using a hierarchical structure of automata [5], [27], [28]. In this structure, the automaton contained in the first level of the hierarchy selects randomly an action which activates an automaton in the second level of the hierarchical structure of automata. In turn, this automaton selects randomly an action and activates an automaton in the third level of the hierarchy. This procedure is repeated until the activated automaton in the last level of the hierarchy selects randomly an action which corresponds to the output of the hierarchical structure of automata (environment input).

Wu and Pugh [28] [29] used a team of learning automata in order to learn the optimal controller parameters of a PID controller in order to generate power efficiently from a synchronous turbo-generator in a single machine infinite bus-bar power system. Frost *et al* [30] used a set of continuous learning automata to control a vehicles roll dynamics with semi-active suspension. Hobday, Wu and Gordon [31] used learning automata to optimise a fuzzy logic controller in a multi-arm robot manipulation task, the automata would learn to co-ordinate the efforts of each arm when performing a particular task. Barto and Anandan [6] used stochastic learning automata with the associative reward-penalty (A_{R-P}) update scheme in order to compare existing supervised learning pattern recognition methods against learning automata methods. Work related to pattern recognition was also studied by Thathachar and Sastry [27] in which a team of mutually cooperating learning automata perform a probabilistic search

through the space of unknown pattern classes in order to learn an optimal classifier. Gullapalli [32] compares discrete action space problems and problems requiring continuous action space problems using a multi-linked robot arm for the evaluation of successful control. Li *et al* use interconnected learning automata to coordinate and control a multi-machine power system using dynamic quadrature boosters to enhance the stability of electric power generated.

Apart from direct applications of learning automata, other uses for the learning automata is in the form of synthesising neural networks by Najim, Chtourou and Thibault [33]. Instead of the more common use of learning automata for optimisation of control processes Najim, Chtourou and Thibault apply stochastic learning automata with variable structures to solve the optimisation problem associated with the estimation of neural network parameters.

1.4.2 Applications of Dynamic Programming

The model dependence of dynamic programming makes it difficult to apply to stochastic control systems which are the main applications for other reinforcement learning. Not including IDP methods such as Q-learning and TD(λ) learning which were developed to overcome the model dependence of classical DP methods and are therefore much more popular, it is rare to find practical optimisation applications using DP.

1.4.3 Applications of Temporal Difference Learning

Since the realisation of the limitations posed by DP, TD learning has become a popular choice of reinforcement learning scheme for the solution of many problems.

One early application to TD learning was to test it on the inverted pendulum problem, Barto Sutton and Anderson [34] used TD learning to successfully keep a simulated inverted pendulum from falling over, while an actual mechan-

ical test rig was built by Jervis and Fallside [35] to evaluate TD learning's control performance. Of all applications one of the most well known using TD learning is Tesauro's TD-Gammon player (1992) for Backgammon [16] [36] and possible the first ever application of TD learning by Samuel (1959) in his celebrated checker (draughts) playing programming [3] [16]. Both these game playing examples use TD learning to learn optimal game positions by learning through self play and observing the consequences. Game playing is a popular evaluation and comparison method for reinforcement learning techniques, Yee *et al* [37] used the game of tic-tac-toe (noughts and crosses) to test a TD learning agent's ability to identify and define concepts that improve its performance in performing tasks in general.

Barto, Sutton and Watkins [38] apply TD learning to solve a route-planning problem in which from any arbitrary start position in the environment the TD learning agent seeks to find the most optimum path to reach a goal state while learning to avoid obstacles in the environment. The classic mountain-car problem was solved using TD learning by Sutton [39], while Tham and Prager [40] worked on applying TD learning to control a multi-linked robot arm for manipulating objects in an environment with obstacles.

Lin and Kan [41] implement a fuzzy command acquisition network for speech recognition using TD reinforcement learning to update the fuzzy weights in a fuzzy predictor whose task is to learn by translating verbal commands into the desired actions. Boyan [42] also discusses how TD learning is used to train a neural network for phoneme recognition in speech.

From single agent learning applications to multi-agent learning, Crites and Barto considered the problem of optimising elevator performance in a tall office building [43]. The simulated system consisted of a building with 10 levels and 4 elevators, each with a capacity to hold 20 passengers. Singh and Bertsekas [44] looked into channel allocation in cellular telephone systems using TD learning to maximise services in the dynamic network. Both of these stochastic dynamic

optimisation problems demonstrate the versatility of reinforcement learning in large-scale systems and the type of problem that they are suitable for solving.

1.4.4 Applications of Q-Learning

Q-learning and its derivatives have commonly been used in robot navigation tasks, often obstacle course or maze environments are designed to test the "robotic learning agent". Rummery [23] compared Q-learning, Sarsa and Q(λ) by implementing a multi-layer perceptron neural network and using each Q-learning method as an update rule for the weights in the neural network in order to learn and navigate an obstacle course. Those findings show that Sarsa was the most successful in producing the most number of robotic agents that completed the task as well as having the quickest convergence times. Robot navigation tasks were also studied by Koenig and Simmons [45] in which the shortest path in a maze problem with the added complexity of one way paths being introduced in the environment. Of the classic test problems in machine learning Q-learning has been tried on the *n-armed bandit* problem by Duff [46], while Jervis [47] applied Q-learning to the inverted pendulum problem. Sutton and Singh [48] [16] used Sarsa to try and solve the mountain-car problem in which a car is in the valley between two mountains but does not have sufficient power to directly drive up a forward facing mountain side but must learn to gain momentum by reversing (i.e. choose a less optimal action) in order to eventually drive up the mountain side.

Examples of practical applications with relevance to engineering found in the literature exist. Tham and Prager use Q-learning to decompose the task of a robotic manipulator arm with "Q-modules" built from *Cerebellar Model Articulation Controllers* (CMACs) [49]. Each Q-module is in charge of the position of a part of the whole robotic arm, for a two link flexible arm with manipulator three Q-modules are required, one for position of the upper arm,

one for position of the forearm and one for position of the hand (manipulator). The problem to be solved was to position the manipulator in order to perform a task such as pick up an object while learning to avoid various objects placed in certain parts of the environment. Clausen and Wechsler developed *Quad Q-Learning* and applied it to image compression. Quad-Q-learning is a method that is suitable for problems that can be solved by dividing the problem state space into smaller more easily solvable sub-problems, Clausen and Wechsler use the term"divide and conquer" [50]. Finally Boyan and Littman use a modified form of Q-learning labelled Q-routing to generate efficient packet routing in a dynamically changing communication network, in which a simple 36 node irregularly connected network was simulated. The principle feature being that each node only uses local information provided by its immediate neighbour in order to make optimal actions when passing on packets of data. Although this problem is similar to a robot navigation task, sometimes the optimal route a data packet takes will not always be based on the shortest route and is more often based on the quickest time which may be a longer route. Piater [51] recognised a similar application for route planning based on traffic light control in a road junction, where an action by the learning agent is to set the traffic lights to a certain combination of settings. the learning agent tries to reduce delays waiting at traffic lights and improve traffic flow in general.

1.4.5 Applications of W-Learning

Humphrys developed and applied W-learning to solve the *ant-world problem* [26]. Almost all previous work that use multiple reinforcement learning agents in teams or hierarchies have concentrated on the co-operation between each learning agent in the team or hierarchy. Humphrys's approach was to think of the complete learning agent as a series of competing tasks, such that for the ant-world problem, the body of the learning agent or 'ant' consists of a number

of competing priorities such as seek food, find shelter and avoid predators while it moves in its environment. A similar proposition by Humphrys was to have a 'robot dog' mainly to detect intruders in a house but also have other priorities, such as fire detection vacuuming the room, re-charge its batteries, empty the vacuum bag when the need arises or priorities change. Many real and interesting systems in life posses multiple parallel and conflicting goals, among which the attention of the learning agent as a whole must constantly switch, this is the *action-selection* problem that the agent must face. W-learning is an interesting phase in reinforcement learning and seeks to approach a problem from another perspective only time will tell if it becomes as popular as the better established TD and Q-learning.

1.5 Learning Control of Large-Scale Systems

The concept of distributed learning and learning embedded in systems of interaction is a relatively new development with great promise. With emphasis on the construction of information through multi-agent interactions, a system interaction approach of learning offers a shift in perspective, from an emphasis on the content of learning to the emergent process of learning. Multi-agent learning becomes something accomplished with others rather than alone and the structure of role relationships between agents and environments which sustains learning should be carefully examined.

Reinforcement learning algorithms, such as the Sutton's *Temporal Difference* (TD) [3] were inspired and motivated by animal learning behaviour and classical conditioning [22]. Subsequently the TD algorithm proved highly successful in game theoretic examples such as playing Chess and Backgammon. Later TD methods also proved useful in solving prediction and control problems, balancing the inverted pendulum being an early example [34]. Now TD methods and the other incremental dynamic programming schemes [52]

are emerging as useful tools for prediction and control of general adaptive real-time systems or stochastic systems using embedded learning agents. Researchers have customarily focused their attention on upon asymptotic learning of maximally-efficient strategies, and not on the optimal learning of these strategies [46], but as will be seen most learning agents can still effectively solve problems without having to learn a completely optimal strategy for most cases.

According to Duff, the most successful applications have been to large, complex problems for which the computational effort required by traditional engineering methods would be unduly burdensome and perhaps unjustified, given that in many cases only approximate models of the underlying systems are known [46]. With this remark we can take the case of large, complex problems to include large-scale systems such as distributed power system networks and use reinforcement learning to optimise the efficiency of such a system.

1.5.1 Connectionist Systems

Boyan [42] defines *connectionism* as

> The study of computational models inspired by models of the brain

and links connectionism to artificial intelligence (AI), machine learning (ML) and statistical pattern recognition (SPR). There is growing interest in the artificial intelligence community for simulated or artificial neural networks. The main reason for this is to avoid the limitations of conventional serial symbolic processing by investigating parallel systems, such as artificial neural networks and teams of interconnected learning automata. Since the permanent knowledge in such systems is stored in the weights associated with the connections rather than in memory cells, such architectures were called *connectionist* systems. The connectionist system adapts to control problems by having its weights changed to improve some performance criterion. The changing of the

weights is governed by the learning method or algorithm to achieve an optimal (or suboptimal) performance in the controller.

The use of learning algorithms and connectionist systems have been successfully applied to areas such as

- Modelling
- System Identification
- Prediction
- Pattern Classification/Recognition
- Adaptive Control

1.5.2 Large-Scale Systems

Two classic applications of the TD algorithm have been Samuel's Draughts player [3] and Tesauro's Backgammon player [24], both used a TD algorithm to update a neural network to learn the corresponding game through competitive play. Recently the TD learning methods and the closely related Q-learning algorithm (by Watkins [24]) have emerged as useful methods in solving real engineering problems. Early work with TD learning for practical problem solving and control tasks began with route planning and navigation for simulated robotic agents in maze solving problems. The usefulness of TD reinforcement learning for optimising control performance in general has led to increasing research into TD learning for the intelligent control of distributed systems, such as multi-arm robots; channel allocation in cellular phone systems [44]; packet routing in information networks [53] and improving elevator performance in large office buildings [43].

What contributes to a large-scale system? Since no formal definition has been found in the literature and relating to the applications investigated so

far, a convenient description would be to imply any system containing enough uncertainty as to be impractical to be solved using conventional or classical methods used so far in learning or adaptive control in engineering. Hence the current need for intelligent learning and control in some form embodied by a learning agent, such as the learning automata or neural network in order to make good control decisions when faced with uncertainty.

From previous examples such as efficient information routing in networks and improving elevator performance in large office buildings, it is apparent that in order to make a realistic model for a learning agent to learn all the possible states and optimal actions would become a next to impossible task requiring unrealistic amounts of processing power. Reinforcement learning methods provide a more elegant approach to solving the uncertainty problem and doesn't require a model either complete or partial in order to learn and make good control decisions. The added benefit with a model free learning system is that the learning agents are not limited by an often approximated model and can be easily retrained if changes occur frequently such as for different fault conditions in large-scale power systems.

Further expanding upon the practical applications the use of TD and in particular multi-agent learning for power system control has been investigated in this research.

1.5.3 Learning Control

The fundamental basis of all reinforcement learning involves interaction between the learning agent and its environment [3]. The role of the environment is like that of a teacher or guide and the interaction between learner and teacher can be described as experience. In the case of power system control, the accumulated experience is used to make better control decisions by learning what are good and what are bad actions to take for any state or given situation in

that environment. Eventually this will lead to improved control decisions and power system performance based on an optimal parameterised controller, such as the popular *Proportional Integral and Differential* PID controller or *Fuzzy Logic Controller* FLC. By using temporal difference reinforcement learning [3] it is possible to continuously learn and update these parameterised controllers making this multi-agent learning method suitable for on-line parameter optimisation.

TD learning originally developed by Sutton [3] is based on conventional dynamic programming methods and the Widrow-Hoff rule for neural network learning [3, 34]. One of the most important aspects of learning is that of assigning credit. In reinforcement learning the reinforcement feedback signal $r(t)$ gives an immediate (or short term) indication of how good or bad the decision was for a particular state. However this is insufficient in many real problems and the long term values of each state must be estimated or known for efficient prediction and/or control, that is the credit assignment problem.

Unlike most prediction and learning methods in which credit is assigned by means of the difference between predicted and actual outcomes, the TD method assigns credit by using the difference between temporal successive predictions. The TD method also has the advantage of incremental learning and unlike conventional dynamic programming it does not need to finish a long sequence of actions before updating it's knowledge base, but can learn new knowledge after successive predictions, this also reduces the overall computation and memory requirements for learning.

In the proceeding chapters two important reinforcement learning methods will be described in detail, the learning automata and temporal difference learning. Both are studied and used for intelligent control of large-scale power systems, various single agent and multi-agent learning architectures have been employed. The learning automaton is a direct development from psychology and is one of the earliest reinforcement learning methods to be formulated.

Thus it contrasts with TD learning which is a relatively new development in the history of reinforcement learning.

1.5.4 Power System Problem Formulation

The application of multi-agent reinforcement learning is explored for the control of large-scale power systems. This involves the application of reinforcement learning such as temporal difference learning to optimise a controller such as a fuzzy logic controller in a distributed power system. Ultimately a three machine power system has been simulated to evaluate the dynamic performance of the generators as well as observing the multi-agent learning control performance. Initial preliminary studies were performed using a single generator to a single load connected by a single transmission line. This basic power system is also known as a *Single Machine Infinite Busbar* (SMIB) system. The SMIB was later expanded in stages to the final multi-machine multi-agent reinforcement learning control system. In all cases the synchronous generator outputs such as the terminal voltage or speed deviation were optimised to provide good damping characteristics over a wide range of operating conditions. Complete details of the synchronous generators and power distribution network are described later in the relevant chapters.

1.6 Contribution of the Thesis

A review of the literature regarding reinforcement learning has been undertaken, mainly to study the methods and applications undertaken by previous research. The viability of using reinforcement learning to distributed large-scale systems, in particular the power system network optimisation problem was proposed and presented to apply multi-agent reinforcement learning. Initial tests used a 3rd order dynamic system to validate the TD learning and

learning automata methods for optimising control.

Research then continued using learning automata and a single machine infinite busbar system (SMIB) but as work progressed and more literature became available it was understood that developments in TD learning would provide a better opportunity for applying distributed multi-agent reinforcement learning. Also the use of adaptive heuristic critics (AHCs) as neuron-like learning elements, the TD learning neural network was developed to solve the multi-agent reinforcement learning problem.

Reinforcement learning provided a means of automatic and continuous learning of control parameters in conventional parameterised controllers such as the PID or fuzzy logic controller. This also made it suitable for on-line learning control of the power system network. The final evaluation resulted in using TD learning control of PID control followed by similar experiments with a fuzzy logic control in a multi-machine power system, both were investigated and implemented in simulation studies.

It has been demonstrated using computer simulations that each temporal difference reinforcement learning can learn to solve global optimisation problems using only local information and without directly communicating knowledge between agents but use the global reinforcement feedback signal $r(t)$as a guide to optimising the global problem. Each learning agent is in effect an autonomous intelligent system and by placing these autonomous intelligent systems at strategic points in the problem environment, the de-centralised nature of learning also provides a certain amount of redundancy in the learning control problem. Finally some of the practical aspects of reinforcement learning are also discussed.

Chapter 2

The Learning Automata

Early reinforcement learning ideas originated from interest by psychologists in animal behaviour. Later this was developed into the learning automaton by Tsetlin [16] [17] to solve the *n-armed bandit* problem, the learning automaton observed the system states and used a feedback reward signal $r(t)$ to evaluate the observed states in order to update a policy based on a probability measure for improving future actions.

Essentially the learning automaton, comprises a decision maker (the automaton) and an environment (the system or plant to be controlled/optimised), these are connected in the classic feedback configuration shown in Figure 2.1. The automaton has a finite number of actions and corresponding to each action, the response of the environment can be either favourable or unfavourable with a certain probability. Using the theory of Markov processes the asymptotic behaviour of the automaton can be established.

The objective in the design of the automaton is to determine how the choice of the action at any stage should be guided by past actions and responses. The important point is that the decisions must be made with little knowledge of the environment. The environment may have time varying characteristics, or the decision maker may be part of a hierarchical decision structure but unaware

of its precise role in the hierarchy. Alternatively, the uncertainty is due to the fact that the output of the environment is influenced by the actions of other agents unknown to the decision maker. In all cases the automaton must be designed to improve some overall performance function. Both deterministic and stochastic rules for choosing the action at any stage are of interest. In the latter case the automaton updates the probabilities of the various actions on the basis of the information received.

2.1 The Environment

A learning automaton system is a sequential machine characterised by a set of internal states, a set of input actions, a state probability distribution, a reinforcement scheme and an output function. These aspects will be described with the aid of Figure 2.1.

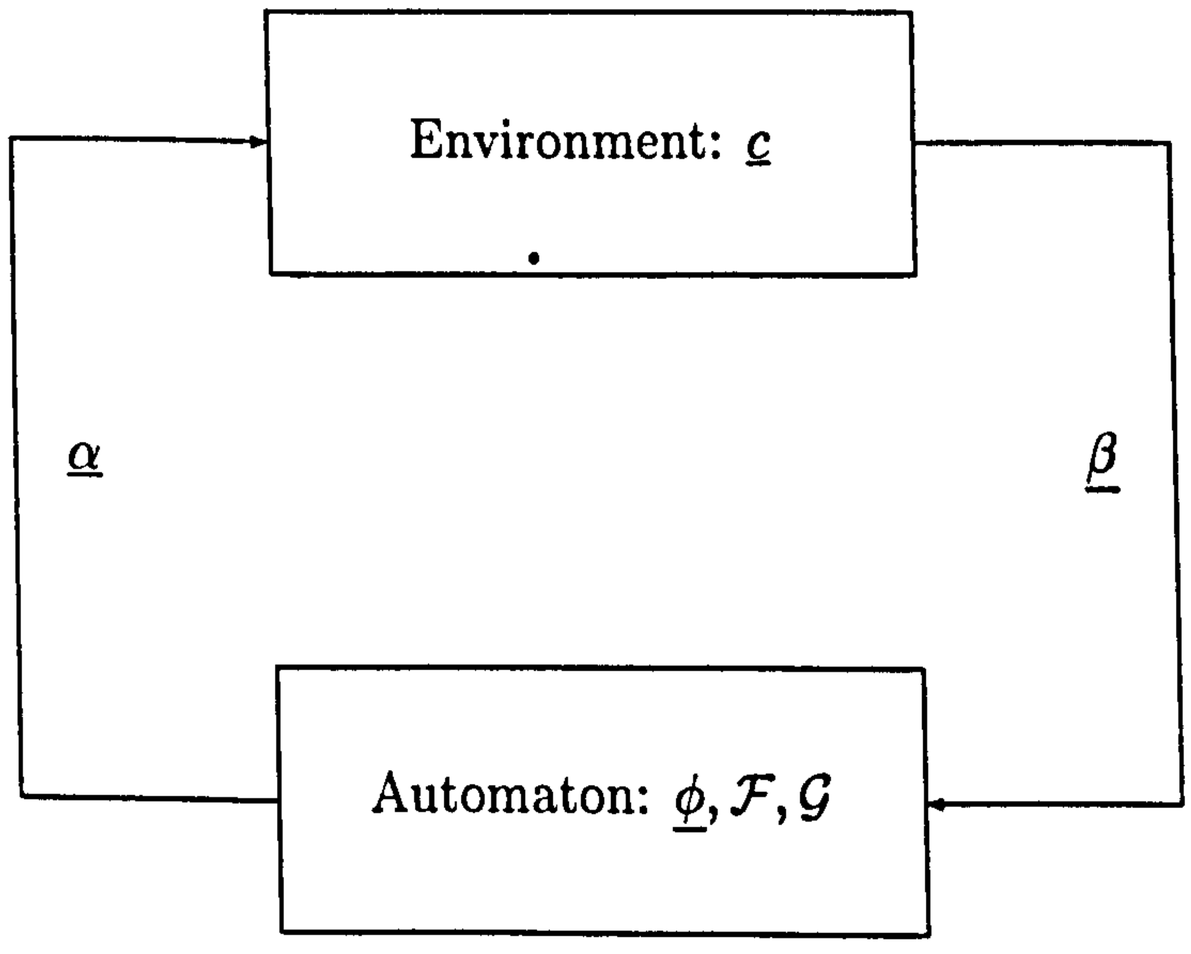

Figure 2.1: The learning automaton.

The environment represents the system which communicates with the learning system and supplies it with information. In the context of automatic control, the environment corresponds to the process to be controlled. In optimisation terms the environment represents the realisation of the function to be optimised. The environment is said to be stationary if the penalty probabilities are not dependent on the time, otherwise it is said to be nonstationary, for example periodic, slowly varying or random. The environment can be defined mathematically by the triple $\{\underline{\alpha}, \underline{c}, \underline{\beta}\}$.

1. $\underline{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_r\}$ represents a finite input set of actions.

2. $\underline{c} = \{c_1, c_2, ..., c_r\}$ represents a set of penalty probabilities for indicating the value of the input actions $\underline{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_r\}$

3. $\underline{\beta} = \{\beta_1, \beta_2\} = \{0, 1\}$ represents a binary output set of feedback rewards for the automaton

Each element of c_i of $\underline{c}$ corresponds to one input action α_i The input $\alpha(n)$ to the environment belongs to $\underline{\alpha}$ and may be considered to be applied to the environment in discrete time steps n where $n = 0, 1, 2, ...$ The output $\beta(n)$ of the environment belongs to $\underline{\beta}$ and can take on one of two values β_1 and β_2. For mathematical convenience β_1 and β_2 are chosen to be 0 and 1 respectively. An output $\beta(n) = 1$ is identified with a failure (unfavourable response) and $\beta(n) = 0$ with a success (favourable response). The element c_i of $\underline{c}$ which characterises the environment is given by

$$Pr(\beta(n) = 1/\alpha(n) = \alpha_i) = c_i \qquad (i = 1, 2, ..., r) \qquad (2.1.1)$$

Consequently, c_i represents the probability that the application of an action α_i to the environment will result in a penalty output.

The output set of the environment has thus far been assumed to be binary for simplicity. In some real physical environments this may be unrealistic

and hence the output set will need modifying. Models in which the output can take only one of two values, 0 or 1 for example, are referred to as P-models. An arbitrary threshold may be necessary to convert the actual outputs of the system into binary outputs, thereby discriminating between favourable and unfavourable responses of the environment to a given action. A further generalisation of the environment allows finite output sets with more than two elements to take discrete values in the interval [0,1]. This is achieved by the normalisation and quantisation of the performance index, such models are referred to as Q-models. When the output of the environment is a continuous random variable which assumes values in the interval [0,1], it is referred to as an S-model. Q- and S-models provide improved discrimination of the nature of the response of the environment to a given action and hence are of greater practical utility. However the concepts developed here have concentrated on using the P-model environment for simplicity, the same concepts can easily be extended to the Q- and S-model.

2.2 The Automaton

The learning automaton collects data from the environment and processes it to achieve a desired goal. They perform a kind of mapping between the inputs and outputs of the random environment where they operate. They can be compared to adaptive systems, in which the behaviour of the system is slightly improved at every time by estimating in real time the parameters of the plant model or controller to attain a specified goal. In a learning automata, the probability distribution is adjusted using a reinforcement scheme (an adaptation mechanism which is at the heart of a learning automaton) to achieve the desired objective.

The automaton can be described mathematically by the quintuple $\{\underline{\phi}, \underline{\alpha}, \underline{\beta}, \mathcal{F}, \mathcal{G}\}$, note that the output of the environment is also the input of the automa-

ton, similarly the output of the automaton is the input to the environment, see Figure 2.1.

1. *The state* of the automaton at any instant n, denoted by $\phi(n)$ is an element of the finite set $\underline{\phi} = \{\phi_1, \phi_2, ..., \phi_s\}$

2. *The output or action* of an automaton at the instant n, denoted by $\alpha(n)$, is an element of the finite set $\underline{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_r\}$

3. *The input* of an automaton at the instant n, denoted by $\beta(n)$, is an element of a set $\underline{\beta}$. This set could be either a finite set or an infinite set, such as an interval on the real line. Thus $\underline{\beta} = \{\beta_1, \beta_2, ..., \beta_m\}$ or $\underline{\beta} = \{(a, b)\}$

4. *The transition function* $\mathcal{F}(.,.)$ determines the state at the instant $(n+1)$ in terms of the state and input at the instant n, $\phi(n+1) = \mathcal{F}[\phi(n), \beta(n)]$, or $\mathcal{F}(.,.)$ is a mapping from $\underline{\phi} \times \underline{\beta} \rightarrow \underline{\phi}$ and could be either deterministic or stochastic

5. *The output function* $\mathcal{G}(.)$ determines the output of the automaton at any instant n in terms of the state at that instant, $\alpha(n) = \mathcal{G}[\phi(n)]$, or $\mathcal{G}(.)$ is a mapping $\underline{\phi} \rightarrow \underline{\alpha}$ and is again either deterministic or stochastic.

Basically, the automaton takes in a sequence of inputs and puts out a sequence of actions. The working of the automaton as the observation time n successively takes values over the set of nonnegative integers $0, 1, 2, ...$ can be conceived as follows. Given initial state $\phi(0)$, the action $\alpha(0)$ is defined by $\mathcal{G}(.)$. With the knowledge $\beta(0)$ of the input and the transition function $\mathcal{F}(.,.)$ the next state $\phi(1)$ is determined. When these operations are performed recursively, the state sequence and the action sequence are obtained for any given input sequence. Note that the state and the action at any instant n depend on only the state and input at the previous instant $n-1$ and not on other

past states or inputs. Knowledge of the environment is therefore not essential to learning, making the learning automaton a very powerful engineering tool. The main limiting factor is the time required by the automaton to learn, this may limit the kind of applications that are suitable for the learning automaton, assuming of course that the convergence to the desired goal exists.

The automaton is called a deterministic automaton if $\mathcal{F}(.\ ,.\ )$ and $\mathcal{G}(.\ )$ are both deterministic mappings. In such a case, for any given initial state and input the succeeding state and action are uniquely specified. If $\mathcal{F}(.\ ,.\ )$ or $\mathcal{G}(.\ )$ is stochastic, the automaton is called a stochastic automaton. In this case there is, in general, no certainty concerning the states and action that follow a given initial state and input sequence; one can only consider probabilities associated with successive states and actions.

2.3 The Stochastic Automaton

At present a lot of research is interested in systems with much uncertainty, this was a result of systems becoming more and more complex as technology advanced. New approaches to control engineering problems were needed, the stochastic learning automaton being one of them.

Consider a stochastic automaton in which at least one of the two mappings $\mathcal{F}$ and $\mathcal{G}$ is stochastic. If the transition function $\mathcal{F}$ is stochastic, given the present state and input, the next state is random and $\mathcal{F}$ gives the probabilities of reaching the various states. Thus $\mathcal{F}$ can be specified in terms of the conditional probability matrices $F(\beta_1), F(\beta_2), ..., F(\beta_m)$ where each $F(\beta)$ for $\beta \in \underline{\beta}$ is an $s \times s$ matrix associated with an input symbol β and whose entries are given by

$$f_{ij}^{\beta} = Pr\{\phi(n+1) = \phi_j \mid \phi(n) = \phi_i, \beta(n) = \beta\}$$
$$i = 1, 2, ..., s \quad j = 1, 2, ..., s \quad \beta = \beta_1, \beta_2, ..., \beta_m \tag{2.3.1}$$

Thus f_{ij}^{β} represents the probability that the automaton moves from state ϕ_i to state ϕ_j following an input β. For the P-model case $\beta = \{0, 1\}$ is a binary output from the environment regarding a favourable response or unfavourable response.

The stochastic mapping $\mathcal{G}$ can be similarly represented by a conditional probability matrix G of dimension $s \times r$ whose entries are given by

$$g_{ij} = Pr\{\alpha(n) = \alpha_j \mid \phi(n) = \phi_i\}$$
$$i = 1, 2, ..., s \quad j = 1, 2, ..., r \tag{2.3.2}$$

Hence, g_{ij} denotes the probability that the state ϕ_i corresponds to action α_j.

Since f_{ij}^{β} and g_{ij} are probabilities, they lie in the closed interval [0, 1]. Further, starting from an initial state ϕ_i, the automaton necessarily has to go to one of the s states at the next instant. Hence, to conserve probability measure we have

$$\sum_{j=1}^{s} f_{ij}^{\beta} = 1 \text{ for each } \beta \in \underline{\beta} \text{ and } i \tag{2.3.3}$$

Similarly,

$$\sum_{j=1}^{r} g_{ij} = 1 \text{ for each } i \tag{2.3.4}$$

The equations imply that the sum of row entries in each of the matrices is unity or the matrices F and G are stochastic

2.3.1 Fixed Structure and Variable Structure Automata

If the conditional probabilities f_{ij} and g_{ij} are constant i. e. , independent of n and the input sequence, then such a stochastic automaton is said to be a *fixed-structure stochastic automaton.* Sometimes it is necessary or useful to update the transition probabilities f_{ij} at each n on the basis of the input at that instant. In this case, the automaton is called a *variable-structure stochastic automaton.*

The main characteristics of the automata can be summarised as follows

- For a deterministic automaton, the transition matrices $F(\beta)$ consist of elements that are only 0 or 1
- For a fixed-structure stochastic automaton, the elements of $F(\beta)$ are constants taking values in the interval $[0, 1]$ and each $F(\beta)$ is a stochastic matrix
- In the case of a variable-structure stochastic automaton, the elements of $F(\beta)$ are in $[0, 1]$ but are no longer constants as they are updated with n.

2.4 Variable Structure Stochastic Automata

The theory of Markov processes forms the principle vehicle for the study of variable-structure automata. To define a general Markov Chain, it is convenient to introduce a transition probability or stochastic matrix which contains the probabilities of transition between successive states $p = \{p_{ij}\}$ with i, j varying over a finite (or countable) state space such that

1. $p_{ij} \geq 0$ for all i, j
2. $\sum_{j \epsilon s} p_{ij} = 1$ for all i

2.4.1 Reinforcement Schemes

In general terms a reinforcement scheme can be represented either by

$$p(n+1) = T[p(n), \alpha(n), \beta(n)] \tag{2.4.1}$$

or by

$$f_{ij}^{\beta}(n+1) = T'[f_{ij}^{\beta}(n), \phi(n), \phi(n+1), \beta(n)] \tag{2.4.2}$$

where T and T' are mapping functions.

Reinforcement schemes are generally classified on the basis of

1. the asymptotic behaviour of a learning using the scheme e. g. , expedient, ϵ-optimal or optimal, or

2. the nature of the mapping T or T' e. g. , linear, nonlinear or hybrid, or

3. the properties of the Markov process describing the learning automaton e. g. , ergodic or non-ergodic.

If $p(n+1)$ is a linear function of $p(n)$, the reinforcement scheme is linear, otherwise it is nonlinear. Sometimes, two or more schemes are combined to form a hybrid scheme.

The aim of such a scheme is to realise the advantages (in a practical sense) of the constituent schemes, for example, speed of convergence or variance. The particular constituent scheme to be used at any moment could then be determined by the value of $p(n)$. Similar remarks hold for the transition probabilities given by $f_{ij}^{\beta}(n+1)$.

The basic idea behind a reinforcement scheme is rather simple. If the automaton selects an action α_i at instant n and a favourable input e. g. ($\beta(n) = 0$) results, the action probability $p_i(n)$ is increased and all the other components of $p(n)$ are decreased. For an unfavourable input ($\beta(n) = 1$), $p_i(n)$ is deceased and all the other components are increased. These changes in $p_i(n)$ are known as *reward* and *penalty* respectively. Occasionally, the action probabilities may be retained at their previous values and in such a case this is known as *inaction.*

The same idea can be used to update the transition probabilities. If $\phi(n) = \phi_i, \phi(n+1) = \phi_j$ and $\beta(n) = \beta$, $f_{ij}^{\beta}(n)$ is increased when $\beta = 0$ and decreased when $\beta = 1$. To preserve the stochastic nature of the transition matrix, the other elements of the i^{th} row must be changed in the opposite fashion. The rest of the transition probabilities are maintained at the original values. The precise

manner in which the $p(n)$ is changed depending on the action α_i performed at stage n and the response $\beta(n)$ of the environment, completely defines the reinforcement scheme.

2.5 The General Reinforcement Scheme

2.5.1 Action Probabilities

A general scheme for updating action probabilities can be represented as follows. If

$$\begin{aligned} \alpha(n) &= \alpha_i \qquad (i = 1, 2, ..., r) \\ p_j(n+1) &= p_j(n) - g_j[p(n)] \quad \text{when} \quad \beta(n) = 0 \\ p_j(n+1) &= p_j(n) + h_j[p(n)] \quad \text{when} \quad \beta(n) = 1 \end{aligned} \tag{2.5.1}$$

for all $j \neq i$. For preservation of the probability measure we require $\sum_{j=1}^{r} p_j(n) = 1$ so that

$$\begin{aligned} p_i(n+1) &= p_i(n) + \sum_{j=1, j\neq i}^{r} g_j(p(n)) \quad \text{when} \quad \beta(n) = 0 \\ p_i(n+1) &= p_i(n) - \sum_{j=1, j\neq i}^{r} h_j(p(n)) \quad \text{when} \quad \beta(n) = 1 \end{aligned} \tag{2.5.2}$$

For this to hold true the following assumptions are made regarding the functions g_j and h_j with $j = 1, 2, ..., r$.

Assumption 1 g_j and h_j are continuous functions

Assumption 2 g_j and h_j are nonnegative functions

Assumption 3 $0 < g_j(p) < p_j$ and $0 < \sum_{j=1, j\neq i}^{r} [p_j + h_j(p)] < 1$

for all $i = 1, 2, ..., r$ and all p whose elements are all in the open interval $(0, 1)$.

for all $i = 1, 2, ..., r$ and all p whose elements are all in the open interval $(0, 1)$.

The updating scheme is given at every instant separately for that action which is attempted at stage n (i. e. , action α_i) in equation (2.5.2), and separately for all those actions that are not attempted (i. e. , actions $\alpha_j, j \neq i$) in equation (2.5.1). While the action that is performed is either rewarded or penalised on the basis of the environment's response, it is not clear how the probabilities of the other actions are to be changed. Hence, in the form stated, the question of determining the updating scheme becomes one of determining the functions g_i and h_i. It is further implied in equation (2.5.1) that as long as $j \neq i$, the functions g_i and h_i are independent of the particular α_i chosen.

The continuity assumption on g_i and h_i is one of mathematical convenience. The fact that both g_i and h_i are nonnegative maintains the reward and penalty nature of the updating. Assumption 3 ensures that all the components of $p(n+1)$ remain in $(0,1)$ when those of $p(n)$ are in the same open interval. Strict inequality is imposed in equation (2.5.3) so that $p(n+1) \neq p(n)$ when all the components of $p(n)$ are in the open interval $(0,1)$.

2.5.2 Transition Probabilities

A general reinforcement scheme, in which the transition probabilities rather than the action probabilities are updated, has the following form. If

$$\begin{aligned} \phi(n) &= \phi_i, \phi(n+1) = \phi_j \\ f_{ik}^0(n+1) &= f_{ik}^0(n) - g_{ik}[F^0(n)] \quad \text{when } \beta(n) = 0 \\ f_{ik}^1(n+1) &= f_{ik}^1(n) + h_{ik}[F^1(n)] \quad \text{when } \beta(n) = 1 \end{aligned} \tag{2.5.3}$$

for all $k = 1, 2, ..., s \quad k \neq j$. Further

$$\begin{aligned} f_{ij}^0(n+1) &= f_{ij}^0(n) + \sum_{k=1, k\neq j}^{s} g_{ik}[F^0(n)] \quad \text{when } \beta(n) = 0 \\ f_{ij}^1(n+1) &= f_{ij}^1(n) - \sum_{k=1, k\neq j}^{s} h_{ik}[F^1(n)] \quad \text{when } \beta(n) = 1 \end{aligned} \tag{2.5.4}$$

and for all $u \neq i$ and/or $\beta(n) \neq \beta$

$$f_{uv}^{\beta}(n+1) = f_{uv}^{\beta}(n) \tag{2.5.5}$$

In this algorithm, $g_{ik}(.)$ and $h_{ik}(.)$ are nonnegative continuous functions mapping $s \times s$ stochastic matrices into $\mathcal{R}[0,1]$. It is also clear that the algorithm updates only the i^{th} row elements of the state transition matrix associated with $\beta(n)$ and that the remaining transition probabilities are maintained at their previous values. As in the case of action probabilities, the functions $g_{ik}(.)$ and $h_{ik}(.)$ satisfy subsidiary conditions to assure that all the transition probabilities remain in the interval $(0,1)$.

2.6 Linear Reward-Penalty (L_{R-P}) Scheme

The linear reward-penalty scheme is perhaps the earliest scheme considered in mathematical psychology (Bush and Mosteller, 1958 [17]). The properties of this scheme have been studied in detail by a number of research workers in this field (Fu and McLaren, 1965; Fu and McMurtry, 1966; Chandrasekaran and Shen, 1968; Viswanathan and Narendra, 1972; Norman, 1972; Lakshmivarahan and Thathachar, 1973 [17]).

Consider a learning automaton with two actions (reward and penalty in this case) and let

$$\begin{aligned} g_j(p(n)) &= ap_j(n) \\ h_j(p(n)) &= b(1-p_j(n)) \end{aligned} \tag{2.6.1}$$

where a and b are reward and penalty parameters with $0 < a < 1, \quad 0 \leq< b < 1$. Substituting equation (2.6.1) in equations (2.5.1) and (2.5.2) the updating algorithm can be written as

$$p_1(n+1) = p_1 + a(1-p_1(n)) \quad \alpha(n) = \alpha_1, \beta(n) = 0$$

$$
\begin{aligned}
p_1(n+1) &= (1-b)p_1(n) && \alpha(n) = \alpha_1, \beta(n) = 1 \\
p_1(n+1) &= (1-a)p_1(n) && \alpha(n) = \alpha_2, \beta(n) = 0 \\
p_1(n+1) &= p_1 + b(1-p_1(n)) && \alpha(n) = \alpha_2, \beta(n) = 1
\end{aligned} \tag{2.6.2}
$$

equation (2.6.2) is referred to as the general L_{R-P} updating algorithm. From this equation it follows that if action α_i is attempted at stage n, the probability $p_j(n)(j \neq i)$ is decreased at stage $n+1$ by an amount proportional to its value at stage n for a favourable response and increased by an amount proportional to $[1 - p_j(n)]$ for an unfavourable response.

The specific case when $a = b$ is called the linear reward-penalty scheme (L_{R-P}) and results in symmetric equations. From equation (2.6.2) it is seen that the effect on the probability action α_1 is the same whether α_1 is performed and results in a favourable (unfavourable) response or α_2 is performed and results in unfavourable (favourable) response.

Chapter 3

Reinforcement Learning

3.1 Introduction

Reinforcement learning has been defined by Haykin as [4]

> the on-line learning of an input-output mapping through a process of trial and error designed to maximise a scalar performance index called a reinforcement signal.

Kaelbling, Littman and Moore [21], produced a survey on reinforcement learning. The survey describes many more specific reinforcement learning methods and deals with many of the problems encountered when applying reinforcement learning. It gives a scope of how reinforcement learning methods have developed by necessity and interest, such as the *dynamic programming*, *learning automata*, *value iteration*, *policy iteration*, *W-learning*, *Q-learning*, and *Temporal difference* (λ).

In this research the main interest is in the learning automata and incremental dynamic programming (or dynamic programming-based reinforcement learning). Bradtke [13] uses the term incremental dynamic programming (IDP) to distinguish them from traditional dynamic programming (DP) algorithms. IDP algorithms attempt to find a globally optimal solution by incrementally

improving the local reinforcement payoff as experience is gained through interaction with the environment. The main advantage of using IDP algorithms over traditional DP algorithms is that IDP methods are computationally cheaper [3]. One of the most familiar IDP algorithms is Sutton's Temporal difference (TD) algorithm [13], [24], [54], [3], [22], [55], [37]. Samuel's draughts player (1959) [3] is one the earliest applications to use the TD learning method.

It is a striking feature of the reinforcement learning solution that it can achieve the effects of planning and lookahead without using a model of the opponent (or environment) and without conducting an explicit search over possible sequences of future states and actions [16].

The scope of reinforcement learning encompasses many schemes and learning algorithms [21]. However the fundamental property of all these methods can be explained by equation (3.1.1) [16] [4]. Where $R(t)$ is the expected sum of the immediate reinforcements $r(t)$ discounted by $gamma^t$. The factor $gamma^t$ is used to keep the expected value $R(t)$ a finite value when a large or periodic sequence of actions is used during learning. The fundamental principle being to maximise $R(t)$ when following a reward driven goal or minimising $R(t)$ when following a penalty driven goal or learning.

$$R(t) = E\sum_{t=0}^{\infty} \gamma^t r(t) \tag{3.1.1}$$

Mainly due to its simple concept as a cognitive process by using trial and error learning to acquire experience, reinforcement learning has attracted interest from first psychology, then science, mathematics and now engineering for its application in the intelligent control of complex systems.

Some fundamental aspects and other important points regarding reinforcement learning will be highlighted in order to better explain the roles and processes involved.

3.2 Variation, Selection and Retention

A few of the most important evolutionary processes for learning are variation, selection and retention. When the performance of a system fails to meet targeted aspiration levels the problem driven search routines are triggered generating variation. These variations lead to further selection as the learning agent seeks to gain a better understanding of the environment through exploration.

Selection among variations in an adaptive learning perspective occurs when the results of actions are compared to some preset aspiration levels. In keeping within the scope of searching through the problem space a learning agent should keep those variations that helped them reach their targets and try other variations to replace those that failed. In short, successful actions tend to be repeated and unsuccessful actions should provoke further search. The importance of selection in reinforcement learning is emphasised due to the way in which a policy is built or learned as the agent evolves. Although the term evolve used in this case is not biological evolution, there are a lot of similarities, as tasks change and goals move, new selections are required while old one become extinct until a stable equilibrium is found when the desired goal is reached.

Retention mechanisms are critical for learning, for without a way to store and retrieve routines or knowledge, learning agents gain nothing from experience. From an adaptive learning perspective, the results of the search are stored in routines and performance programs that can be reused when needed. Learning is then embodied in a set of interlocking role behaviours between learning agent and environment, supported by internal state transitions, action descriptions and external interactions.

3.2.1 Actions

Some actions in the learning agent are simple random variations, a method often used to encourage exploration. Other actions are the result of repetition, when it is best to repeat the same past action since no better alternative has been learnt yet, some would treat repetition as just another action only if the underlying rationale for repetition has also been learnt, arguing that only intentional learning should count as real learning.

Most actions by learning agents are intendedly rational within the scope of the task to be solved, but often they are denied complete rationality because of limitations. In complex systems a learning agent is sometimes precluded from making optimal choices by cognitive deficiencies and peculiarities, limits on information availability, and constraints on information processing. Information search costs (most often the cost is time) will lead most actions to choose satisfactory, rather than optimal alternatives. Learning agents also act with self interest at the expense of other learning agents, although the scheme and multi-agent learning architecture used in this research there is no conflict of interest when agents act selfishly.

3.2.2 Optimal Control

For a deterministic system containing no disturbances, given any closed-loop policy and initial state, there exists an open-loop policy that produces the exact system behaviour. However for a stochastic system, or a system with unmodelled disturbances this cannot be true because the outcome of random or unmodelled event cannot be anticipated in designing an open-loop policy. Game-playing systems such as Tesauro's backgammon player [24], [54] and Samuel's draughts player (1959) [3], always use closed-loop control for this reason, with the opponent acting as a kind of disturbance. For the same reason closed-loop control is usually better than open-loop control for single

agent problems involving uncertainty.

The most familiar control objective is to control a system so that its output matches some desired setpoint or tracks the setpoint trajectory as close as possible when disturbances are present. In an optimal control problem, the objective is to find the best trajectory within some function that controls the systems behaviour. A typical optimal control problem is finding the minimum-cost trajectory from an initial start state to some defining goal state. Thus many optimal control problems are also related to problems involving heuristic search algorithms (reinforcement learning).

3.2.3 Hidden States

The solution to optimal control problems requires the access of system state information. Sometimes, the system state is not immediately apparent from the information available. The problem of hidden state information has been termed *perceptual aliasing* by researchers concerned with building autonomous agents that learn through interaction with their environment [13], [56]. Most problem environments for autonomous learning agents can be modelled as a *Markov decision problem.*

One solution to overcome the problem of hidden states has been developed by McCallum [57]. McCallum used a reinforcement learning approach, called *instance-based state identification* and applied it to robot navigation and task problems. The approach applies instance-based (or memory-based) learning to history sequences, this history information uncovers the hidden state(s) using state identification techniques.

3.2.4 Markov Decision Problems

Many problems of practical importance have been formulated as Markov decision problems, and extensive treatment of the theory and application of

this framework can be found in many books [10]. Any problem that can be cast as a Markov decision problem is theoretically solvable using reinforcement learning. A Markov decision problem contains a set of states ϕ which contain a subset of start states S and a set of actions α. A reinforcement function R and an action model A are required such that $R(\phi, \alpha)$ is the expected immediate reinforcement for taking action α in state ϕ and $A(\phi_j \mid \phi_i, \alpha)$ gives the probability that executing action α in state ϕ_i will lead to state ϕ_j with $i \neq j$. In order to be a Markov decision problem the choice of action must depend only on the current state observation ϕ, if knowledge of prior actions or states affects the current choice of action then the decision problem is not Markov.

One of the simplest and common examples to cast as a Markov decision problem is that of a two-dimensional robot navigator. This is normally illustrated as a grid-world, with each grid element being a state. Some states are freely accessible while others contain obstacles to limit the directions a robot can take. The problem posed is that of finding the shortest path from an initial start state to some goal state. The objective of the robot is to find the optimal policy, this maximises a return or payoff, usually a scalar reinforcement signal. The policy maps states to actions and the magnitude of the payoff received in response to an action is inversely related to the cost of the path travelled (so that maximising the total payoff will minimise the total path cost).

3.3 Properties of Reinforcement Learning

There are only two fundamental elements to a reinforcement learning system, one is the learning agent itself, the other is the environment. The emphasis on learning by trial and error with the environment acting as a teacher/guide has been stated. Without the environment there wouldn't be any of the interaction needed for the learning agent to learn. But there will always be an environment since the point of the learning agent is to solve some task

A *r = 0*	**B** *r = -1*	C *r = -1*
D *r = -1*	**E** *r = -1*	**F** *r = -1*
G *r = -1*	**H** *r = -1*	**I** *r = -1*

Figure 3.1: A simple three by three grid state space representation (the environment) showing the reinforcement reward feedback signals and one goal state 'A'.

within the environment. However, no explicit knowledge of the environment is required, making reinforcement learning very flexible as well as powerful.

The learning agent on the other hand is more important, within there are four identifiable sub-elements which Sutton has identified that are common to all reinforcement learning [16]. These are

- A reward function
- A value function
- A policy
- A model of the environment (optional)

What a reinforcement learning system needs to do and how they can be achieved will be illustrated in order to understand the strategies for solving them more efficiently if possible.

A simple three by three grid state space structure (see Figure 3.1) is used to help illustrate the points discussed. Each state is labeled from 'A' to 'I' with 'A' indicated as being some desired goal state for some learning agent to reach,

the nature of a goal in reinforcement learning can be a simple decision/action such as choosing a good control action or a situation such as don't get lost (e.g. the exit in maze solving problems) or even high level actions such as a change in state of mind, an example of which is from avoiding predators to searching for food as used by the ant problem [26]. In the three by three grid environment (or world) all transitions from state to states are limited to moving perpendicular to the current state or only moves of north, east south and west from the current state are permitted unless an outer edge prevents such a transition.

3.3.1 Reward and Penalty

The first thing all reinforcement learning needs to know is what is currently good and what is currently bad. A *reward function* is used to guide the reinforcement learning agent to reach a desired goal or achieve a certain objective. It is a scalar value mapped to each state to indicate the desirability of that state when a given action is performed from that state. All the reinforcement learning agent has to do is to maximise the accumulated rewards (or minimise the penalty) in order to reach its objective. The reward function defines what are good and bad events and give an immediate measure of this, they are used to alter the agents policy. For example the terminal voltage of a turbo-generator needs to be within a certain range, $1.2kVto1.4kV$, and at a time t the terminal voltage is $1.2kV$, then the condition is a good one, therefore return a reinforcement feedback signal $r(t) = 1$ to indicate this. Else if the action selected by a policy is followed by a low reward $r(t) = 0$, the policy is then changed to select some other action when in that situation is encountered in the future.

The feedback reinforcement signal $r(t)$ is the first important element that all reinforcement learning agents use in order to come to a useful decision for selecting future actions. The actual nature of the reinforcement feedback signal

varies from one problem to another. The most common is to use a binary signal of the form 0 for good situations and -1 to indicate bad/undesirable situations since most reinforcement learning has been defined as a problem of maximising the expected sum of discounted payoffs $r(t)$ by satisfying equation (3.1.1), where the parameter γ is a discount factor to return a finite sum if the sequence of events is very large or infinite. This enables the design of a control system to be simplified by allowing it to discover the control policy for itself but by necessity the task must be fully described by the payoff function.

In its simplest form the agent is basically trying to maximise the rewards which is what our simple illustration will hope to achieve. The choice of selecting a suitable reward function $r(t)$ will mainly depend upon how to best represent the states of an environment. For example if the task is best served by minimising a risk then a reward function of the form $r(t) = 1$ for an undesirable state and $r(t) = 0$ for a desirable state can be used. In this case the risk is minimised if the shortest path from any arbitrary start state to any goal state is taken. If the case had been to maximise a profit then $r(t) = 0$ for a good state and $r(t) = -1$ for a bad state can be used. Although there will be a limited way of choosing a reward function $r(t)$ for a particular task, in general the best reward functions take into account the fact that some environments are very large or can have continuous periods (loops) and include a discount factor to compensate.

For the purposes of the illustration and referring to Figure 3.1 a simple reward function is used, all states return a reward of $r(t) = -1$ unless it is the goal state (State 'A') then $r(t) = 0$.

3.3.2 The Value of a State

A *value function* in reinforcement learning is used to provide an indication of the value of a state. The value of a state is its expected accumulated reward

for starting from that state until reaching some desired goal state. The value of a state is used by the learning agent to guide its actions in order to reach the goal state in the most efficient way. Where the reward is an immediate desirability of a state, the value indicates how valuable the state is in the long term by taking into account the states that follow and the rewards available in those states. So a state can have a low reward but a high value because it is often followed by states with high rewards, the reverse can also be true. The value function uses the rewards generated by the reward function in order to assign a value to each state. Without rewards, values cannot be estimated, while values are estimated to achieve more reward by maximising the expected accumulated reward. It is these values which the agent uses to choose actions and hence change or follow a policy. Rewards are determined by actions taken in the environment, but values must be estimated and re-estimated from the sequence of observations the agent makes in its lifetime.

In the simple example using Figure 3.1 the first stage any reinforcement learning must do is to explore its environment, this is usually achieved by some sort of searching such as a random search in this case actions from an arbitrary start state are randomly chosen from the possible four actions available, these are to move up, down, left or right. If the start state is 'E' then suppose the random search followed this trajectory, E, H, G, D, H, B, A. Each non goal state transition returns a -1 reward signal which is added to the current sum until the goal state is reached, so using this example a value of -6 is assigned to state 'E'. After further exploration trials starting from state 'E' the trajectories that do not lead to further improvements in the state values are E, B, A, or E, D, A. Both of these provide the value of -2 to state 'E' which is the optimal value for this state. The other states are also optimised in this manner, the optimal values for each state are shown in Figure 3.2. The value function is arguably the most important component in a reinforcement learning system.

A *v = 0*	**B** *v = -1*	C *v = -2*
D *v = -1*	**E** *v = -2*	**F** *v = -3*
G *v = -2*	**H** *v = -3*	**I** *v = -4*

Figure 3.2: The optimal state values of a simple three by three grid environment with one goal state.

3.3.3 The Policy

A *policy* determines the behaviour of a reinforcement learning agent and is a mapping from perceived states of an environment to actions available to the learning agent when in those states. As shown by the learning automaton a policy can either be deterministic or stochastic. A reinforcement learning agent can explore an environment by continually changing the policy. Eventually a policy, or policies will be found that will maximise (or minimise depending on the task) the total reward received when using that policy. When this happens the policy is said to be a optimal. This is what psychologists call *stimulus-response* rules [16].

In most cases of reinforcement learning the optimal policy is a simple task to achieve if the values of each state are optimised first, the optimal policy simply becomes the policy that uses the state values that maximise the reward or minimise the risks. This type of policy has been referred to as a *greedy policy* by Watkins using his Q-learning method [38], [46], [24], [54], [44]. In the illustrated example (see Figure 3.1 and Figure 3.2), from the state 'H' the greedy policy would select actions to move through states E, B, A, or G, D, A,

or E, D, A in order to maximise the accumulated rewards.

3.3.4 System Model

A *model* of the environment is an optional part of reinforcement learning. Models though are sometimes useful for planning by the reinforcement learning agent. Most reinforcement learning systems can be applied without any knowledge of the environment but some build up a model as learning progresses and then use this for planning. Traditional dynamic programming methods are limited to solving problems with manageable state-space dimensions. Incremental dynamic programming methods such as TD learning overcome this limitation by updating knowledge after every action (state to state transition) rather than waiting for a whole sequence of many actions to finish.

Early reinforcement learning techniques for adaptive control, such as the learning automaton [21], [17], [28], and later incremental dynamic programming such as TD learning [13], [21], [45], [58], did not require a model of the plant/environment to be developed. Therefore when a model of the plant is unknown, two alternatives for the development of an adaptive control system are available [58].

- Estimate a model and from this develop a control rule
- Develop a control rule without building a model

If a model-based approach is used, then it will be necessary to build a model first. There are two primary arguments for taking a model-based approach. First, building a system model and then using that model to solve the optimal control problem is often much easier than trying to solve the optimal control problem directly. This is especially true for most linear systems where a model can be easily built. However, solving even a very simple problem of this type using a direct or model-free approach can be relatively difficult [13]. The second

reason for using a model-based approach is that, once the model is formed it may be used as a basis for solving many related control problems.

Two examples of model-based reinforcement learning are *Dyna*, developed by Sutton [3], [59], [52] and *prioritised sweeping*, developed by Moore and Atkeson [58]. A *Dyna* architecture integrates trial and error (reinforcement) learning with execution time planning into a single process, operating alternately on the real system and a model of that system. Incremental learning methods such as TD-learning have fast real time performance because they use the most optimal policy which is not necessary the actual final optimal policy, while traditional control methods are slower but more accurate by waiting until they have obtained the optimal policy after full use of all the observations. Prioritised sweeping combines both advantages into one system by using all previous experiences to prioritise important dynamic programming sweeps and also guide the exploration of state-space.

For a model-free control system design the primary argument is that it may be less expensive to find the optimal (or an acceptable) controller by direct interaction. It is relatively easy to model a linear dynamic system, however to obtain an accurate model of a stochastic complex system such as the financial market can be difficult or impossible. The usual approach is to idealise such complex systems using approximations, the result is a model that is only accurate within certain constraints. For some applications, especially safety critical ones, this is usually not acceptable. Even if an accurate model of the system is known, a model-free control system may still be required, since it will often be the case that the derivation of an optimal controller obtained directly from that model, is analytically and computationally intractable. Consider the game of backgammon [42], [13], [24], [17]. The state transition function for backgammon is specified by the rules of the game. However, backgammon is estimated to have at least 10^{20} states. Trying to find the optimal policy for the game of backgammon using a traditional model-based approach is an

intractable problem.

Other motivations for using model-free methods such as the learning automaton are: the interesting fact that these methods nevertheless do learn and the possibility that they simulate some types of biological learning [22]. Since the main interest is in model-free reinforcement learning methods, the research effort will be concentrated here.

3.4 Limitations to Learning by Experience

Apart from random action selection when faced with any new environment are there any other techniques that can be used? Since in the initial stages any action will acquire new knowledge using a random selection does not pose a problem. The problem however is knowing when to use the acquired knowledge instead of searching for more maybe less useful knowledge (e.g. when the current calculated state value is worse than the previous state value). This is the *exploration/expliotation* problem encountered by all self learning agents. In most reinforcement learning techniques there is a switching between exploring (updating state values) and exploiting (using the current state values to drive a policy). It is formalising a strategy in order to successfully change from exploration to exploitation which is the challenge.

In the illustrated example there are only a small and finite number of states, the optimal state values can be quickly found since the possible ways to randomly choose actions from any state is also finite and manageable. Given enough time any reinforcement learning agent can eventually through exploration find all the optimal state values. However that is the question, when will it find all the state values (preferably the optimal ones) and will it even know it has explored all the states. Presumably learning will stop when all the state values cannot be improved anymore, in which case it has explored all states. For stochastic systems and systems that have a very large state space learning

can take a long time which may be undesirable or unacceptable. One way to alleviate this problem is to use a team of reinforcement learning agents each exploring a subset of the state space so collectively they will reduce the exploration time and acquire knowledge quickly. However the co-ordination between reinforcement learning agents will need to be studied to see how they use their knowledge to gain wisdom if they are to collectively make good decisions. Two types of co-ordination are possible they can either become competitive and come to decisions based on which reinforcement learning agent has the most influence and therefore takes charge, or they can become co-operative in which all knowledge is shared and used to come to an agreeable action between agents. So far the most prominent examples of these two opposing view points are by Humphrys [26] with W-learning using selfish motivations in learning agents and Wu [28] using learning automaton in a team architecture with shared reward functions.

3.4.1 Experience in Reinforcement Learning

There are essentially two ways of using experience in reinforcement learning. One is called *model* or *indirect learning*, and the other *direct reinforcement learning*. The possible relationships between experience, model, value and policy can be shown in Figure 3.3 [16]. Indirect methods make fuller use of a limited amount of experience and can converge to a good policy with fewer interactions. On the other hand direct methods are much simpler and are not affected by past model experiences (or bias) if the environment changes, thereby requiring to re-learn another model of the environment.

However both methods use their experience to change a policy which updates the values which are used to further improve the policy. The universal task for all reinforcement learning agents is to maximise the accumulated rewards using the state values as a guide. This interactive relation with the state

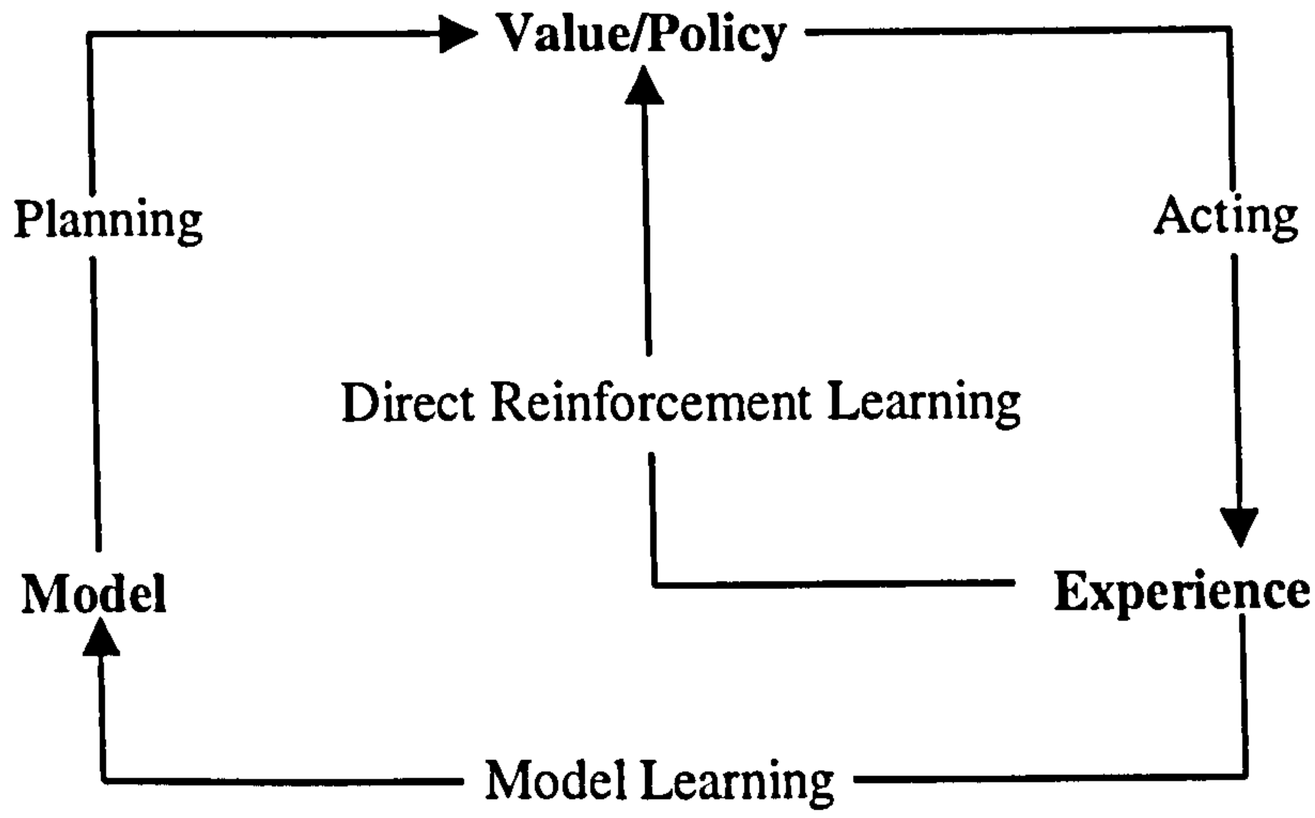

Figure 3.3: Relationships among learning, planning and acting [16].

values and the policy repeats until no more improvement in the accumulated rewards can be made. When this happens the state values can no longer be improved and the policy is said to be an *optimal policy*, i.e. the most efficient way of reaching the desired goal.

The choice of method will depend upon the type of task being solved (stochastic, linear, non-linear, deterministic, noisy, etc...) and what is required of the reinforcement learner to achieve. The learning automaton and TD reinforcement learning methods are both examples of *direct reinforcement learning* that explicitly learn from trial and error. There have been some developments in making TD methods more model based by Sutton, known as *Dyna* [60] which is an architecture for learning, planning and reacting. Also conventional dynamic programming methods cannot learn without a model, so it is evident that modern reinforcement learning methods span the entire spectrum from low level, trial and error learning to high level deliberative planning.

3.4.2 Pure Delayed Reward and Avoidance

One of the most important aspects of learning is that of assigning credit. Here the agent solves a *credit-assignment problem* in order to find the optimal policy [45], i.e. which action(s) of an action sequence to blame if the total reward is non-optimal. It is not necessarily optimal to always execute the action with the largest immediate reward, because executing actions with smaller immediate rewards may be necessary to make large future rewards possible. This is called the *problem of delayed rewards* or reinforcement learning with delayed rewards. The learning agent has difficulty in assigning credit to good, or blame to bad actions/states because no evaluative feedback is given until the end of a trial or learning sequence. In reinforcement learning the reinforcement feedback signal $r(t)$ gives an immediate (or short term) indication of how good or bad the decision was for a particular state. However this is insufficient in many real problems and the long term values of each state must be estimated or known for efficient prediction and/or control, that is the credit assignment problem.

For example, a reinforcement agent is required to play the game of draughts. The sign of the reinforcement scalar at the terminal state indicates whether the terminal state is a goal state (reward) or a state that should be avoided (penalty). The state space is the position of each players pieces on the game board, the available actions the agent can make are the set of legal moves. The reinforcement function is defined to be zero after every turn except when an action results in a win (+1 reinforcement) or a loss (−1 reinforcement). The goal of the learning agent is to maximise the reinforcement so it will learn that states corresponding to a win are goal states and states resulting in a loss are to be avoided.

3.4.3 Minimum Time to Goal

Reinforcement functions of this type find the shortest path or trajectory to the goal state. An example of this is a stationary car between two steep inclines as described by Sutton [16], the goal of the driver (reinforcement learning agent) is to reach the goal state on top of the hill by driving up the incline. The state of environment is the position and velocity of the car. Three actions are available to the agent in each state, forward acceleration, backward acceleration or no acceleration. The problem is that the car cannot simply drive up the hill due to the steepness of the incline. The driver must learn to use momentum in order to gain enough velocity to successfully climb the hill. In order to maximise the reinforcement scalar, the agent learns to choose actions that minimises the time taken to reach the goal state and hence learns the optimal strategy for driving the car up the hill.

3.4.4 Exploration Versus Exploitation

The fundamental question to all reinforcement learning is to find the goal state in an optimal or efficient manner. Since initially the agent is faced with an unknown environment, the agent must perform a search or exploration of the state space. In order to explore the agent must choose an action that is not considered the best for the purpose of gaining new knowledge of unseen or seldom seen states. The need for the learning agent to explore is fundamental in identifying the optimal as well as sub-optimal states and sufficient exploration of the state space must be conducted. Sutton [16] shows many examples when reinforcement learning is better overall after some time is used to explore the environment seeking better actions/decisisions rather than purely following a greedy policy.

Take the example of a robot in an unknown environment, initially some time has been spent exploring and acquiring knowledge of its environment.

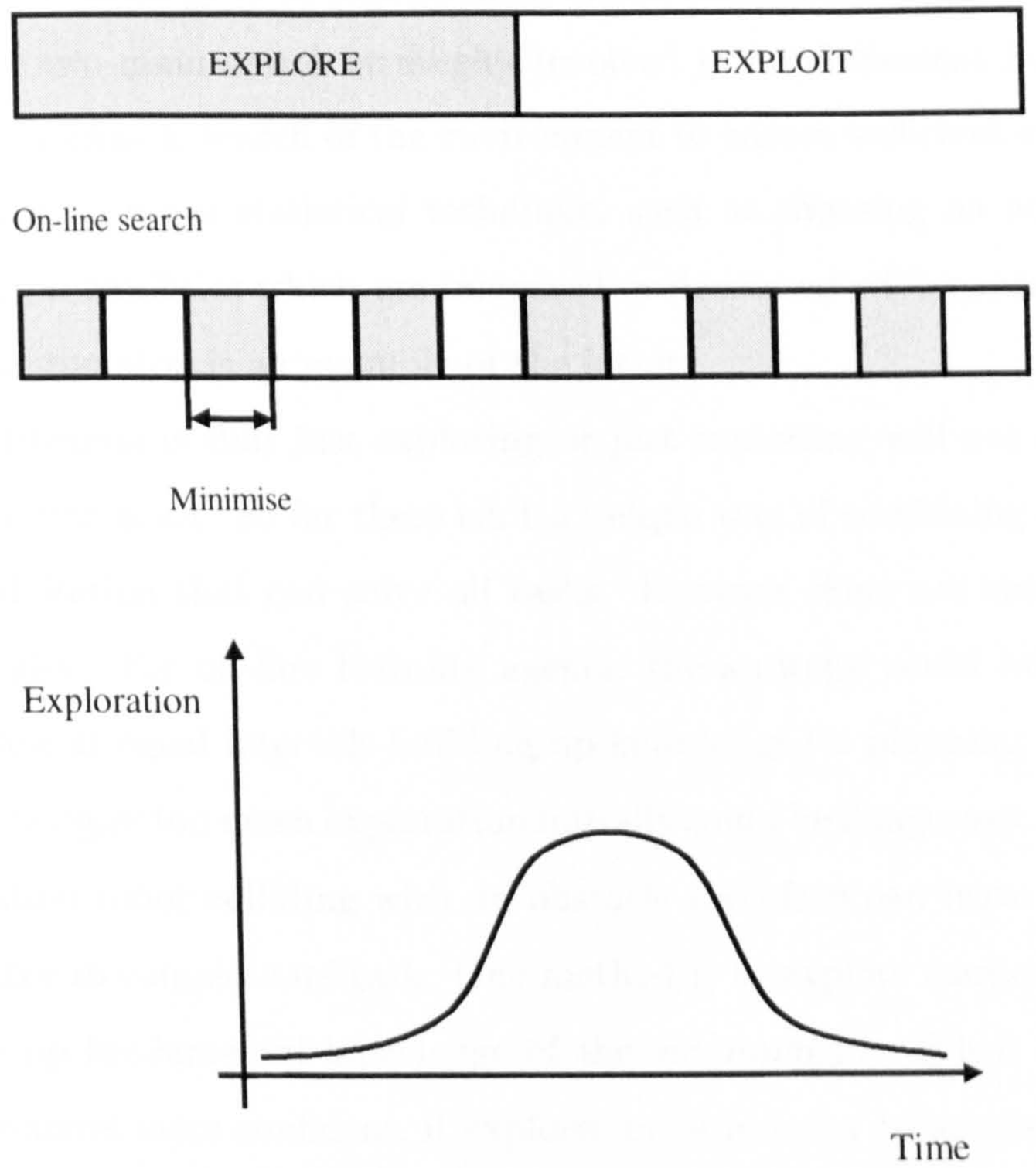

Figure 3.4: On-line and off-line exploration/exploitation strategies.

Experience gained during exploration must also be considered during action selection to minimise any negative reinforcements (penalties) such as collisions with obstacles (which can be fatal). However, the robot does not know which actions will lead to a collision until all of the state space has been explored. Sometimes it is possible for a policy to be sufficiently good that exploring the complete state space is not required, hence there is this trade-off between exploration and exploitation. This *exploration-exploitation* problem is fundamental to all reinforcement learning agents, the ideal reinforcement learning agent will

use exploration to maximise the knowledge gained during learning while also minimising the costs of exploration and learning time. In practical systems, the standard strategy is to exploit most of the time and explore from time to time. The two main search strategies involved in reinforcement learning are, one use a stochastic search of the environment to ensure sufficient exploration, the other is to use a statistical technique, such as choosing an action based on action probabilities which are increased or decreased with experience. The learning automaton is an example of the latter.

The dilemma is that just exploring or just exploiting will not achieve the task the agent is set. So far there isn't a unique way of combining exploration with exploitation that can solve all tasks. However there are some common sense guides. For off-line learning agents, the strategy could be to explore and exploit at equal intervals building up knowledge for planning. For on-line search strategies too much exploration initially could be dangerous, for example a navigation robot colliding with an obstacle too often can leave it damaged and unable to complete its task. One method is to explore cautiously at first, building up fundamental knowledge of the environment. When the learning agent becomes more confident, it explores more in order to acquire knowledge at a faster rate. After sufficient exploration the agent exploits more, using the knowledge gained when exploring to ensure it converges to a policy (hopefully the optimal policy). However it may be wise to still explore a little after convergence to a policy, because the current best policy may not be optimal and exploration may find a better policy. This can be illustrated in Figure 3.4.

Chapter 4

Temporal Difference Learning

4.1 Introduction

Temporal difference learning emerged from the study of ADALINES (adaptive linear elements) by Sutton [34], [3] and motivated by adaptive systems such as artificial neural networks. Temporal difference (TD) learning was later formalised by Sutton [3] when he realised that they were closely related to conventional dynamic programming. Although TD reinforcement learning is based on conventional dynamic programming methods and the Widrow-Hoff rule for neural network learning [3] [34], they have many advantages over either method. TD has the advantage of incremental learning and the updating of weights is performed each step incrementally rather than all at once after a complete sequence of events. So unlike conventional dynamic programming it does not need to finish a complete sequence of actions (which can be very long) before updating its knowledge base, and can learn new knowledge after successive predictions. The other advantage is that the overall computation and memory requirements for TD learning is relatively small compared to traditional dynamic programming making TD learning more suitable for practical applications.

Like the development of the learning automaton temporal difference learning is a milestone in furthering the practical evolution of reinforcement learning. Temporal difference learning has brought a lot of disciplines closer together. The neuron like structure of temporal difference was a progression from previous work on neural networks [3] which are further based on biological models of the brain [4], [12]. The decay of the eligibility traces in temporal difference learning has a basis in animal behaviour and models the memory forgetting factor in animals as a function of time [22]. Early reinforcement learning using the learning automaton started from the psychologists point of view in which temporal difference learning is part of but no longer confined to [3], [54], [17]. The incremental updating nature of temporal difference learning was also an improvement on another well established mathematical computation method known as dynamic programming, which has origins in mathematics and computer science [7], [8], [13], [9], [10], [59], [52]. Finally with the increased interest in temporal difference learning many engineers are seeing the advantages of this reinforcement learning method and applying them to control large-scale systems.

4.2 Temporal Difference Learning

The temporal difference learning is a relatively recent reinforcement learning algorithm for optimising control and machine learning in general. As mentioned in the introduction Temporal Difference (TD) learning has already attracted a diverse range of applications, from playing complex games such as Samuel's Draughts player (1959) [3] and Tesauro's Backgammon player [24], to improving elevator performance [43] in large office buildings. The reason for using reinforcement learning for many of the examples is due to the stochastic nature of those systems, making it a difficult task using conventional non reinforcement learning control methods.

Learning to predict is one of the most important tasks required in learning. The conventional approach to prediction learning is to adjust the parameters in the predictor based on the error between actual and predicted values. TD learning updates the predictor parameters by using the error between successive predictions. Accordingly, learning occurs in TD methods whenever there is a change in prediction over time. The training examples are taken from the temporal sequence of input vectors and hence TD methods are unsupervised and learn on-line, sometimes these methods are referred to as *adaptive prediction* methods [38].

Sutton formalised a complete class of TD learning methods in a paper published in 1988 comparison with supervised learning [3]. The general TD method was introduced called TD(λ), where λ is a *weighting factor* with a value $0 \leq \lambda \leq 1$. Instead of updating a state value (or approximate value) based on the values of the immediate successor states, TD(λ) bases the update on an exponential weighting of values of future weights. With the two extreme cases TD(0) being similar to Q-learning and TD(1) being similar to supervised learning, since TD(1) updates the value (or approximate value) of state n solely on the value of the terminal state [42], [3]. The general temporal difference (λ) algorithm for updating weights $w_{(t)}$ in a connectionist system such as a neural net has the form given in equation (4.2.1).

$$\Delta \mathbf{w}(t) = \alpha(p(t) - p(t-1)) \sum_{k=1}^{t} \lambda^{t-k} \nabla_{\mathbf{w}} p(k) \qquad (4.2.1)$$

The features are the temporal difference error $p(t) - p(t-1)$ which drives the learning, with α being a learning rate and $\nabla_{\mathbf{w}} p(k)$ provides gradient information to indicate if the improvements need to be adjusted positively or negatively. The factor λ is a trace decay, where $\lambda = 0$ models short term memory and only the past single step is remembered, when $\lambda = 1$ long term memory is modelled and all the previous steps are remembered. The most useful and practical representation of memory fading is one that has a value

$(0 < \lambda < 1)$.

In all reinforcement learning the learning agent tries to maximise the expected sum of discounted payoffs $r(t)$ received by satisfying equation (3.1.1). Temporal difference learning is a method for predicting the expected sum of discounted payoffs. The agent learns by minimizing the error difference between the predicted and actual values.

4.2.1 Adaptive Heuristic Critic and TD(λ)

The temporal difference reinforcement learning algorithm was later adapted for use within the *adaptive heuristic critic* (AHC) architecture using neuron like structures by Sutton, Barto and Anderson [34] to solve difficult learning control problems. The AHC also makes it easier to applying TD learning because of its neuron like structure for connectionist architectures. Similar methods to the AHC in which a critic is used to enhance the action evaluation are also known as *actor-critic* methods [16].

The adaptive heuristic critic reinforcement learning system uses TD methods of prediction and learning control to solve stochastic sequential decision tasks. The two important elements of the AHC are the *adaptive critic element* and *associative search element* [34].

A block diagram to this approach is given in Figure 4.1 and consists of two components, the critic (AHC) and reinforcement learning component (RL). The reinforcement learning component is designed or chosen to maximise the heuristic reinforcement value, $\hat{r}$, that is computed by the critic. The critic uses the external reinforcement signal to learn to map states to their expected discounted values given that the policy being executed is the one currently in the RL component [21]. The system is designed to learn under *delayed reinforcement*, this is a temporal sequence of input state vectors (stimuli) that eventually result in the generation of the heuristic reinforcement signal. Instead

of acting to maximise instantaneous reward r, the AHC tries to maximise the heuristic reinforcement value $\hat{r}$ computed by the critic. TD learning and the AHC is one of the first methods to address the credit assignment problem because of its use of learning by delayed reinforcement. This is achieved by the use of eligibility traces for modelling the decay of short term memory in animals.

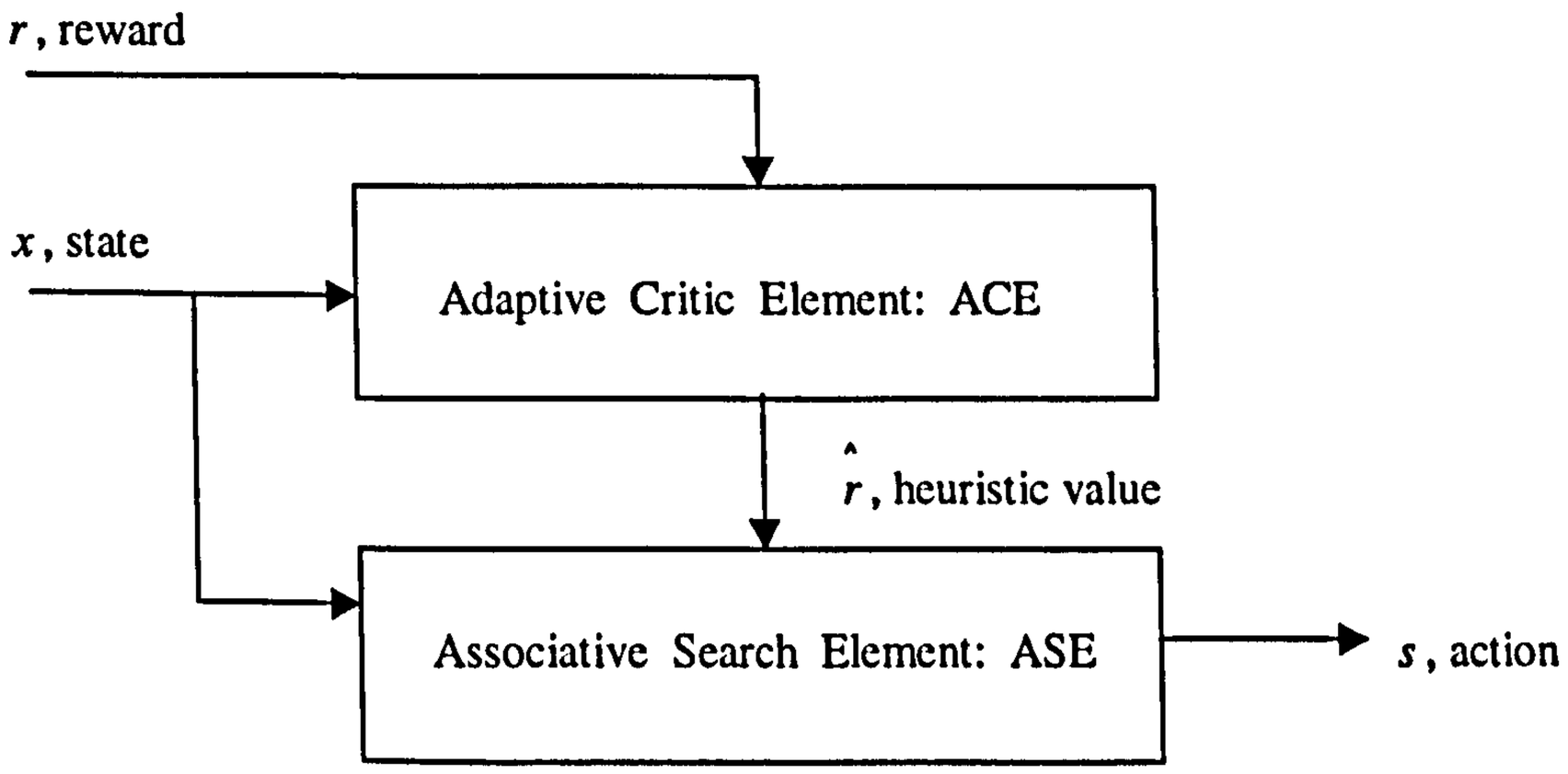

Figure 4.1: Architecture for the Adaptive Heuristic Critic [21].

The adaptive heuristic critic system contains an *adaptive critic element* (ACE) and an *adaptive search element* (ASE). As learning proceeds the ASE constructs associations between inputs and outputs by searching under the influence of a reinforcement feedback. The reinforcement feedback r is zero for example in non fail states and -1 in fail states. The ACE uses r to provide a more informative evaluation function, the heuristic reinforcement, which is used by the ASE to evaluate if an action selected was good or bad.

4.2.2 The ACE

The central idea behind the ACE is an algorithm that predicts the future reinforcement and is a function of the system state vector. Barto et al [34] uses the following equation to determine the prediction $p(t)$, the ACE predicts performance $p(t)$ by using equation (4.2.2),

$$p(t) = \sum_{i=1}^{n} v_i(t)x_i(t) \tag{4.2.2}$$

subsequently the ACE weights $v_i(t)$ are updated using equation (4.2.3),

$$v_i(t+1) = v_i(t) + \beta[r(t) + \gamma p(t) - p(t-1)]\overline{x}_i(t) \tag{4.2.3}$$

where the eligibility trace $\overline{x}_i(t)$ by equation (4.2.4) is used to model the gradual fading of memory.

$$\overline{x}_i(t+1) = \lambda\overline{x}_i(t) + (1-\lambda)x_i(t) \tag{4.2.4}$$

The ACE output, known as the *heuristic reinforcement* $\hat{r}(t)$, is given by equation (4.2.5)

$$\hat{r}(t) = r(t) + \gamma(p(t) - p(t-1)) \tag{4.2.5}$$

where β, γ, λ are constants in the range $0, 1$ and the external reinforcement $r(t) = 0$ except at failure $r(t) = -1$, however the reinforcement feedback signal $r(t)$ range can be selected depending upon the nature of the optimisation task, such as maximising reward, then $r(t) = 1$ indicates a good action and $r(t) = 0$ is a poor action for example. Another example is when minimising an error then $r(t) = 0$ is representative of a good action and $r(t) = 1$ means a bad action selection. The feedback signal $r(t)$ in temporal difference learning is very similar to the reinforcement feedback signal β for the learning automata.

The purpose of the heuristic reinforcement is to provide an indication of the long term value for each state. The ACE uses the immediate reinforcement signal $r(t)$ to generate the heuristic reinforcement value $\hat{r}(t)$. The role of the ACE is to act as the value function in reinforcement learning.

4.2.3 The ASE

The ASE uses the improved heuristic reinforcement signal $\hat{r}(t)$ to generate the control action $s(t)$. The ASE weights $w_i(t)$ are updated using equation (4.2.6),

$$w_i(t+1) = w_i(t) + \alpha \hat{r}(t) e_i(t) \tag{4.2.6}$$

with eligibility traces $e_i(t)$ updated by equation (4.2.7)

$$e_i(t+1) = \delta e_i(t) + (1-\delta) y(t) x_i(t) \tag{4.2.7}$$

where parameter α and δ are constants in the range $0, 1$. Note the similarity between updating the ACE weights $v_i(t)$ and the ASE weights $w_i(t)$ when equation (4.2.5) is substituted for $\hat{r}$ in equation (4.2.6). The ASE control output $s(t)$ is given by equation (4.2.8),

$$s(t) = f(\sum_{i=1}^{n} w_i(t) x_i(t) + \text{noise}(t)) \tag{4.2.8}$$

The control output $s(t)$ is used to determine what new action is required after evaluation by the AHC when $r(t)$ is returned. The function $f(.)$ can be any function which can provide the output of the controller with a useful set of actions. For non-linear systems a sigmoidal or other non-linear function that can be used to provide a continuous output range from $0, 1$, which can then be further discretised to provide a finite output set. The usage of function $f(.)$ can be better illustrated in the case study example of balancing an inverted

pendulum which follows. In that case only two output actions were required push cart left or push cart right, there the choice of using a threshold activation function sufficed.

4.3 Inverted Pendulum control

Sutton, Barto and Anderson used the adaptive heuristic critic architecture with temporal difference learning to successfully balance the inverted pendulum problem to validate their idea. The result being that successive trials for balancing the inverted pendulum would lead to success longer, or in other words the learning agent learned to improve its current performance relative to past performances.

The TD learning used to balance the inverted pendulum problem can be described with the aid of the flow chart, Figure 4.2 shown below.

Figure 4.3 shows how the ACE and ASE elements relate in the adaptive heuristic architecture Barto, Sutton and Anderson used to solve the inverted pendulum control problem [34].

As learning proceeds the ASE constructs associations between input and output by searching under the influence of a reinforcement feedback. The reinforcement feedback r is zero everywhere except for the states in which the pole falls or the cart hits the ends of the track as shown in Figure 4.4, when this happens the agent receives a -1 reinforcement. The ACE uses r to provide a more informative evaluation function (the heuristic reinforcement) than the reinforcement feedback r alone can provide. Referring to Figure 4.3 each situation is represented by a states in the inverted pendulum system. Four input variables, x, cart position on the track, $\dot{x}$, cart velocity, θ, angle of inverted pendulum relative to the vertical (vertical position is equal to zero degrees) and $\dot{\theta}$ is the angular velocity. The decoder converts the four parameters into 162 states for use in the ASE and ACE. Two actions are available to the learning

agent, these are move the cart to the left or move the cart to the right. This is somewhat simplistic in that if the agent learns one action (e.g. push left) in a given situation is wrong then the sole alternative (i.e. push right) must be the correct action in that state. The agent selects a policy based on the heuristic reinforcement that will maximise the total pure reinforcements r in order to select a sequence of actions required to balance the inverted pendulum for as long as possible. The effect of this learning can be seen in Figure 4.5. Again the common elements that make up a reinforcement learning system can be identified in the adaptive heuristic critic architecture. First the policy is determined by the ASE which chooses actions biased by the ACE, the ACE itself is the value function for generating long term values (the heuristic reinforcement) for each state, and the feedback rewards r are used to measure the states immediate utility.

A simulation of the inverted pendulum balancing problem was used to illustrate the learning of the ACE/ASE reinforcement learning system. Each trial begins with the cart pole state $x = 0$, $\dot{x} = 0$, $\theta = 0$, $\dot{\theta} = 0$ and ends with a failure signal $r = -1$ indicating θ has left the interval $[-12°, 12°]$ or x has left the interval $[-2.4m, 2.4m]$. All initial trace variables e_i and weights v_i, w_i were also zero at each trial start. As more trials were made the learning agent would try to improve its performance each time so that after many trials it could keep the inverted pendulum balanced the longest, see Figure 4.3. Each trial length varied and was determined by the length of time the learning agent could keep the inverted pendulum balanced, a total of 100 trials was performed and in most cases as observed in the results Figure 4.5, the current trial performance was an improvement on the previous trial.

In the inverted pendulum problem with only two actions to select from, push cart left or push cart right, the function $f(x)$ in the output function equation (4.2.8) can simply be the threshold activation function defined by

equation (4.3.1),

$$f(x) = \begin{cases} +1 \quad if \ x \geq 0 \ \text{(control action right)} \\ -1 \quad if \ x < 0 \ \text{(control action left)} \end{cases} \tag{4.3.1}$$

where α, δ are constants in the range $[0, 1]$. The role of the ASE is to generate policies that will lead to optimal actions and hence optimal control when in any given state provided by the current heuristic reinforcement value $\hat{r}(t)$. Thus the ASE is the policy generator in this reinforcement learning method.

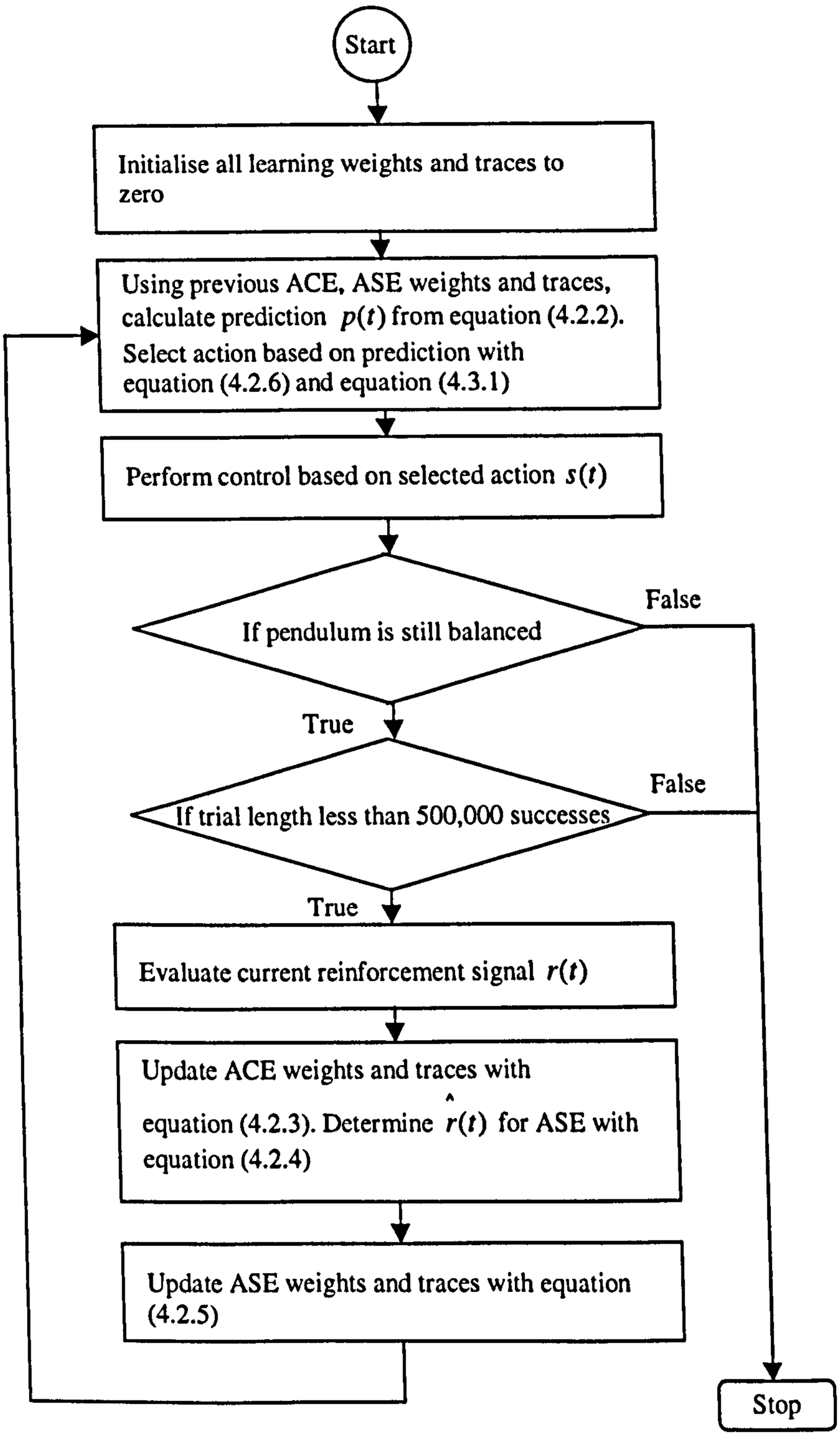

Figure 4.2: Flow chart for learning to balance an inverted pendulum using temporal difference learning control.

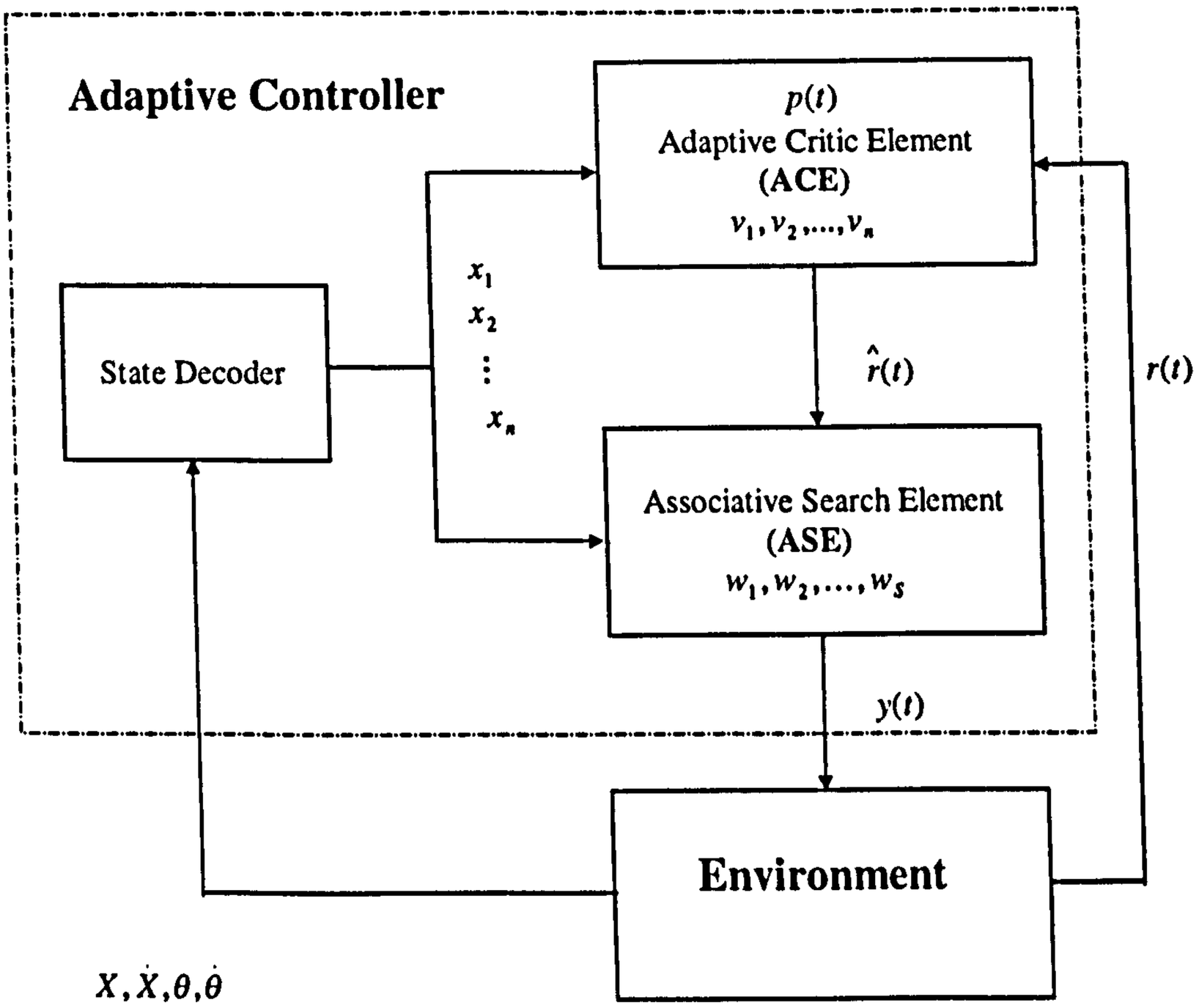

Figure 4.3: The task of balancing an inverted pendulum using the adaptive heuristic critic architecture and temporal difference reinforcement learning. The ACE and ASE can each be implemented using a neural network.

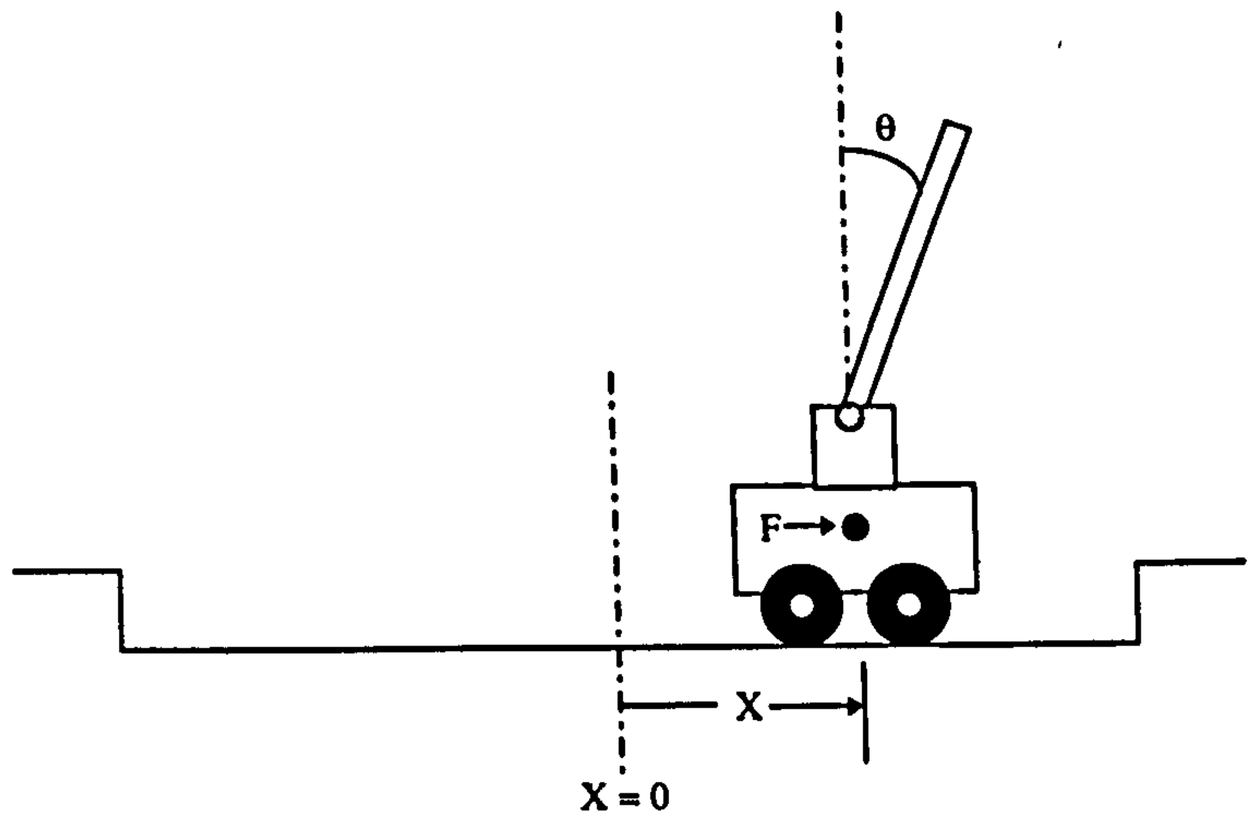

Figure 4.4: Inverted pendulum control problem.

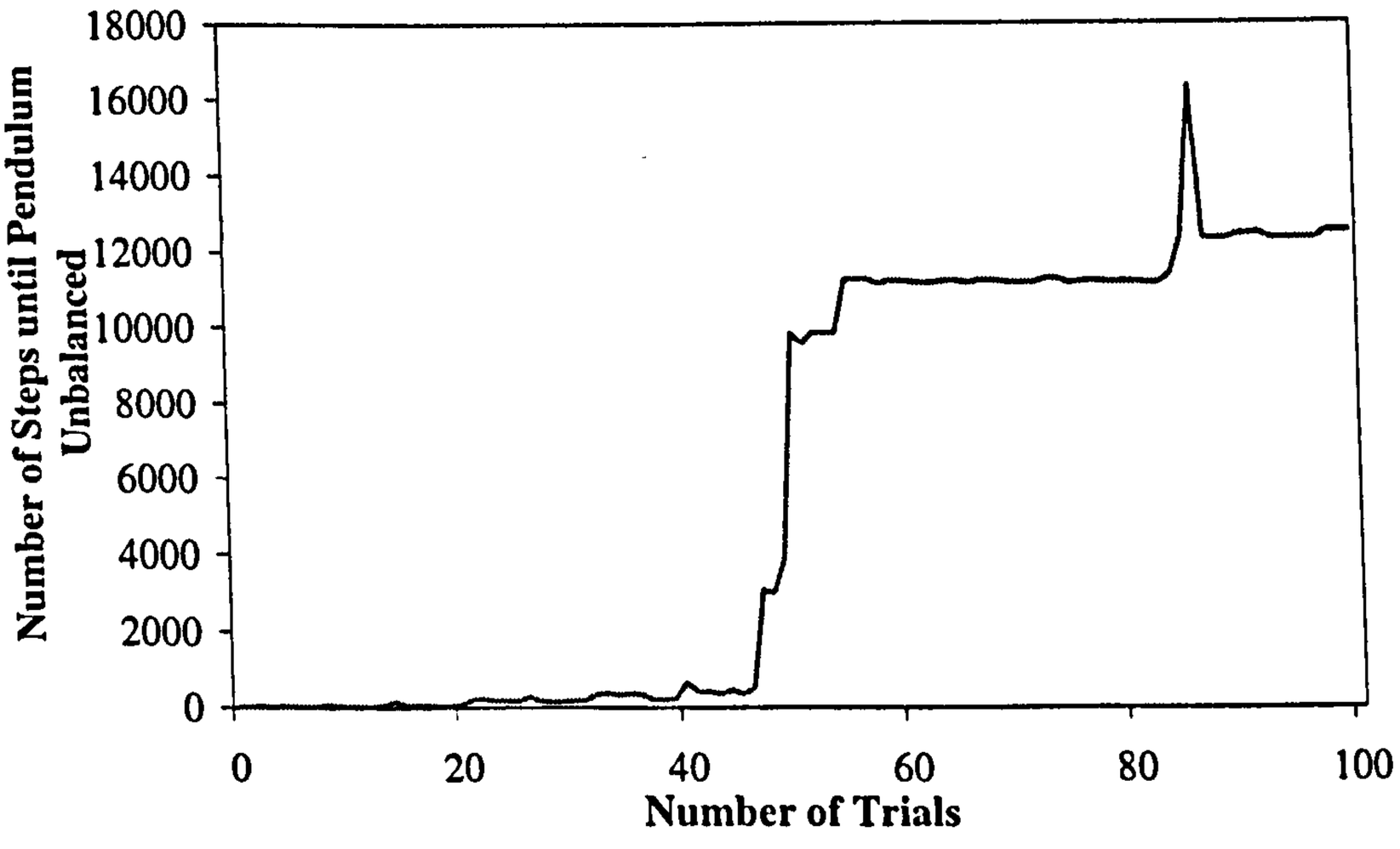

Figure 4.5: Inverted pendulum balancing simulation results.

Chapter 5

Learning Control of Dynamic Systems

Learning control is an intelligent form of adaptive control. Adaptive control allows systems to respond to problems in the environment. The ability to learn from changes due to unforeseen (or un-anticipated) environmental circumstances goes one step further than just adaptive control. The flexibility of reinforcement learning makes it suitable for applications in which exact knowledge of a system, at all times is unknown. Such a system is usually dynamic and often stochastic in nature. The stochastic nature of some problems also makes it difficult to apply conventional adaptive control, thus making reinforcement learning an invaluable alternative, and sometimes the only option.

5.1 TD(λ) Learning Control

Parameterised controllers such as the PID and FLC are suitable for the implementation of learning control in which the nature of adjusting parameter values to improve a controllers performance is intuitive and progressive.

The general idea and process is better illustrated with the aid of the flow

chart, Figure 5.1and diagram, Figure 5.4.

This procedure is the most basic and describes the action selection process for optimising a single parameter in a controller. For extension to multiple parameter optimisation in a controller (as will be described later) it becomes a simple case of scaling the desired number of learning agents (one learning agent per parameter). All the learning agents work in parallel and are independent of each other, therefore no conflict of interest occurs between learning agents.

5.2 Applying TD(λ) for Control of Dynamic Systems

Temporal difference learning was used to optimise the parameters of a PID controller. A preliminary test using a single learning agent was studied to optimise the K_P parameter of a PID controller, while the other two parameters K_I and K_D were preset to fixed values, although not as flexible as a PID controller it did prove that the temporal difference reinforcement learning scheme could learn to control a simple system. The principle of the complete optimisation task is shown in Figure 5.3.

Using this principle each of the PID parameters K_P, K_I and K_D has its own learning agent to update it. The update is performed by the temporal difference reinforcement learning scheme and adaptive heuristic architecture described previously. Thus for the PID optimisation task, three learning agents were required. The simulation study comprised three parts as shown in Figure 5.3, the PID controller, the plant and the temporal difference neural network. Note that the PID controller and plant make up the learning environment. A *pseudo random binary signal* was used to excite the system during training, see Figure 5.2, y(ref) the reference for the plant output y(out) is set to zero.

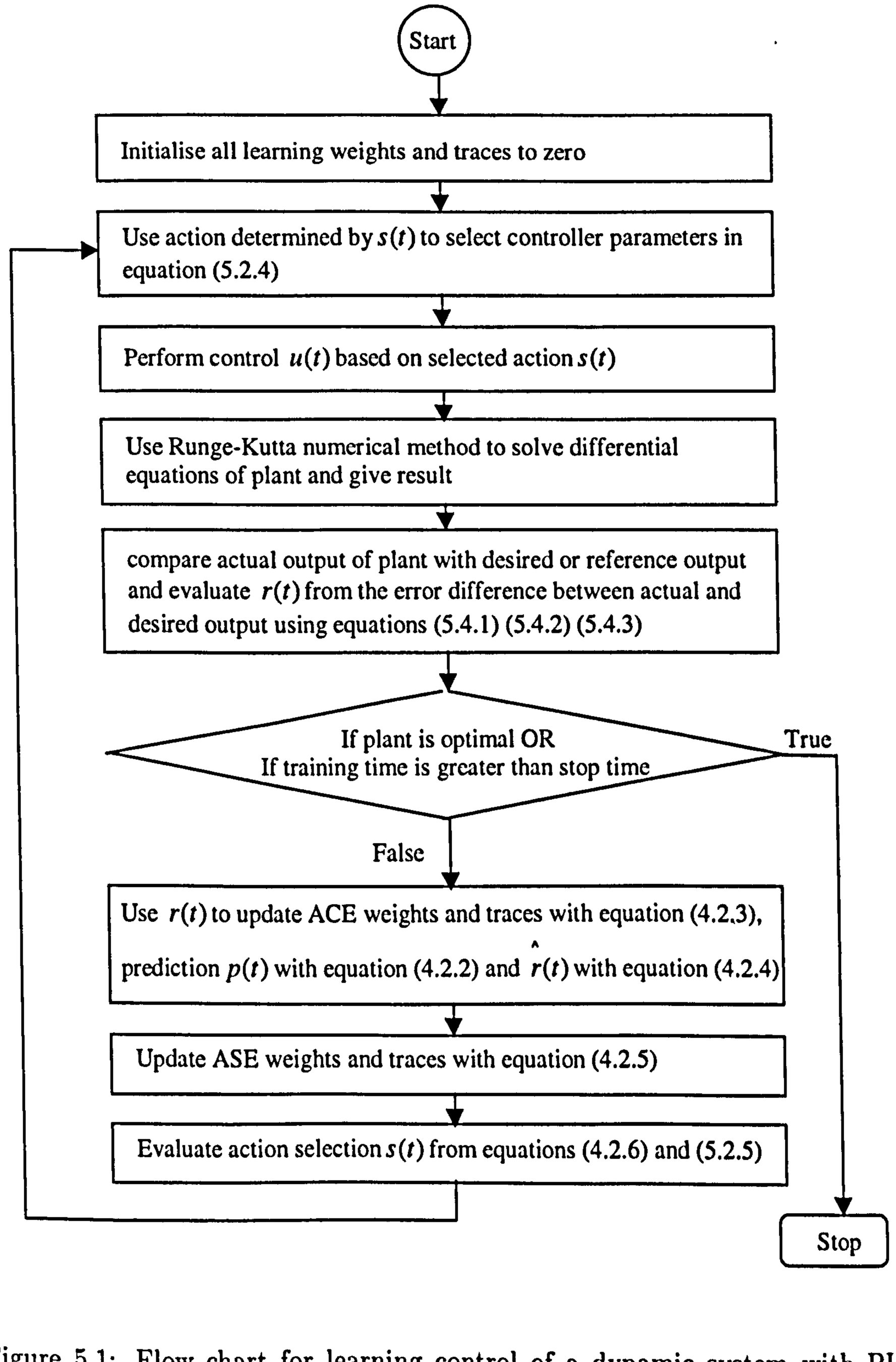

Figure 5.1: Flow chart for learning control of a dynamic system with PID control using temporal difference learning.

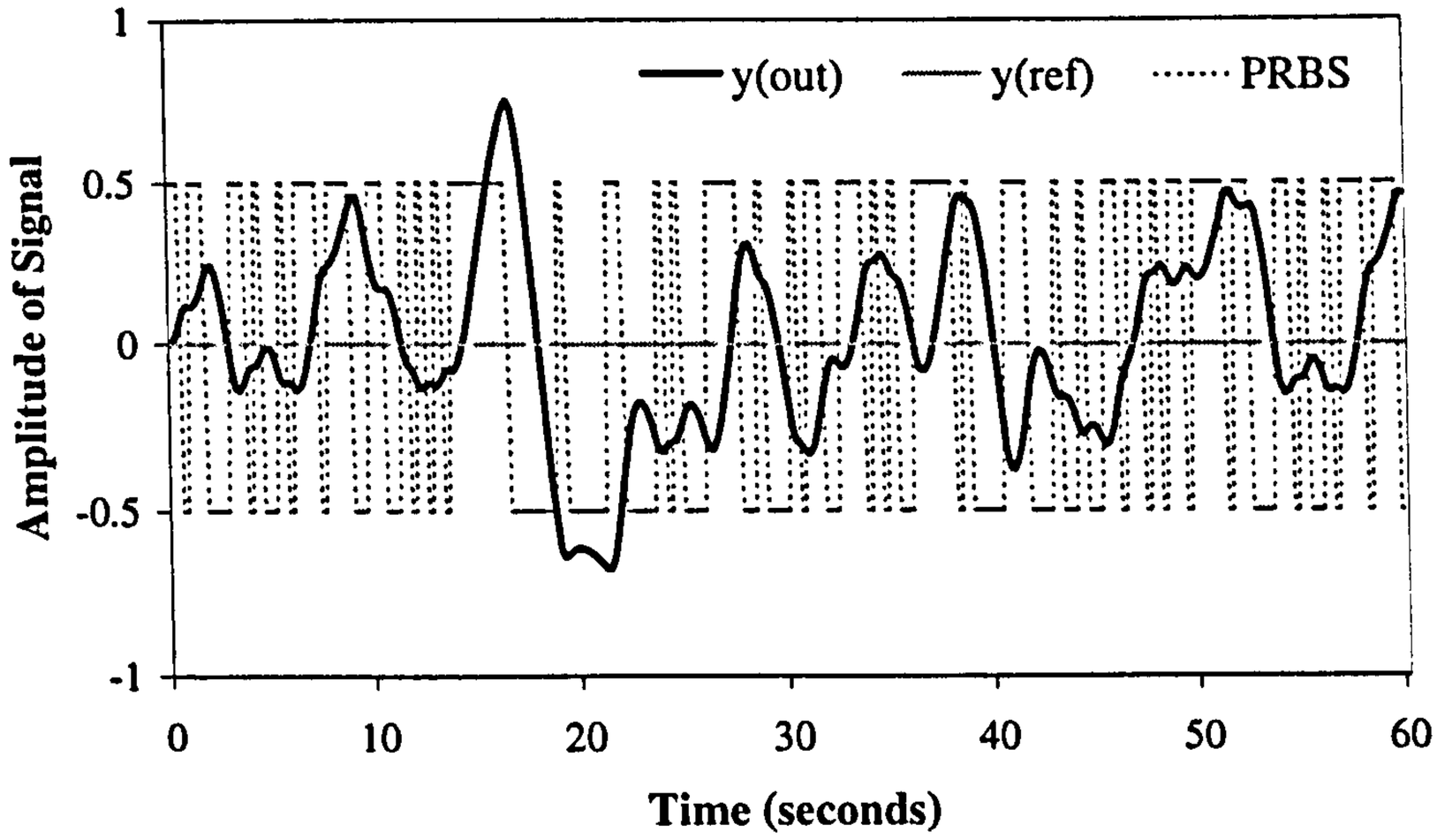

Figure 5.2: A pseudo random binary signal used to train the temporal difference neural network.

5.2.1 The Plant

For the initial testing of the temporal difference learning scheme a third order system was used to represent the plant. A third order system with corresponding transfer function is shown in equation (5.2.1). This example was used in the simulation originally came from an exercise paper in which a manual solution could be found and provided a starting point for initial testing.

$$\frac{Y(s)}{U(s)} = \frac{s+4}{s^3+8s^2+17s+10} \tag{5.2.1}$$

and in state space representation by equation (5.2.2),

$$\dot{\underline{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -10 & -17 & -8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u \tag{5.2.2}$$

with output function represented by equation (5.2.3),

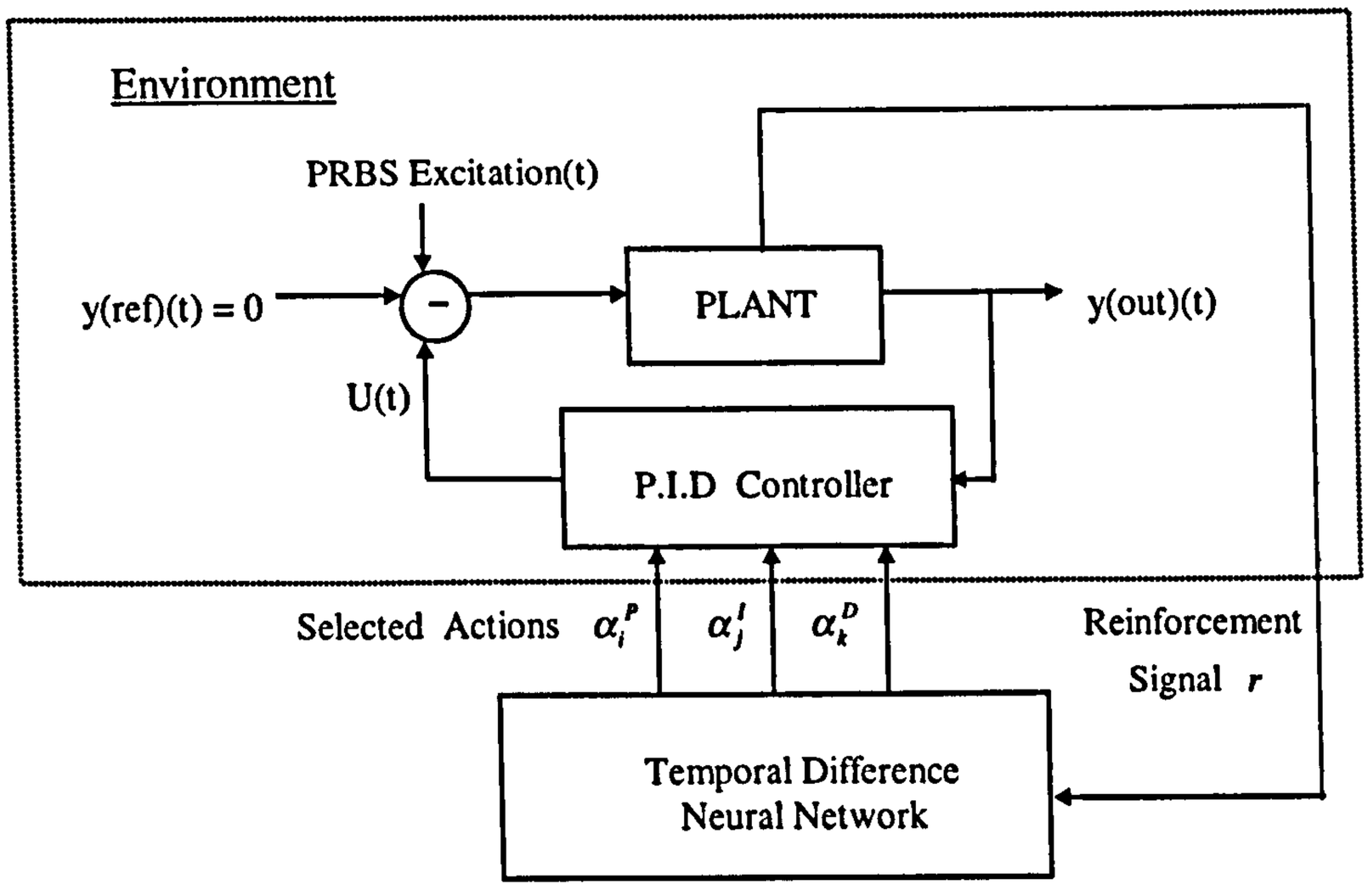

Figure 5.3: Using temporal difference reinforcement learning to optimise a PID controller.

$$y = [4 \quad 1 \quad 0] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{5.2.3}$$

5.2.2 The PID Controller

PID controllers are popular and proven systems, in fact the nature of tuning a PID controller by varying its parameters makes it ideal for learning systems that incrementally update these parameters in a continuous manner. The temporal difference learning is such an incremental learning scheme and can be used to continuously tune the PID parameters. Noise, drift and other environmental changes can potentially be corrected for by the ability of the temporal difference learning scheme to continuously learn. This is the major aim that

the discussed learning schemes try to achieve in order to ensure that the PID parameters are always optimised. The PID controller used in the simulation study was based on the one used by Wu [28] and shown in equation (5.2.4), see also Figure 5.3.

$$u(t) = u(t-1)+K_P[y(t) - y(t-1)] + K_I y(t) + K_D[y(t) - 2y(t-1) + y(t-2)] \tag{5.2.4}$$

where K_P, K_I and K_D are the PID parameters to be optimised, $y(t)$ is the feedback to the PID controller and $u(t)$ is the controller output.

5.2.3 The Temporal Difference Neural Network

At first the action selection for the temporal difference neural network was only limited to selecting the K_P parameters while K_I and K_D were fixed. Figure 5.4 shows the initial configuration of the temporal difference neural network. This initial configuration is very similar to the one used by Sutton, Barto and Anderson in their inverted pendulum control task [34].

Modifications were made to the threshold activation function $f(x)$ in equation (4.2.8), this has been replaced by a *sigmoid* activation function, such as the *logistic* function given by equation (5.2.5),

$$f(x) = \frac{2}{1 + exp(-ax)} \tag{5.2.5}$$

where a is a constant and determines the slope of the *sigmoid* function. The threshold function used in the inverted pendulum example only assumes discrete values of 0 or 1, but the *sigmoid* function assumes a continuous range of values between 0 and 1. For the PID controller parameter selection a *sigmoid* activation function is preferred. A total of ten actions were made available to the ASE output $s(t)$ to select K_P. The actions of the temporal difference neural network were divided into discrete values in a set range. The range

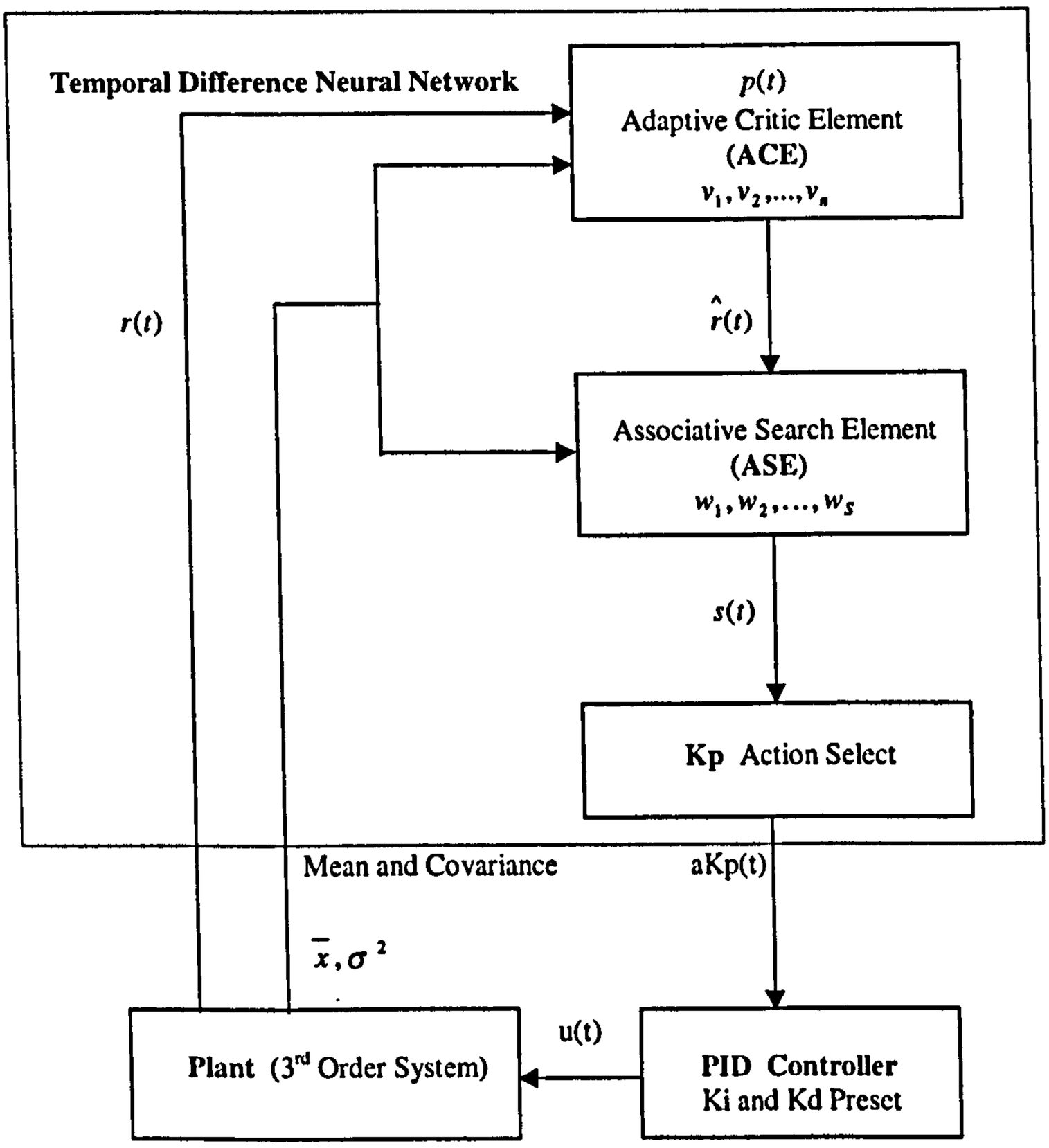

Figure 5.4: PID controller using temporal difference reinforcement learning to select K_P parameter with K_I and K_D preset.

was chosen after preliminary tests indicated where the best PID coefficients would be. If a learning scheme is given enough time to perform a thorough search of any large range then learning will eventually find an optimal solution, in order to save time in the learning process the preliminary tests narrowed the potential search range for each parameter in order to speed up learning. The main point of the tests and simulations was in order to see if learning would converge and select optimal actions. A long learning time will always be a potential weakness for all learning schemes which rely on minimal initial

knowledge of their environments.

For the initial testing when only K_P is optimised the actions are shown in Table 5.1, which also shows the eventual action selections available for the other PID controller parameters. Each parameter is given a choice of ten actions equally subdivided from a range of values. Each action available is chosen by using the output of the activation function which outputs a continuous value between zero and one. The output of the *sigmoid* activation function is divided into ten regions with each region representing one action, so for example if the *sigmoid* activation function outputs a value in the range $\{0 \leq output < 0.1\}$, then the first action is chosen which represents the first value of the PID controller parameter $\{K_P[i] : i = 0\}$ choosing a coefficient of $\{K_P = 0\}$ etc... At this point K_I and K_D are not changed.

Quantised output: $i =$	0	1	2	3	4	5	6	7	8	9
action $(K_P(i))$	0	3	6	9	12	15	18	21	24	27
Quantised output: $j =$	0	1	2	3	4	5	6	7	8	9
action $(K_I(j))$	0	3	6	9	12	15	18	21	24	27
Quantised output: $k =$	0	1	2	3	4	5	6	7	8	9
action $(K_D(k))$	0	1	2	3	4	5	6	7	8	9

Table 5.1: Action selection for PID controller by temporal difference learning neural network.

To monitor the progress of learning and compare the actions selected all the action frequencies were uniformly initialised. In this case there were ten actions with an initial frequency for selecting any particular action of a on in ten chance. As the temporal difference neural network trains the action frequency will be updated, in general if the action used was good then the frequency of selecting that action is improved and vice versa. The frequency sum must be conserved and thus as one frequency was changed the others

must also be changed in an inversely proportional manner. One example of the action frequency updates for K_P is shown in Figure 5.5.

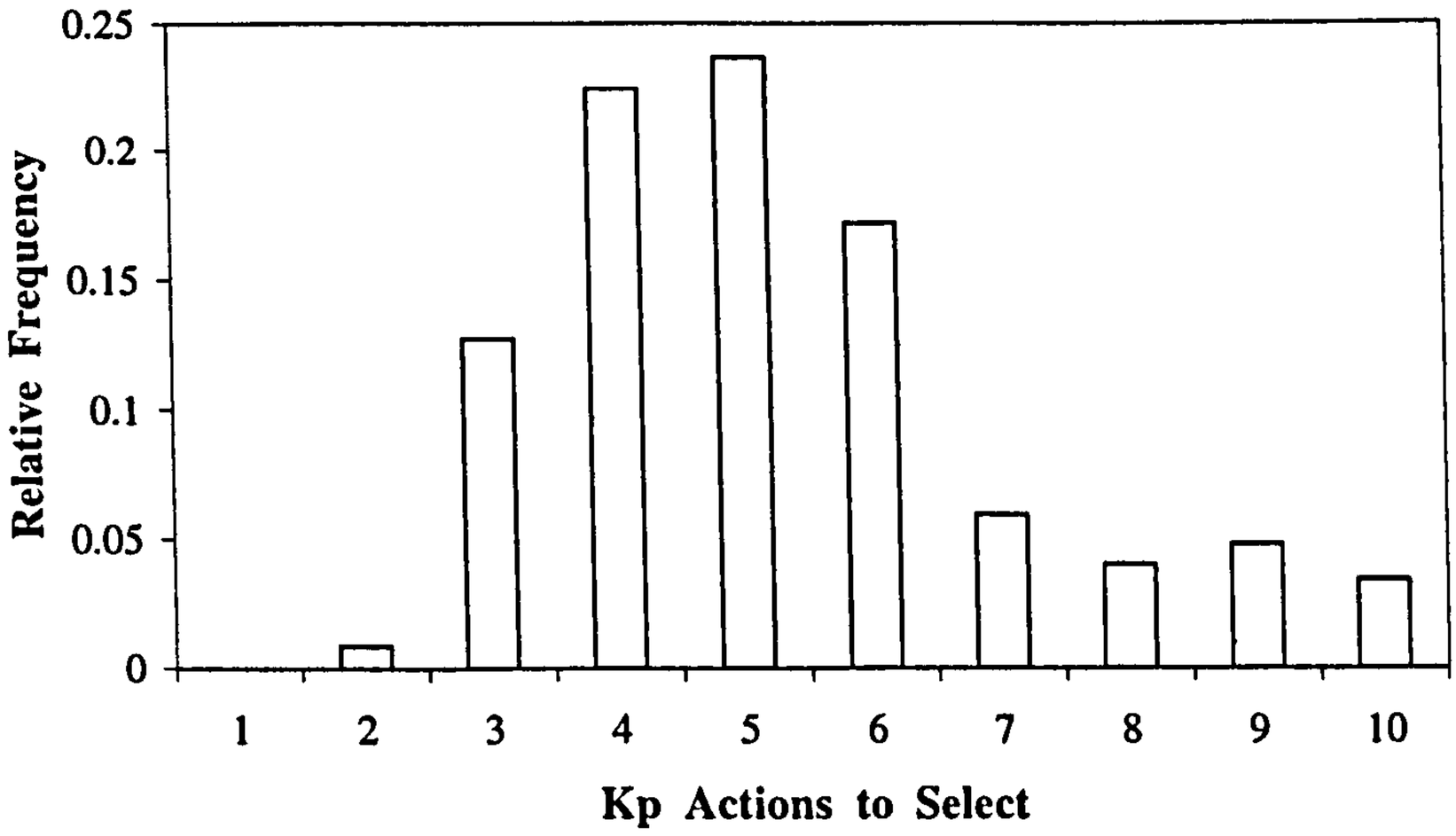

Figure 5.5: Action probabilities for the PID controller parameter K_P after training of the temporal difference neural network with K_I and K_D preset.

The input to the temporal difference neural network was formed by calculating the mean and covariance of the plant output for a given sample size or window. This was repeated at each time step to drive the learning, selection and updating of the action probabilities. Various window sizes were tried but a window size in the region of between 200 and 400 provided sufficient information for learning improvement while simultaneously allowing for a relatively quick training time.

The weights of the temporal difference neural network converge as shown in Figure 5.7 and the resulting dynamic response to a step input after training is shown in Figure 5.8. However it can be seen that the step response could be further improved, the next stage of developing the temporal difference neural network was to extend it to optimise two of the three PID controller parameters.

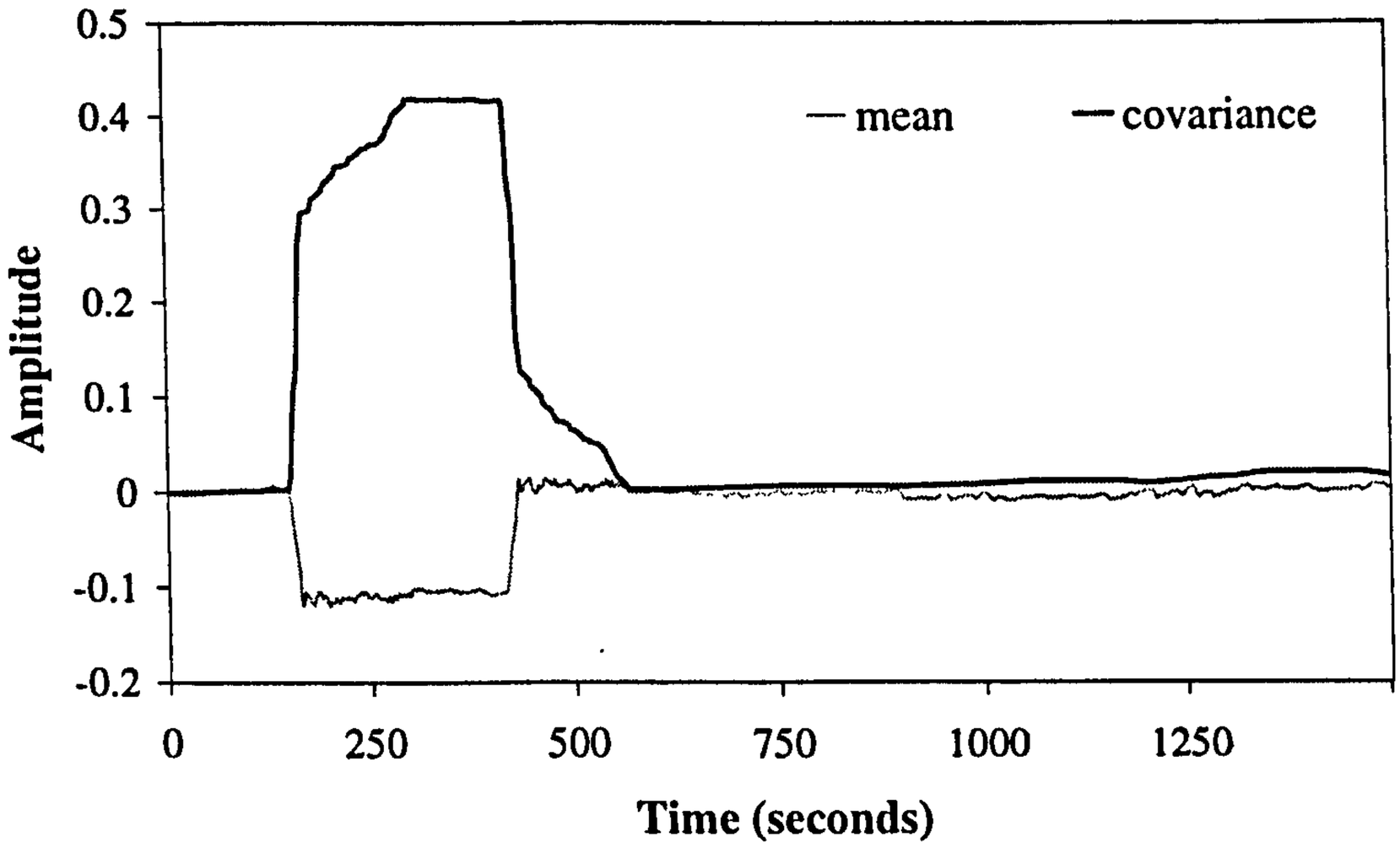

Figure 5.6: Mean and covariance for y(out) from plant and input to the temporal difference neural network, with a window size of 300 samples.

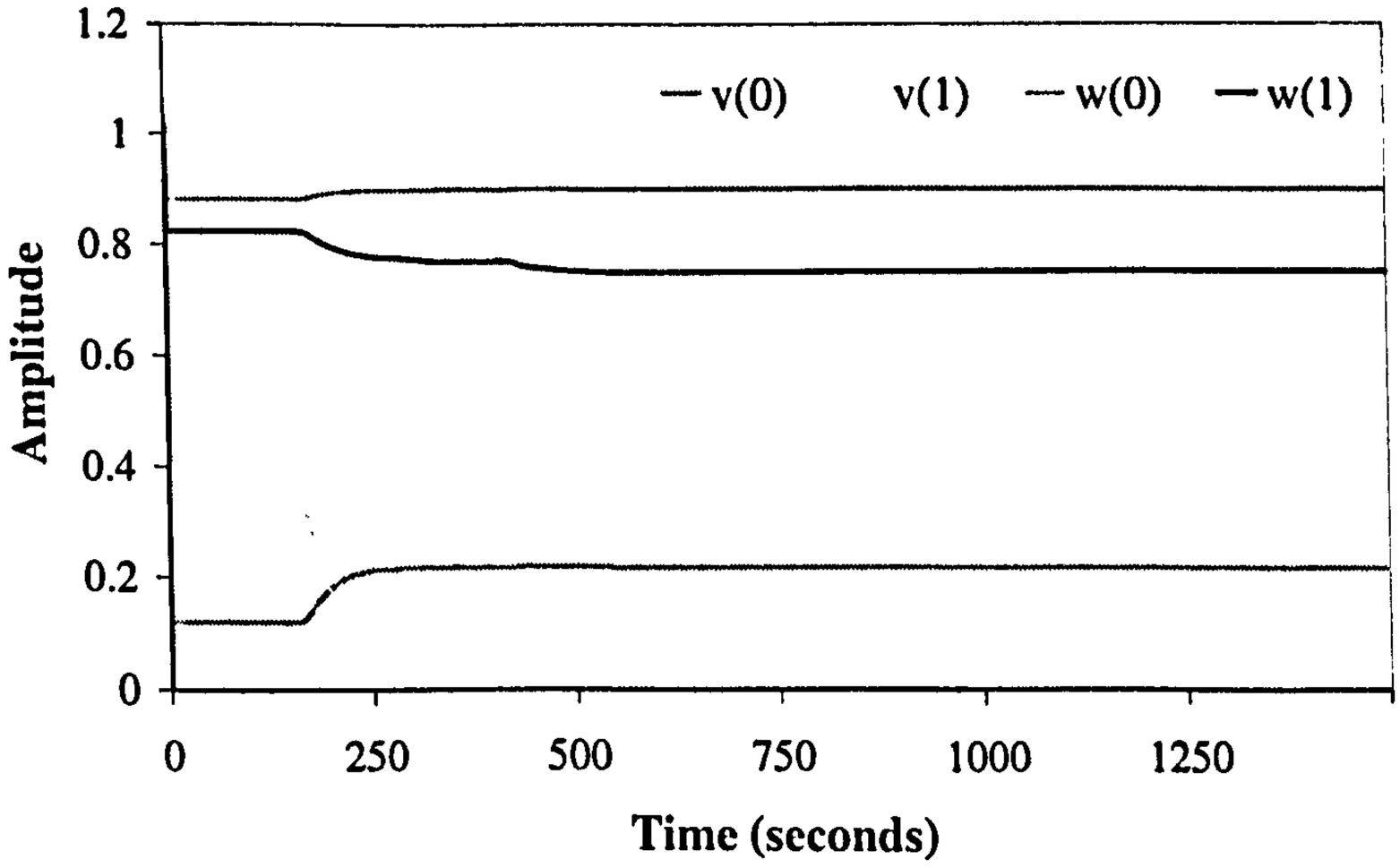

Figure 5.7: The temporal difference neural network weights from the ACE and ASE for optimising the K_P parameter of the PID controller with K_I and K_D preset.

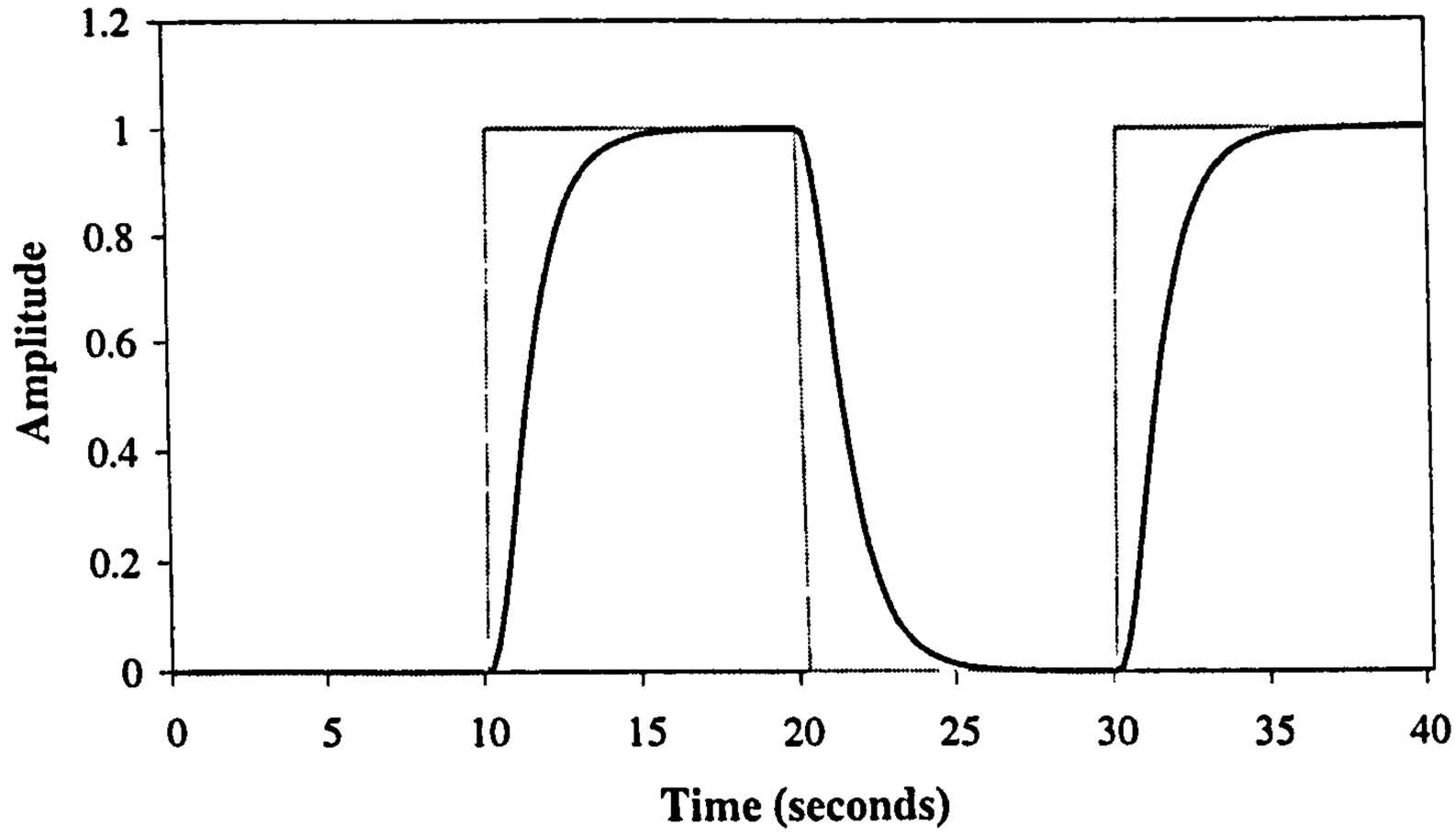

Figure 5.8: Step response of the plant with the PID controller parameter K_P optimised by the temporal difference neural network where K_I and K_D are preset.

This initial test however illustrates the value of using a temporal difference neural network.

5.3 Optimising PID Control Parameters Using A TD(λ) Neural Network

The simplest way to extend the temporal difference neural network was to treat the ACE/ASE pair as a single learning agent and have a learning agent optimise one of the PID controller parameters, this is very much like the team of learning automaton principle by Wu [28], but in this case using temporal difference learning agents. The K_P and K_D parameters were optimised in the next test with the K_I parameter left preset as before. The architecture for optimising these two PID controller parameters is shown in Figure 5.9. Thus a single learning agent when using a temporal difference neural network com-

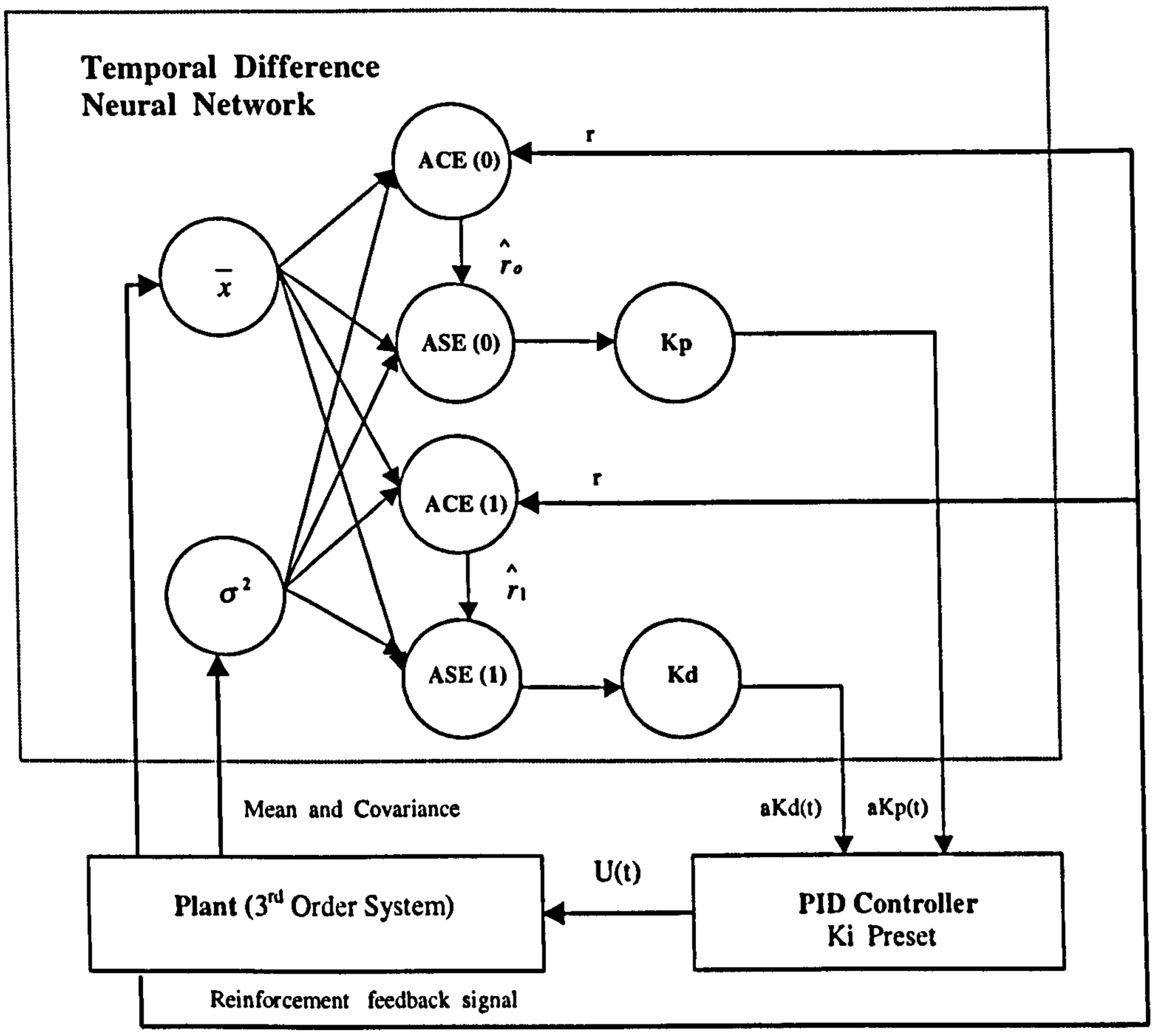

Figure 5.9: Optimisation of K_P and K_D parameters of a PID controller using a temporal difference neural network.

prises a *critic* the ACE, an *action selector*, the ASE and an output neuron, such as a *sigmoid* activation function. The inputs to the temporal difference neural network have been unchanged and still comprise the mean and covariance of the plant output y(out).

From the action probability results for selecting K_P in Figure 5.10 and K_D in Figure 5.11 it can be seen that the resulting step response Figure 5.12 of the plant due to the selected parameters has improved over the result of the previous test, when only K_P was optimised. This gave further confidence that

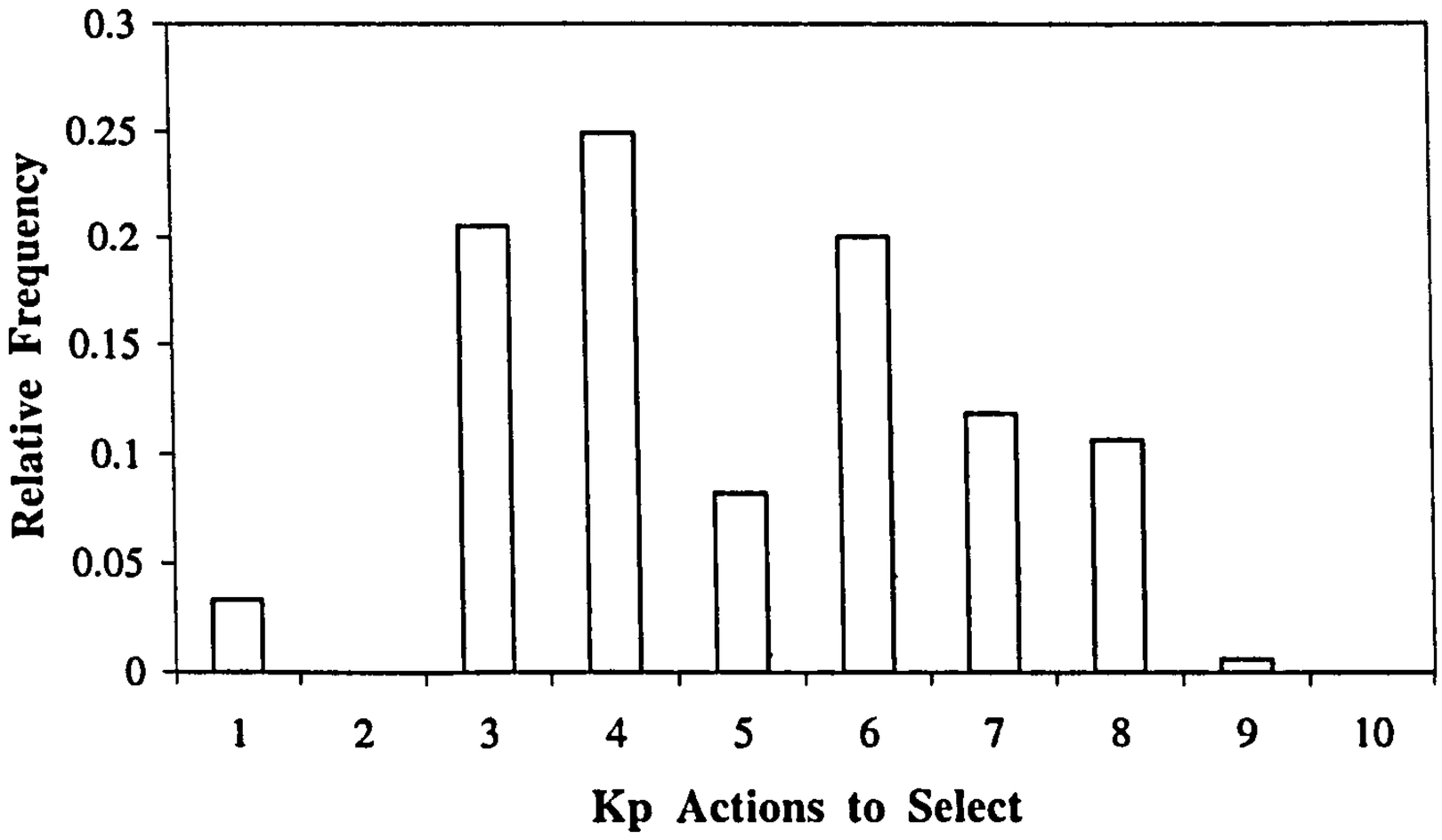

Figure 5.10: K_P action probabilities after training of temporal difference neural network to optimise K_P and K_D parameters of a PID controller with K_I preset.

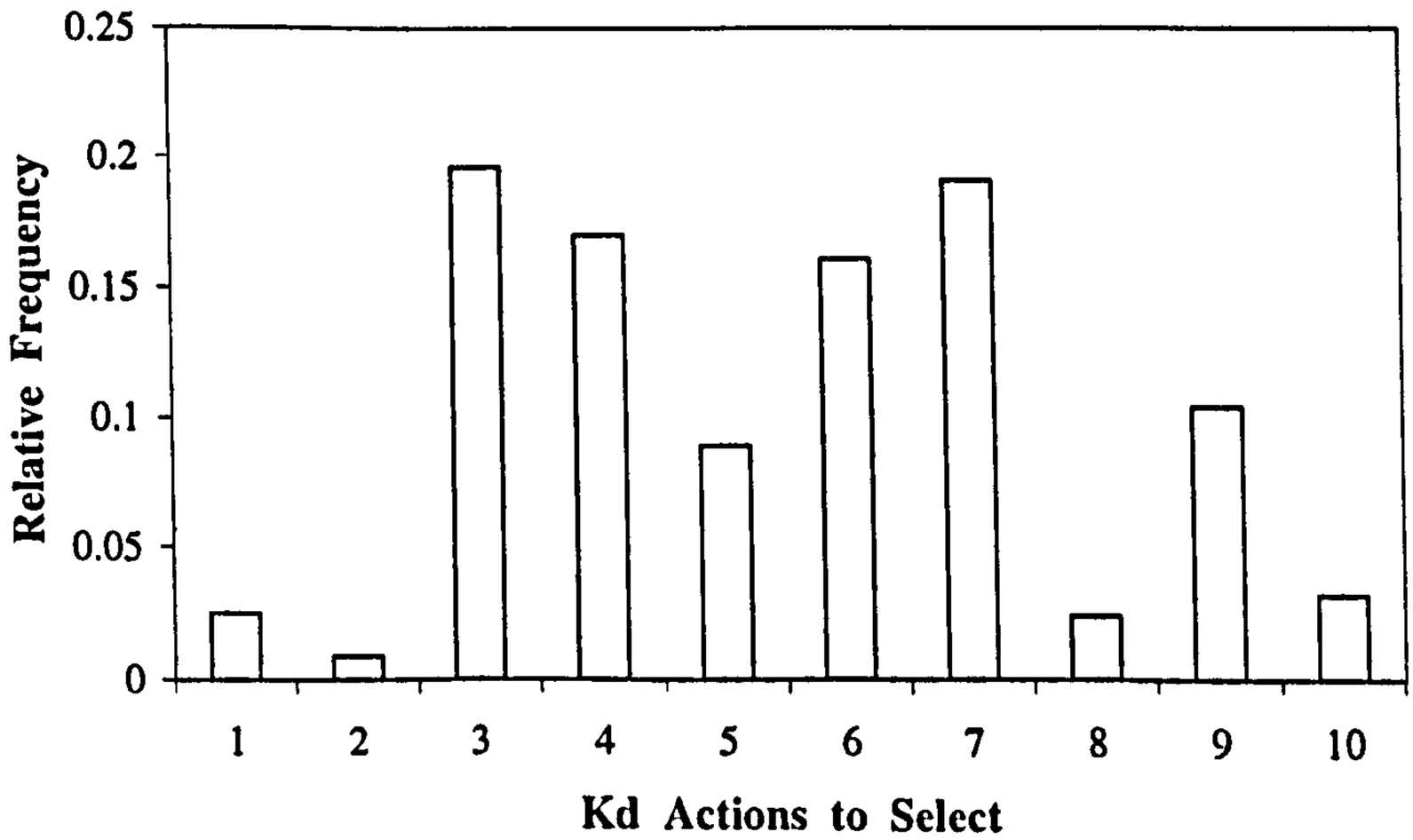

Figure 5.11: K_D action probabilities after training of temporal difference neural network to optimise K_P and K_D parameters of a PID controller with K_I preset.

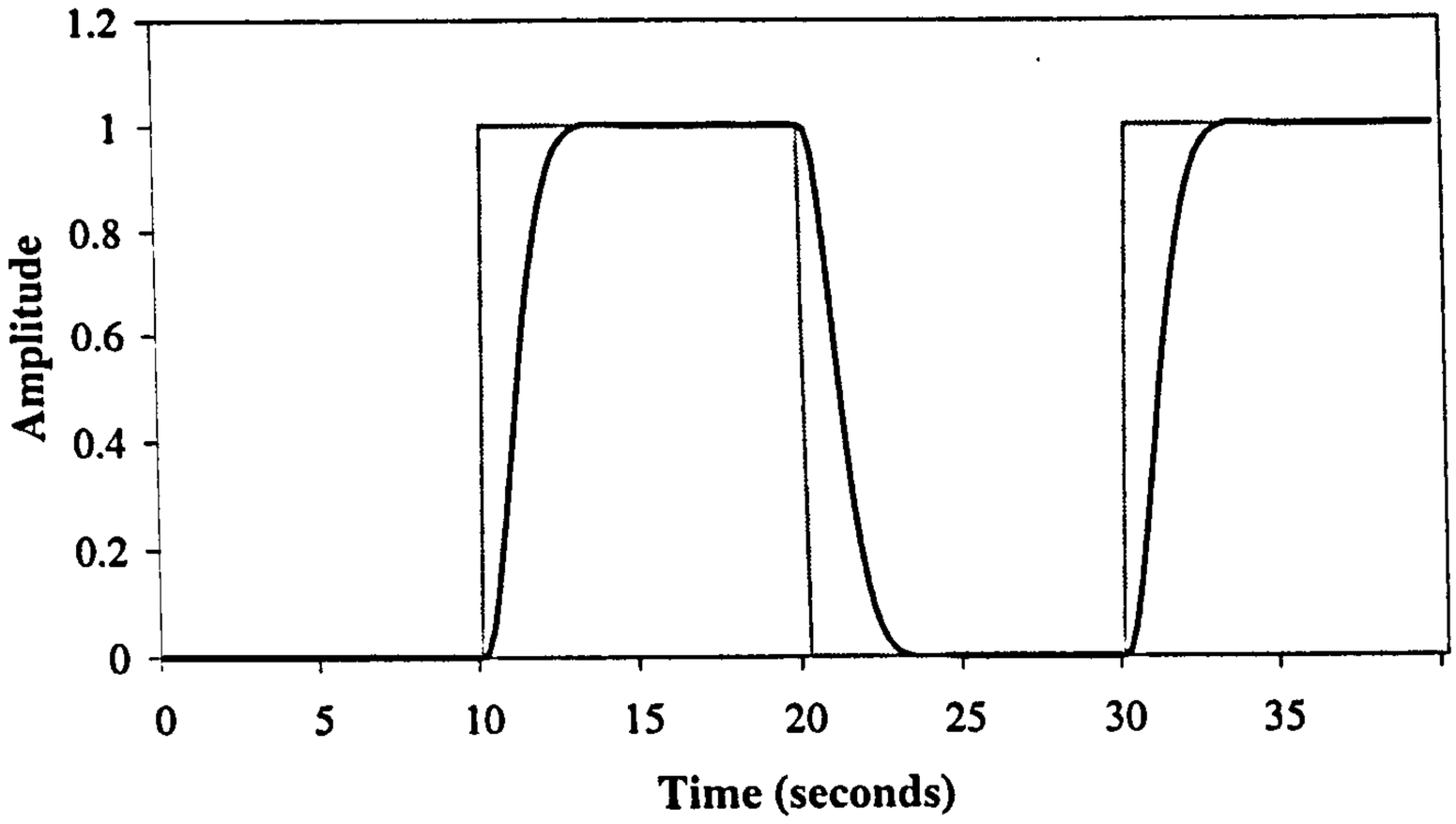

Figure 5.12: Step response of the plant with the PID controller parameters K_P and K_D optimised by a temporal difference neural network where K_I is preset.

the eventual use of three temporal difference learning agents, each optimising a single PID controller parameter would also prove successful.

5.4 Optimising K_P, K_I and K_D

The final test was to try and extend the temporal difference neural network to optimise all three PID controller parameters. Again using the principle that one learning agent consists of a *critic*, *action selector* and an output *sigmoid* activation function, the resulting architecture shown in Figure 5.13 was used.

After initial testing of the full temporal difference neural network, it was found that the action selection process by the output neurons did not converge very well to the optimum actions. Although most of the time the optimal actions were chosen the other actions had action probabilities very similar to the optimum action. An example of this is shown in Figure 5.14 where actions two, three and four have similar probabilities to each other even though there

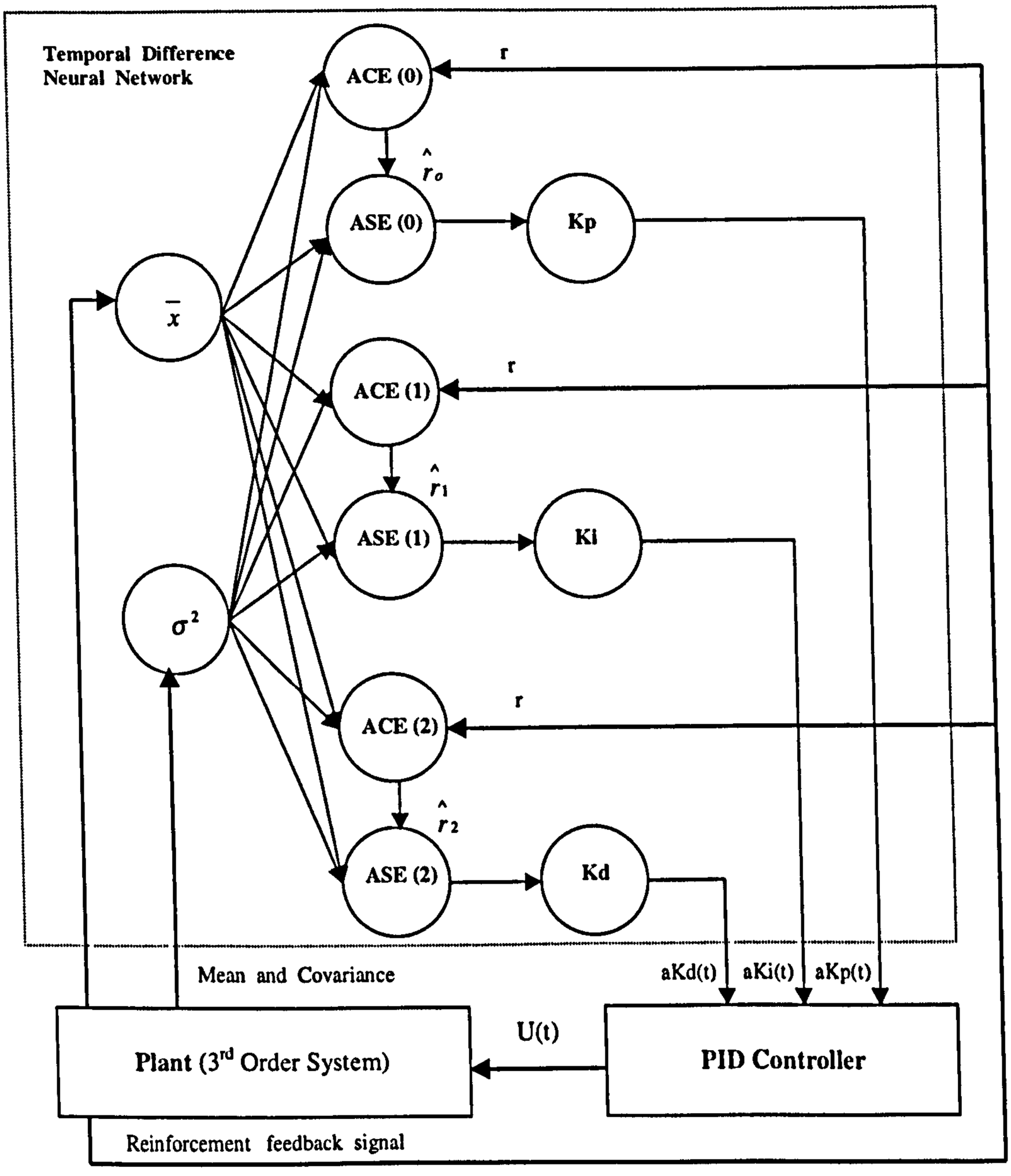

Figure 5.13: Optimisation of PID parameters using a temporal difference neural network.

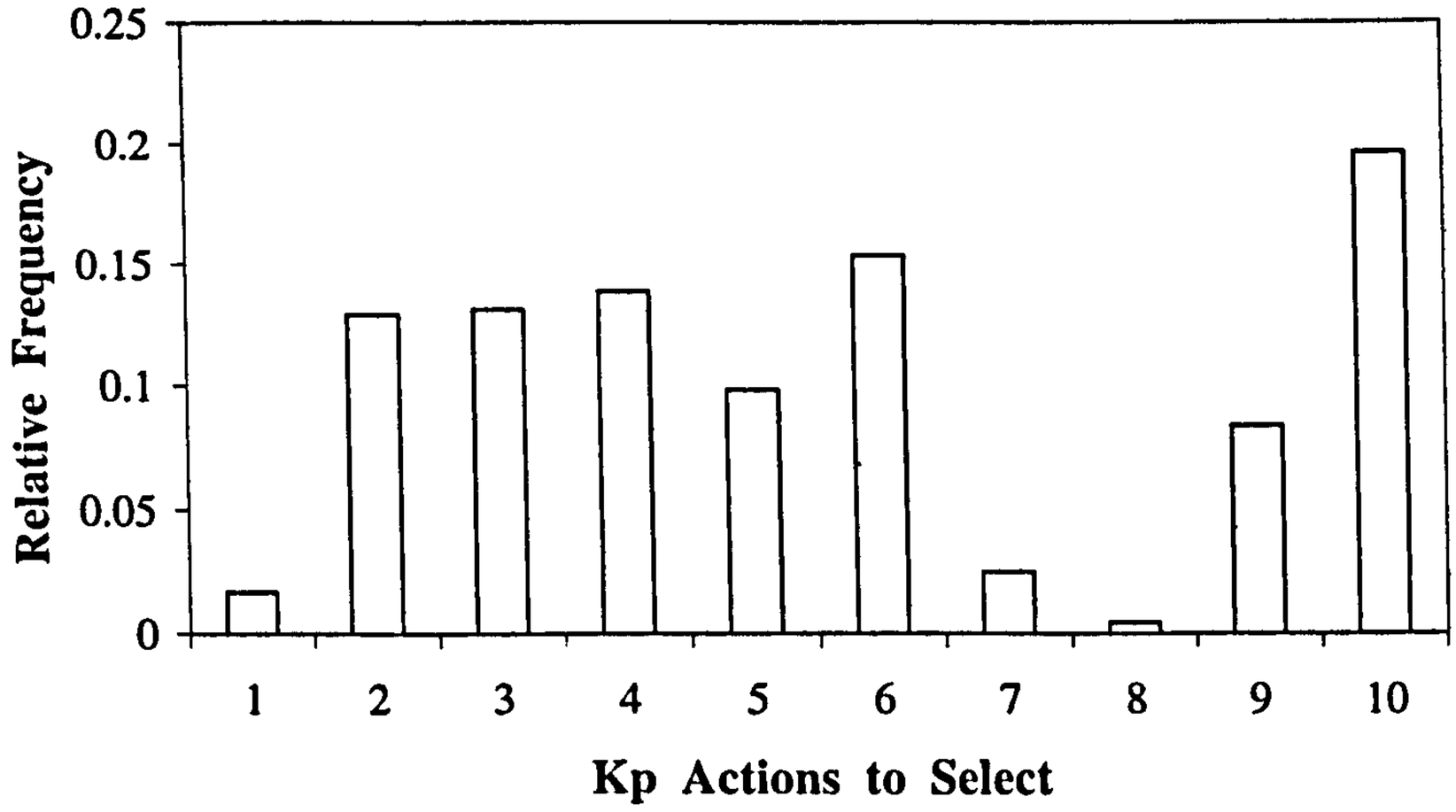

Figure 5.14: Action probabilities that do not converge well to one action.

is an optimal action (the tenth action) the probability of this isn't large enough to be decisively selected each time the PID controller parameters are chosen so a bad action being chosen is still just as likely.

In order to overcome this the learning rates of the temporal difference learning neural network algorithm, namely α, β, γ, δ and the forgetting factor λ in equation (4.2.3) and equation (4.2.6) were adjusted to see if there would be a better response from the neural network. Another factor in improving the action selection process was to change the noise distribution in equation (4.2.8) which affects the exploration of actions and normalising the feedback signal r similar to the method used in the learning automata and given by equations (5.4.1), (5.4.2) and (5.4.3). Where r becomes a cost function and is used to measure the performance of the controller after each action selection. The cost function was obtained using equation (5.4.1),

$$J(k) = \frac{1}{N}\sum_{i=1}^{N}(y(ref) - y(out))^2 \tag{5.4.1}$$

where N is the window size or sampling interval for calculating the mean and covariance of the outputs $y(out)$, where $y(ref) = 0$. In order to use this cost measure as the reinforcement feedback signal it must be normalised first. This was achieved using equation (5.4.2),

$$r(k) = \frac{J_{\max}(k) - J(k)}{J_{\max}(k) - J_{\min}(k)} \tag{5.4.2}$$

where

$$J_{\max}(k) = \left\{ \begin{array}{cc} J(k) & \text{if } J(k) > J_{\max}(k-1) \\ J_{\max}(k-1) & \text{otherwise} \end{array} \right\} \tag{5.4.3}$$

$$J_{\min}(k) = \left\{ \begin{array}{cc} J(k) & \text{if } J(k) < J_{\min}(k-1) \\ J_{\min}(k-1) & \text{otherwise} \end{array} \right\}$$

The original form of the feedback signal r was purely just 1 or 0, with 1 representing a good action for when the covariance was calculated to some upper limit for example $r = 1$ when $covariance < 0.5$ else $r = 0$. This meant that the feedback signal returned a discrete signal that represented a completely good action or a completely bad action but not a continuous feedback signal representing various levels of success and failure. The purpose of equation (5.4.2) was used to rectify this and improve the action selection process.

It was found that the convergence of the neural network weights were very sensitive to changes in the learning rates α, β, γ and δ. Also by increasing the amount of noise present in the ASE output in equation (4.2.8) the exploration of the action space by the learning agents increases, however there was a limit to the amount of exploration that was desirable due to the exploration/exploitation nature of all learning tasks. There is a balance which

must be found (and is different for differing tasks) between exploring and using existing knowledge in order to choose optimal actions to complete a given task. Therefore to discover such actions, it has to try actions that it has not selected before or to *explore* as well as having to *exploit* what it already knows in order to obtain better rewards.

5.5 Results for Optimised PID Controller

The final simulation results show the potential possibility of temporal difference learning for the control large-scale systems. This example also provides a practical use of temporal difference learning beyond that of playing complex games such as backgammon and draughts. The results after optimising the PID Controller parameters are shown in Figures 5.15 to Figure 5.20. As shown the weights converge quickly during training and the step response of the plant after selecting the optimal PID controller parameters is satisfactory.

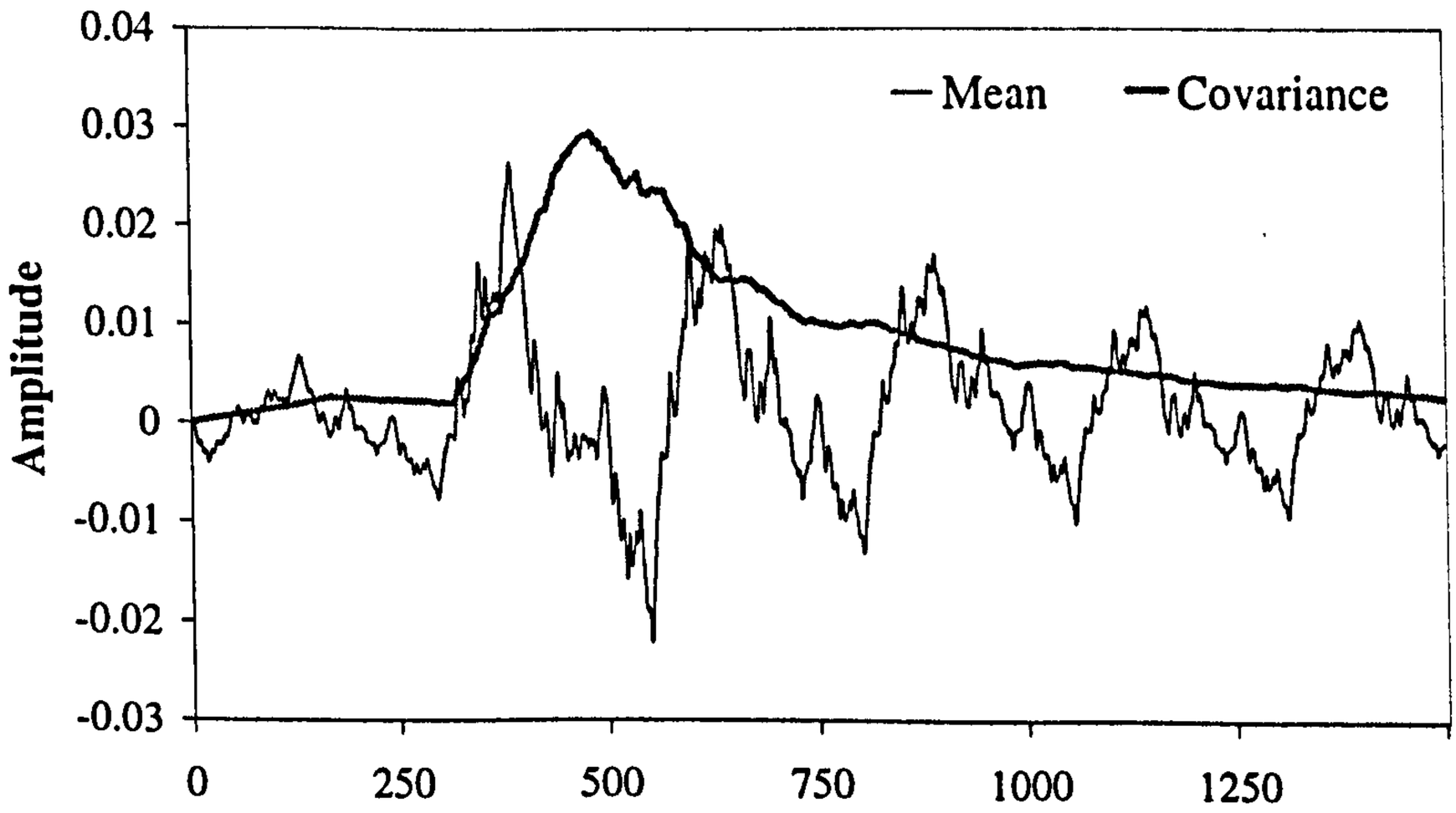

Figure 5.15: Mean and covariance input used to train the temporal difference neural network for optimising a PID controller.

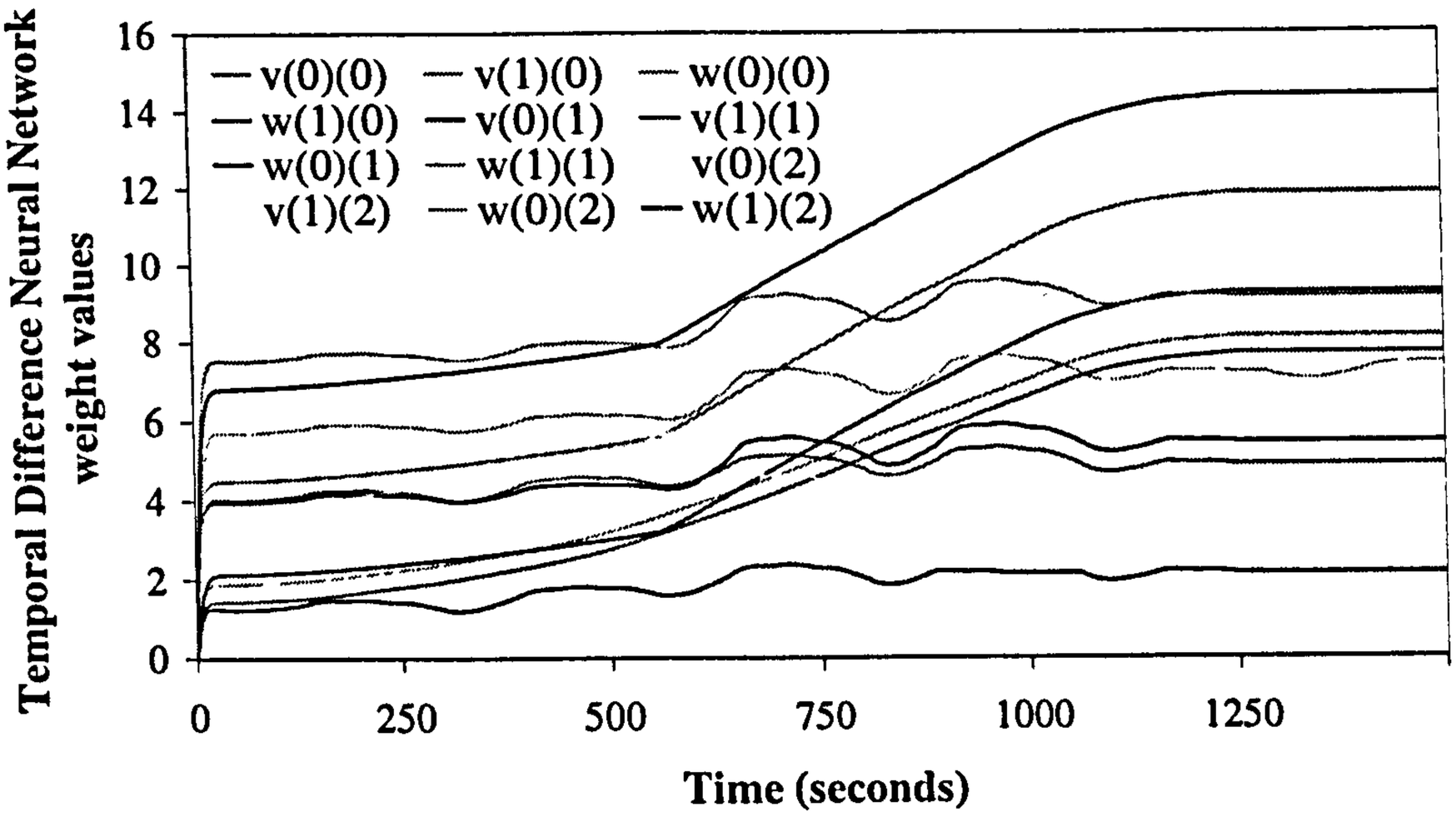

Figure 5.16: Temporal difference neural network weights for optimised PID controller.

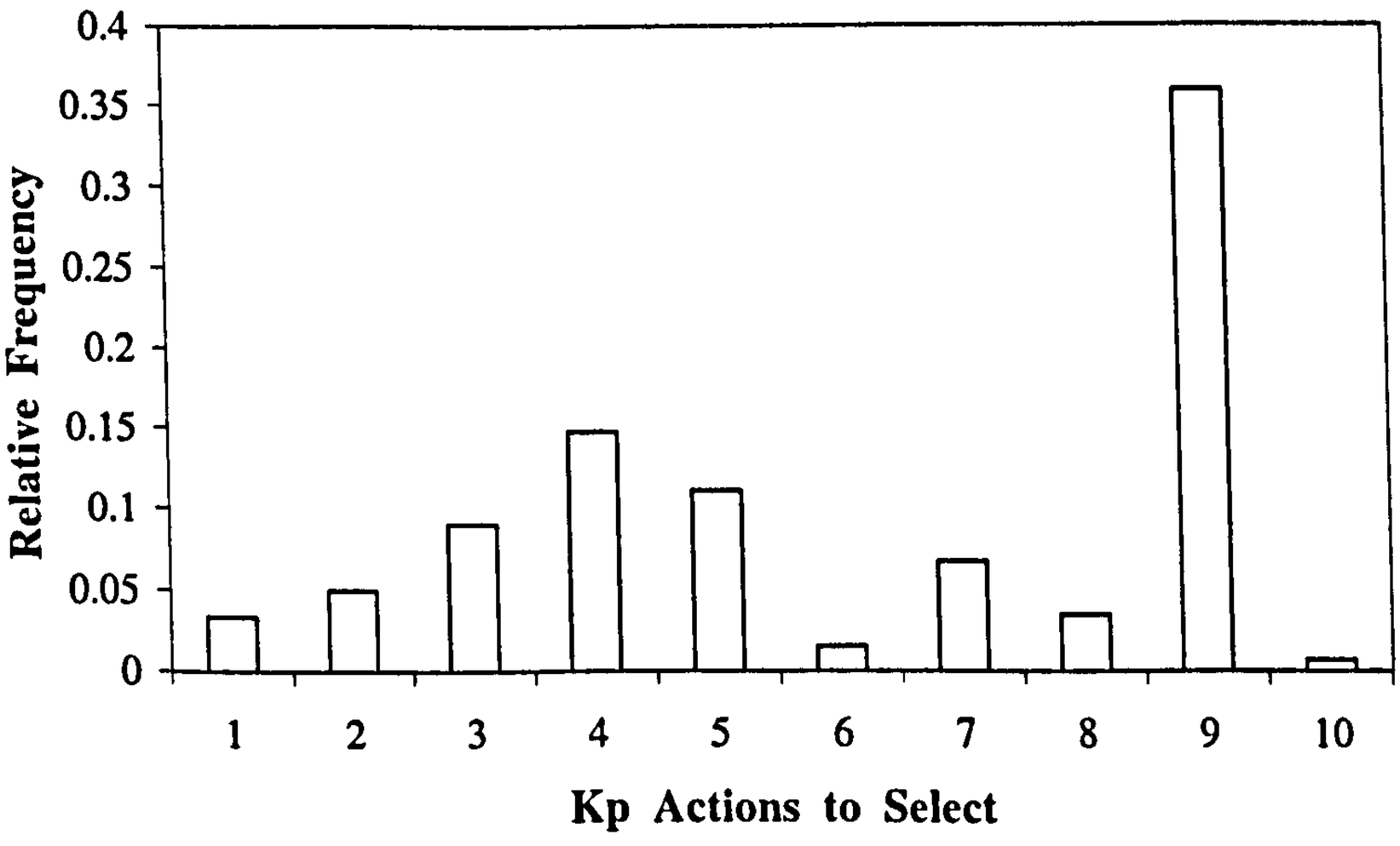

Figure 5.17: K_P action probabilities for optimised PID controller using a temporal difference neural network.

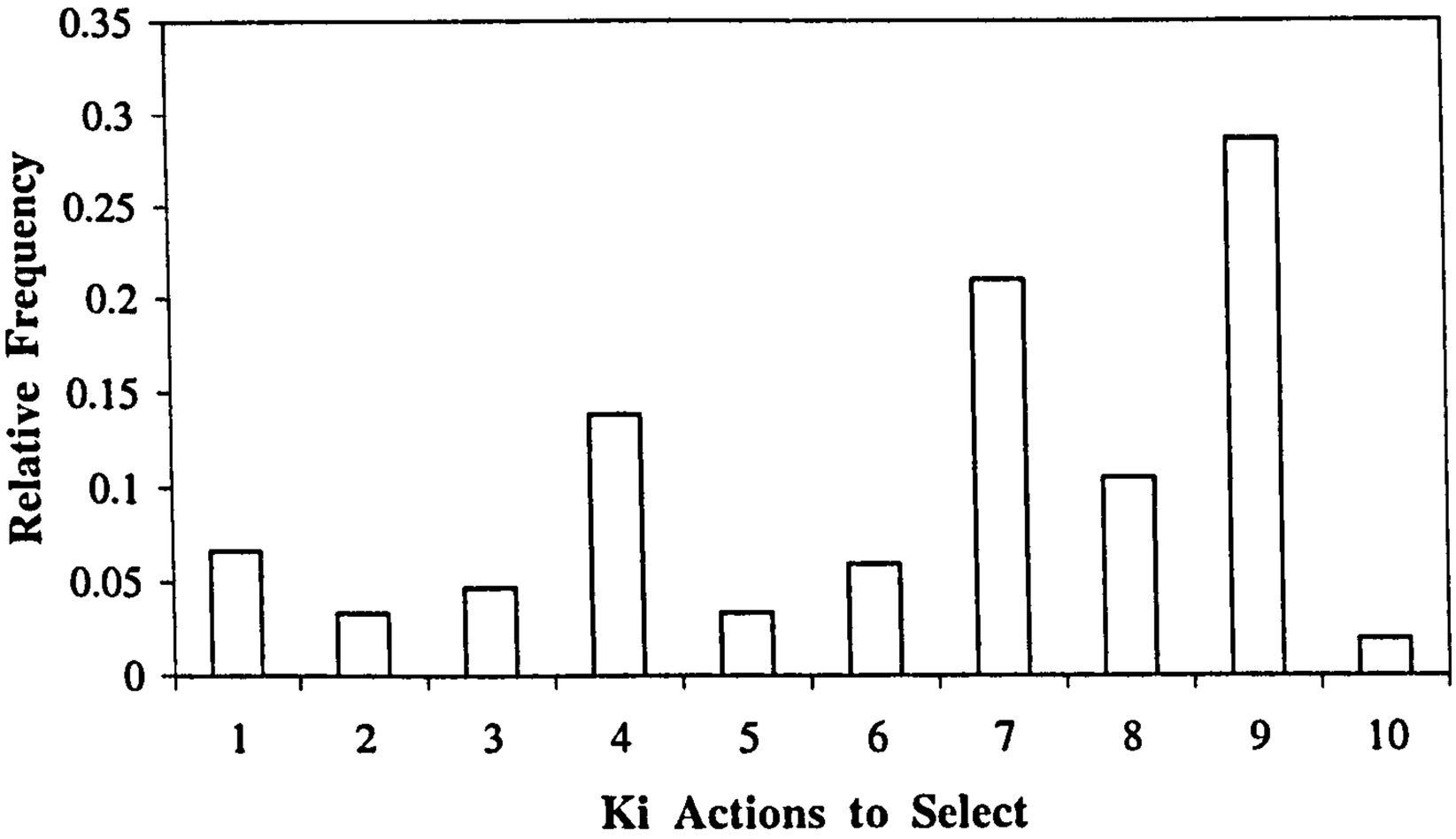

Figure 5.18: K_I action probabilities for optimised PID controller using a temporal difference neural network.

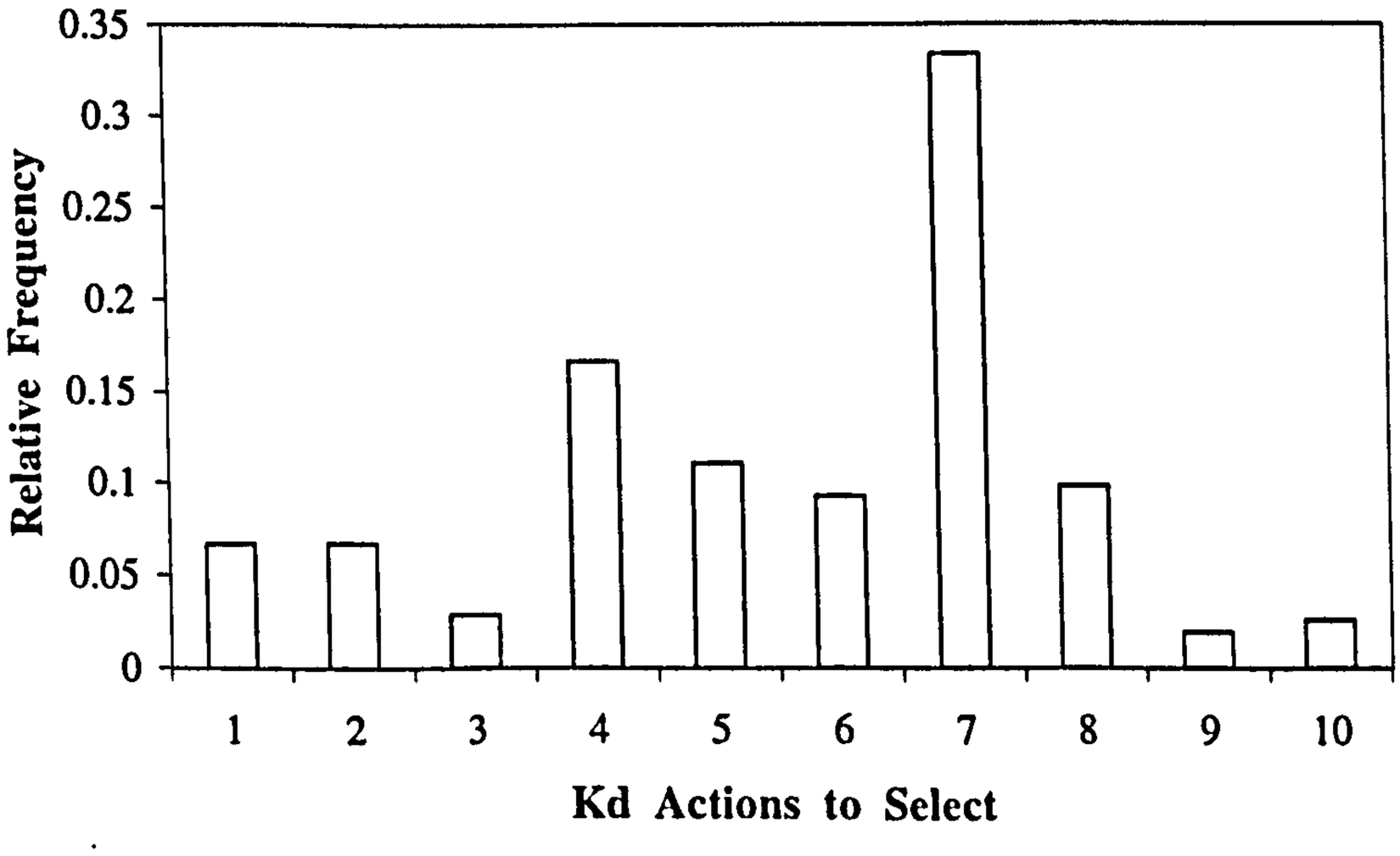

Figure 5.19: K_D action probabilities for optimised PID controller using a temporal difference neural network.

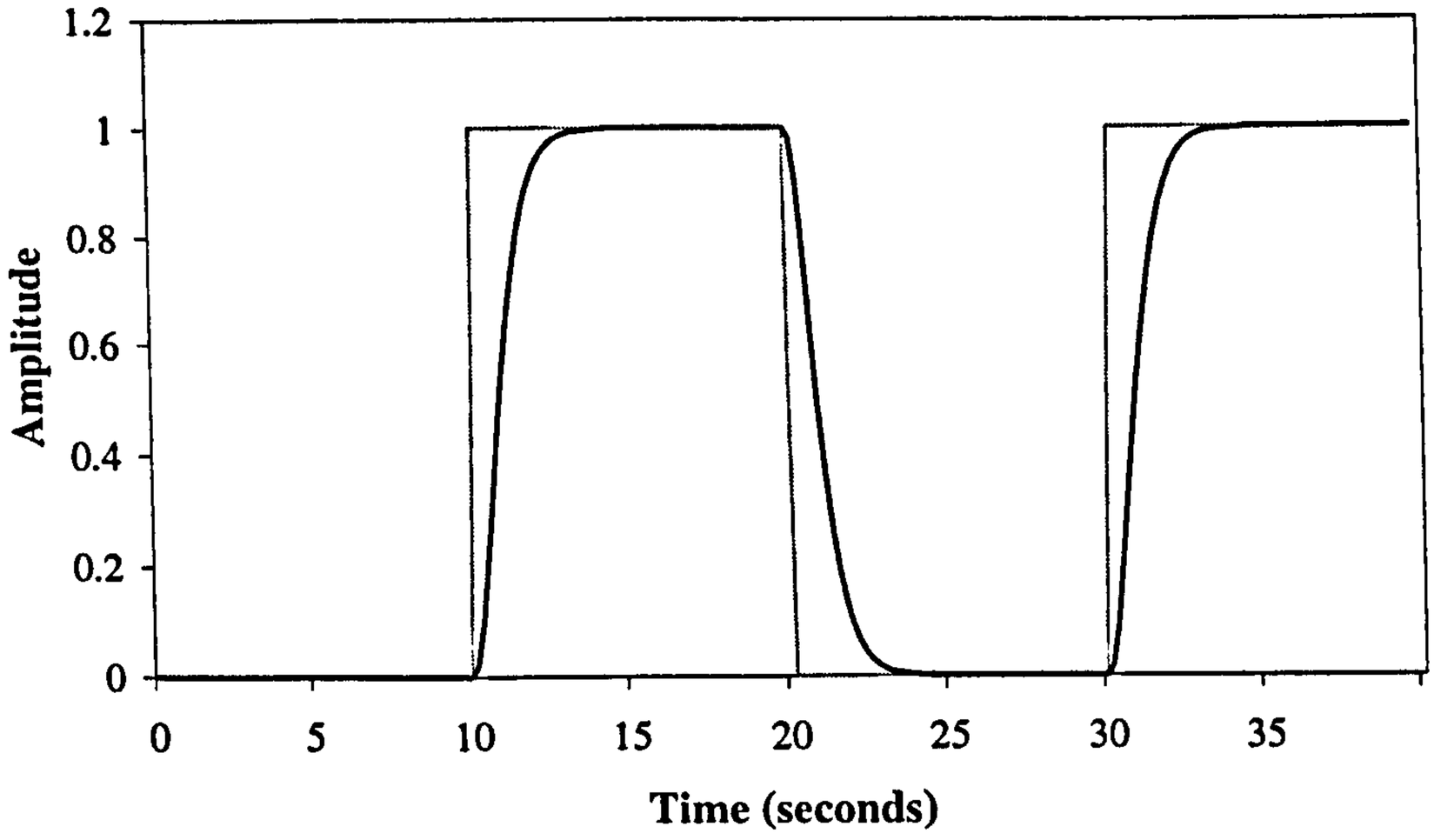

Figure 5.20: Step response of system plant using optimal K_P, K_I and K_D parameters of a PID controller selected using a temporal difference neural network.

5.6 Learning Control Implementation using Learning Automata

The learning automaton was next used to optimise the parameters of the popular PID controller. A team of learning architecture was used as shown in Figure 5.21 [28]. This study was performed to compare the performance of TD learning and the learning automata.

Using this principle each learning automaton seeks to optimise one parameter, thus for the PID optimisation, three learning automaton are required, for K_P, K_I and K_D. The objective in the design of the learning automata is to determine how the choice of action at any stage should be guided by past actions and responses. The important point is that the decisions are made with little knowledge of the environment, making the learning automaton very flexible to changes in the environment. The environment may have time vary-

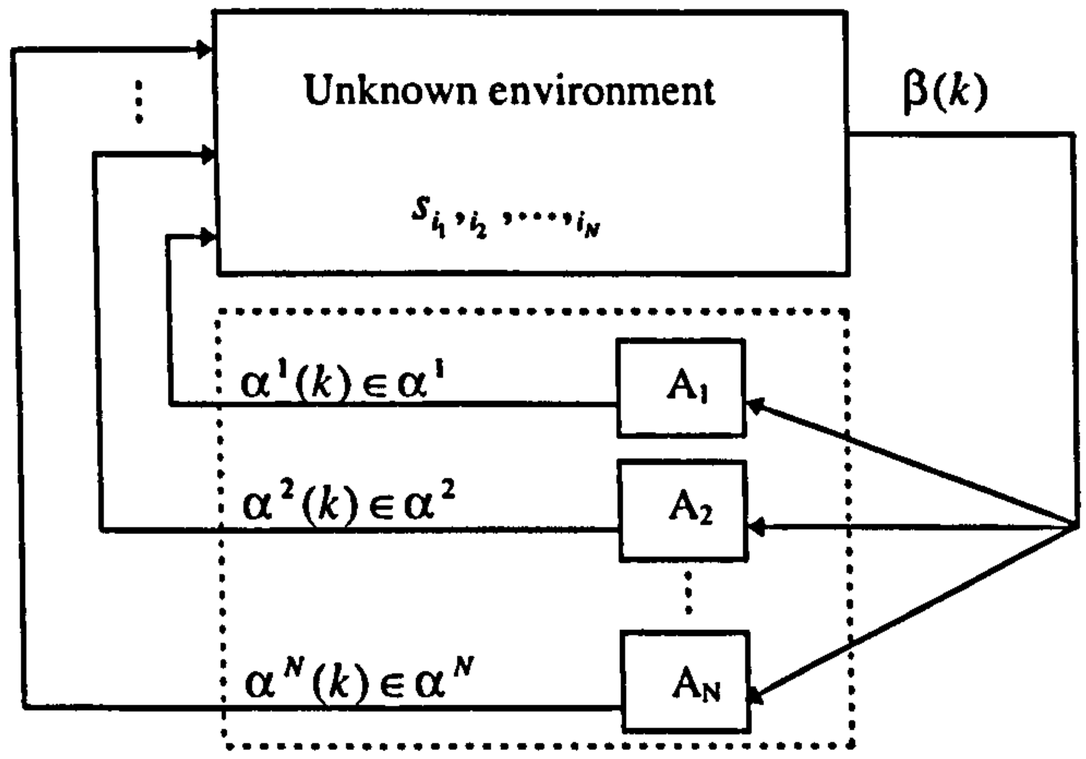

Figure 5.21: N learning automata operating in an unknown environment with identical payoff [28].

ing characteristics, or the decision maker may be part of a hierarchical decision structure but unaware of its precise role in the hierarchy. Alternatively, the uncertainty is due to the fact that the output of the environment is influenced by the actions of other agents unknown to any particular decision maker.

A flow chart Figure 5.22 below shows the general updating of the action parameters in the PID using a team of learning automata, see also Figure 5.23

5.7 Applying Learning Automata

The simulation study comprised three parts as shown in Figure 5.23, the PID controller, the plant and the team of learning automata. Note that the PID controller and plant make up the environment.

5.7.1 The Plant

The simulation study of the team of learning automata architecture used a third order system to represent the plant. The plant was identical to the

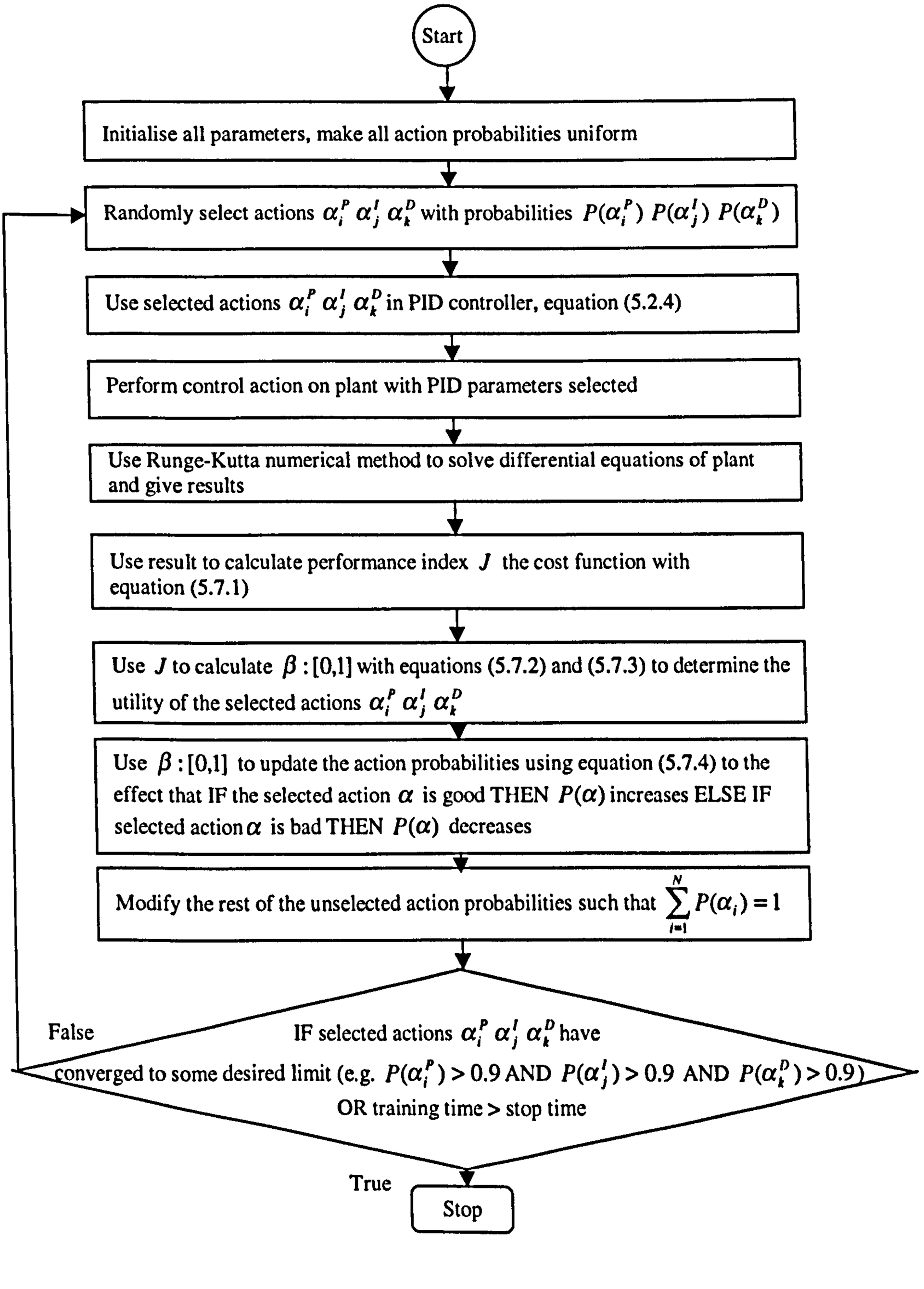

Figure 5.22: Flow chart for the optimisation of PID parameters using a team of learning automata.

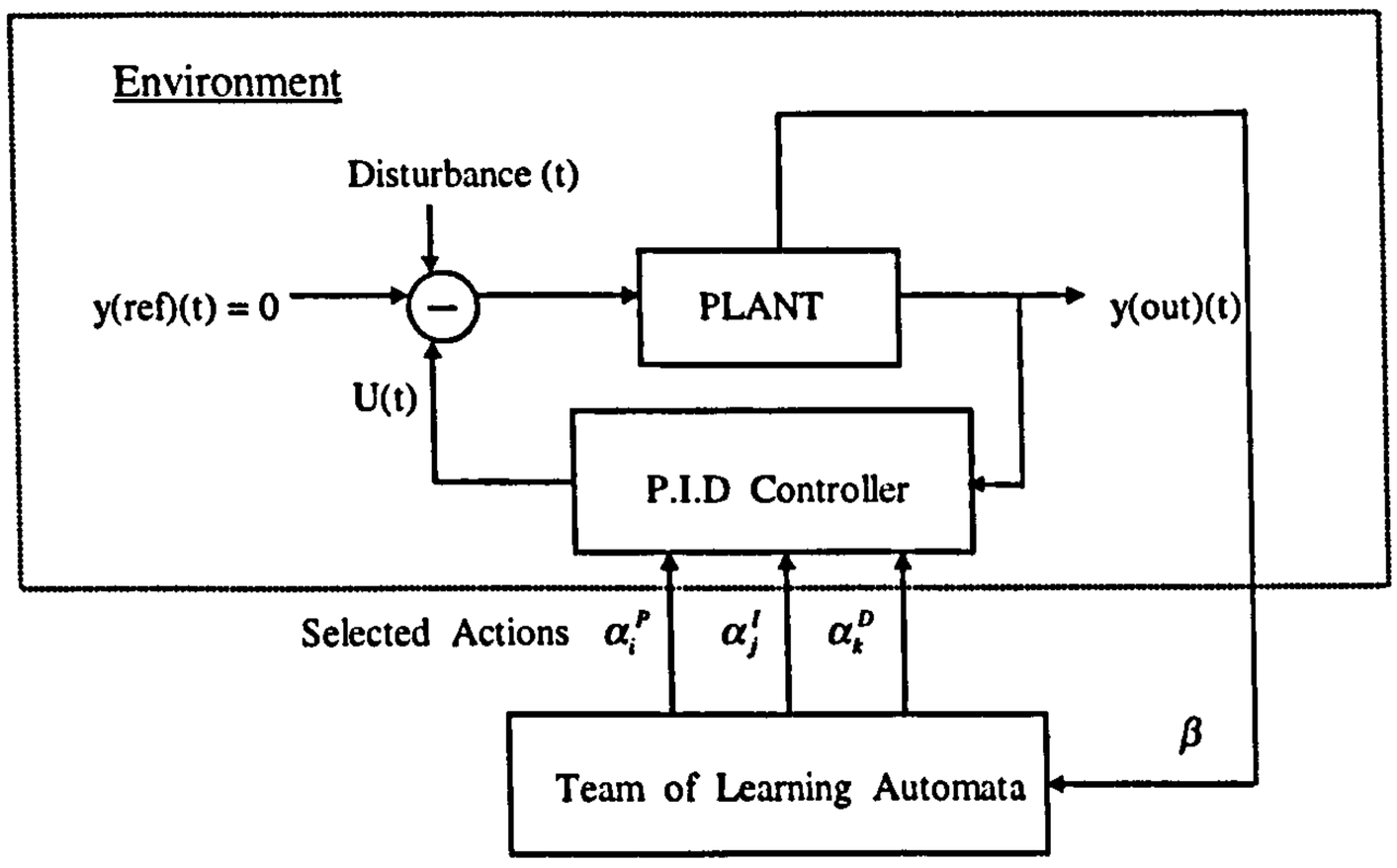

Figure 5.23: Optimisation of PID parameters using a team of learning automata.

one used in the study of the TD learning neural network. The third order system transfer function equation was given by equation (5.2.1) and in state space representation by equation (5.2.2). The output function can be seen in equation (5.2.3),

5.7.2 The PID Controller

The PID controller used in the simulation study was based on one used by Wu [28], it is the same as that used in the TD learning simulation and with the form shown in equation (5.2.4), see also Figure 5.23.

5.7.3 The Team of Learning Automata

The team of learning automata is a co-operative method for applying multi-agent learning. Each automata shares a part of the task using a common feedback reward signal β to achieve a global goal, in this case to optimise

an on-line PID controller. The actions of each automaton were divided into discrete values within a given range. The range was chosen after preliminary tests indicated where the best PID coefficients would be. It was interesting to compare if the learning automata selected PID coefficients similar to the manually selected ones in the initial tests.

All the action probabilities were uniformly initialised, so for ten actions the initial probability for selecting each action was a tenth. As each learning automaton gains experience and learns the action probabilities will be updated, in general if the action used was good then the probability of selecting that action is improved and vice versa. The probability sum must be conserved and thus as one probability was changed the others must also be changed in an inversely proportional manner. Initially the choice of actions is rather coarse but can be improved by fine tuning the PID coefficients. For example the K_P coefficients increment in steps of three (see Table 5.1), if the best coefficient fell between these limits how could we have found that out? One method is to centre a new search around the best coefficient, reset all the action probabilities and learn again. Therefore to discover such actions, it has to try actions that it has not selected before. As with the TD learning neural network each automaton has to *exploit* what it already knows in order to obtain reward, but it also has to *explore* in order to make better action selections in the future.

$$J = \frac{1}{N}\sum_{i=1}^{N}(y(ref) - y(out))^2 \tag{5.7.1}$$

here N is the RungeKutta numerical method sampling interval for calculating the outputs $y(out)$. This essentially gives an error measure between our actual $y(out)$ and reference $y(ref)$ signals by comparing the area between $y(out)$ and $y(ref)$, the error is reduced as $y(out)$ equals $y(ref)$, see Figure 5.24.

In order to use this cost measure in each learning automaton it must be normalised. This was achieved by using equation (5.7.2)

$$\beta(k) = \frac{J_{\max}(k) - J(k)}{J_{\max}(k) - J_{\min}(k)} \tag{5.7.2}$$

where

$$J_{\max}(k) = \begin{cases} J(k) & if \;\; J(k) > J_{\max}(k-1) \\ J_{\max}(k-1) & \text{otherwise} \end{cases} \tag{5.7.3}$$

$$J_{\min}(k) = \begin{cases} J(k) & if \;\; J(k) < J_{\min}(k-1) \\ J_{\min}(k-1) & \text{otherwise} \end{cases}$$

The feedback reward signal β was used to update the action probabilities for each automaton. The reinforcement scheme used was a linear reward-penalty $L_{(r-p)}$ scheme and is given by equation (5.7.5) [28],

$$P_n^\alpha(k+1) = P_n^\alpha(k) + \theta_1\beta(k)(1 - P_n^\alpha(k)) - \theta_2(1 - \beta(k))P_n^\alpha(k) \tag{5.7.4}$$

when $m = n$

$$P_m^\alpha(k+1) = P_m^\alpha(k) - \theta_1\beta(k)P_m^\alpha(k) + \theta_2(1 - \beta(k))\left[\frac{1}{(r-1)} - P_m^\alpha(k)\right]$$

when $m \neq n$

where P_n^α is the selected action probability, θ_1 and θ_2 are learning rates, P_m^α are the probability updates for the other un-selected actions, and r is the total number of actions available ($r = 10$ was used in this simulation).

It must be emphasised that during learning the action selection is random and the randomness must be uniform, otherwise there will be biases in the selection process which will not give the desired effect of exploring all actions with equal chance. The results of the probability updates for each PID parameter is shown in Figure 5.25. As each automaton learns the action probabilities are increased or decreased depending upon the utility of the action experienced.

After finding which action are the best to take the learning automata must use its knowledge to optimise the PID controller. The results shown in Figure 5.25 indicate that $p(kp[4])$, $p(ki[4])$ and $p(kd[3])$ are the best. Referring

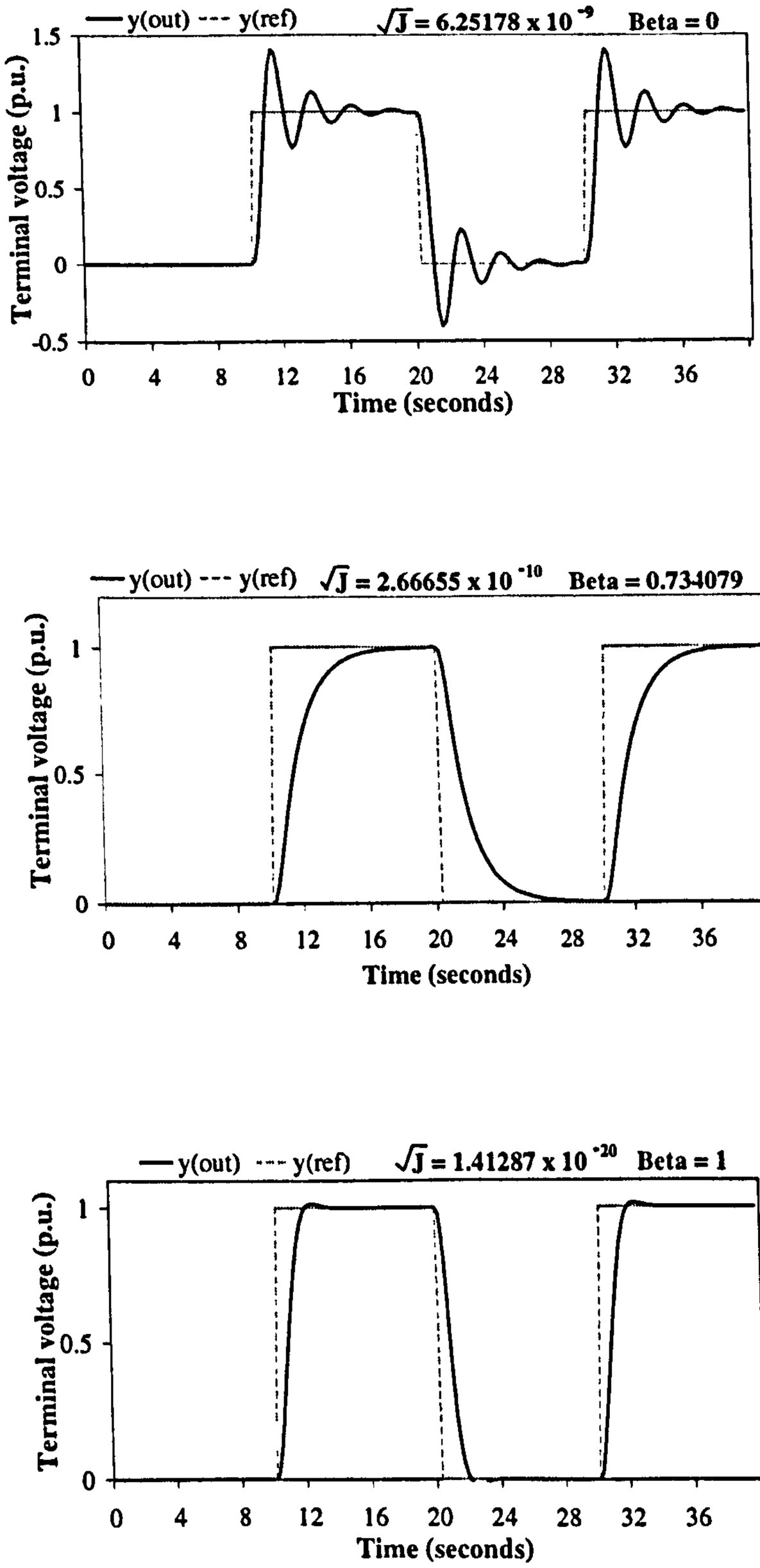

Figure 5.24: Dynamic response of plant, J is a cost measure used to calculate Beta the feedback reward signal. Beta = 1 indicates that the PID coefficients are good and the associated action probabilities are increased.

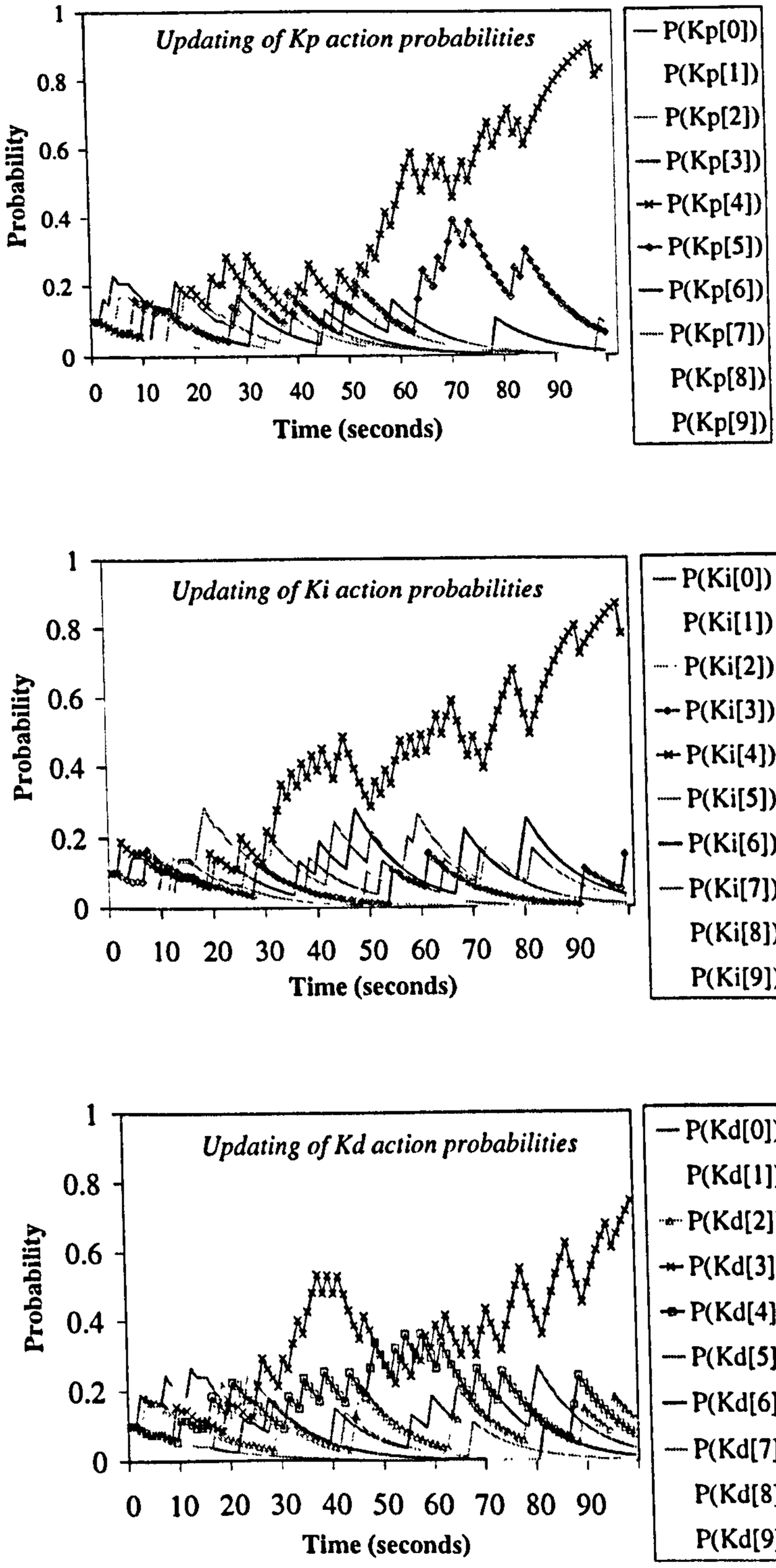

Figure 5.25: How the learning automata update their action probabilities with experience.

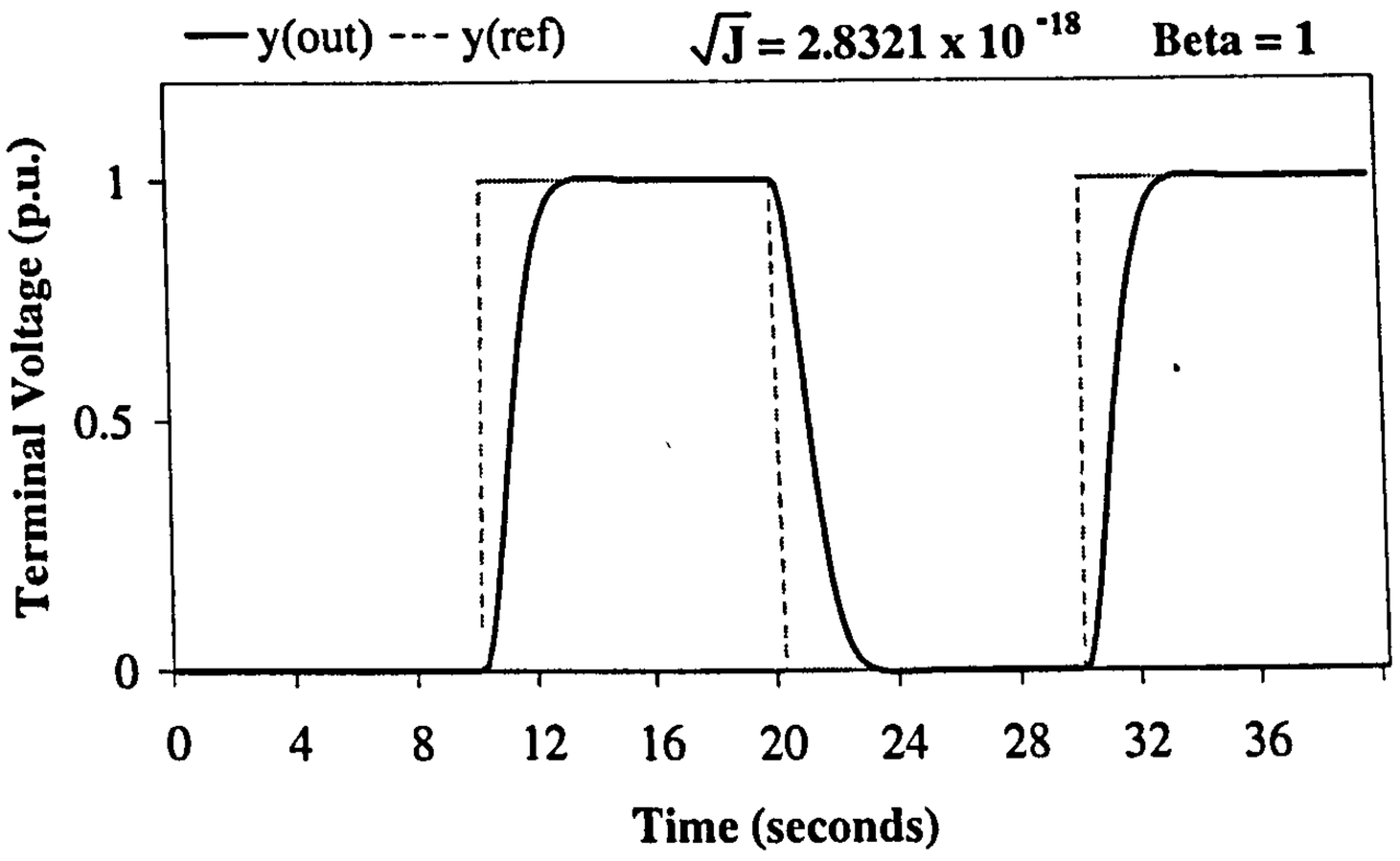

Figure 5.26: Dynamic response of plant for the optimised PID controller, optimised using a team of learning automata.

to Table 5.1 these action probabilities relate to the PID coefficients $Kp = 12$, $Ki = 12$ and $Kd = 3$. The results of these actions are shown in Figure 5.26.

It is interesting to observe that the team of learning automata optimised the PID control coefficients to give a desired output without any overshoot. Early testing of the PID controller showed that the output with the smallest cost function would give some overshoot but the rise time is fast resulting in a small error area. It was expected that the optimised PID controller would give such a dynamic response and return a result with the smallest cost value J. An example of this is shown in Figure 5.25, where a cost of $\sqrt{j} = 1.41287x10^{-20}$ is obtained yet the output suffers from some overshoot. However the optimised PID controller gives a cost of $\sqrt{J} = 2.8321x10^{-18}$ but doesn't suffer from overshoot. So although the team of learning automata isn't solving the problem of minimising the cost function directly, it can use J or really β the normalised cost function to guide it into choosing actions that can solve the problem. The situation arises though that there is a possibility for not finding a global optimal solution but a local optimal which can be a limitation to using multi-

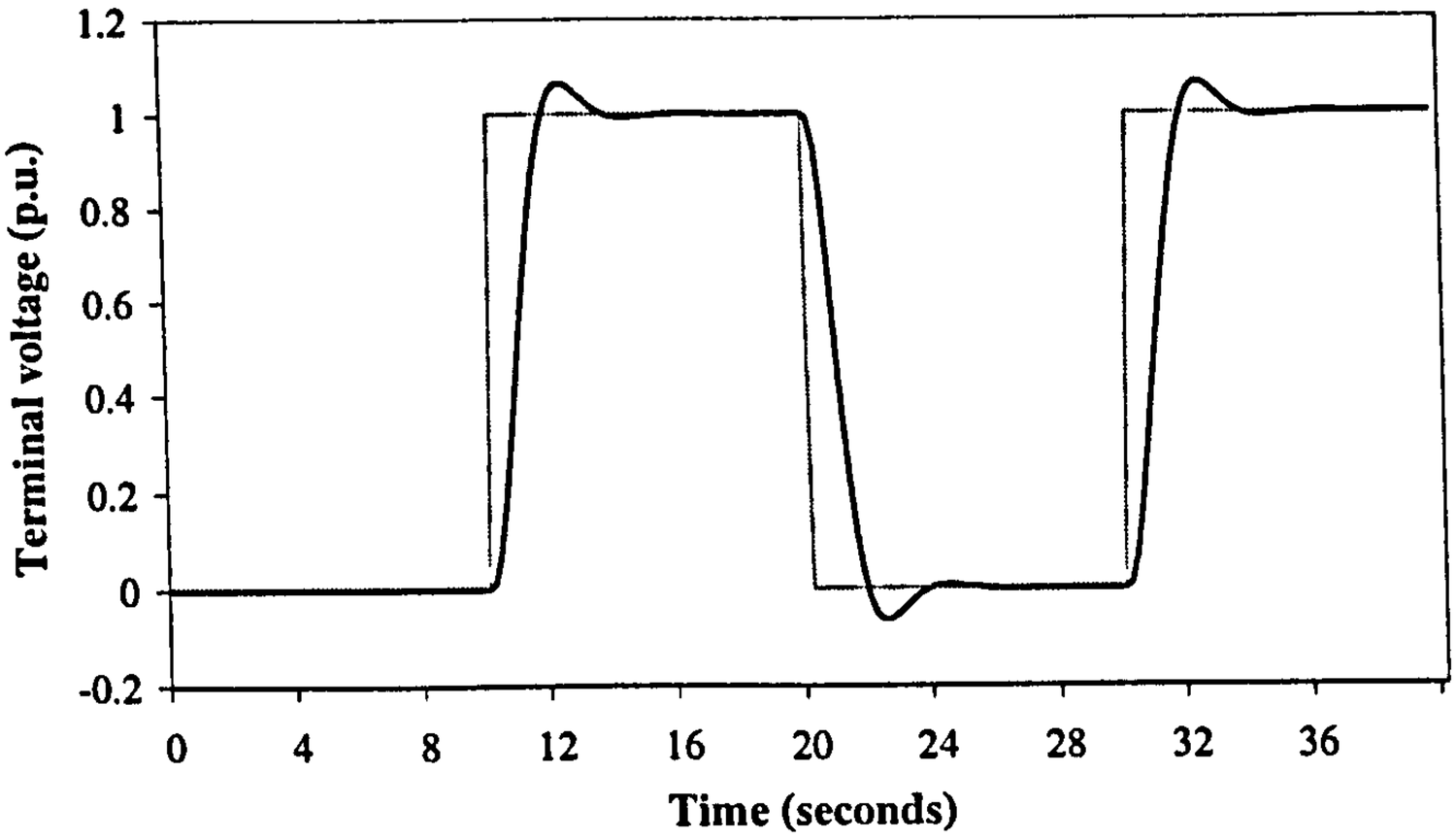

Figure 5.27: Control response of plant with PID controller using non-learning control but using conventional manual design.

agent learning.

Both reinforcement learning control methods, TD learning and learning automata provide an automatic method for learning control parameters in a progressive and adaptive manner. Conventional hand designed controllers and those whose parameters are tuned manually require some knowledge of the system to be controlled, which isn't always possible. Using the 3*rd* order system (equations (5.2.1)) as a comparative test between learning control and conventional non-learning manual design control methods, using the root locus method to find the coefficients of the parameters in a third order system. The PID controller parameters evaluated by manual tuning provides a response shown in Figure 5.27. When compared, the learning control provides a more than adequate automatic method of tuning control parameters for use in dynamic systems, but more importantly it has the ability to respond to unforeseen changes in a dynamic system, such as faults and changes in system performance by re-learning a new control response.

Chapter 6

Learning for Power System Control

As a practical and important application to a large scale optimisation problem power systems were identified as providing a range of opportunities for using reinforcement learning. Therefore the problem of power system control and optimisation was investigated. It was also understood that using temporal difference learning on power system control and optimisation had never been studied before. Following the previous successful work using interconnected learning automata for optimising the parameters of a PID controller, further study of using a neural network based reinforcement learning schemes such as temporal difference reinforcement learning was investigated. As demonstrated in the previous chapter the ability of learning control is a flexible and powerful method of optimising a parameterised controller.

6.1 Power System Control Problems

Power systems provide a consumer with electrical energy, the quality of this energy needs to be maintained, primary factors that ensure this quality are,

- Constant frequency
- Constant voltage
- Reliability
- Purity of sinusoidal waveform

The frequency and voltage of a system is dependent upon the active power balance in other words when the power output is equal to the power demand plus system (e.g. transmission) losses. In reality a power system is never in such a state of equilibrium because the power demand changes continuously as consumers switch on and switch off appliances. Since the consumers behaviour cannot be predicted the need for adaptive control is essential in power systems.

Reliability depends on the ability of a power system to survive sudden faults, overloads and loss of generators, transformers and transmission lines. If the damage is severe enough then some consumers will experience power interruption. Of course in some cases even a very short interruption is fatal, such as in a hospital operating theatre.

The purity of a sinusoidal is only recently becoming important. Any non-linear load absorbs a non-sinusoidal current, the harmonics of a non-sinusoidal current cause harmonic voltage drops and distortions in the power [61]. Traditionally rectifiers and fluorescent tubes have been the causes of these voltage distortions. However the present use of power electronics in appliances, such as transistors and thyristors in television rectification, light dimming switches washing machine controllers etc... can only mean that this problem will increase.

Some of the types of unwanted disturbances that occur in power transmission are shown in Figure 6.1. Power system reliability in terms of short term interruptions such as *Outage* and *Sag* have been primary reasons for a concept known as *Custom Power* [62]. Custom Power is a response to poor

power quality present in factories homes and offices, the aim is to provide a better quality of supply, since in times of greater electronic (especially computer) expansion, poor power quality cannot be tolerated. The instabilities in the power supply can result in computer data loss, which can be expensive to correct. *Harmonics*, *Impulses* and *Swells* interfere with electronic circuits and stress the electrical insulation of end use equipment reducing the lifetime of electrical equipment. The use of reinforcement learning to optimise/control FACTS, *Flexible AC Transmission Systems* (or other power system) devices can be used to realise the concept of custom power [63] [64] [65].

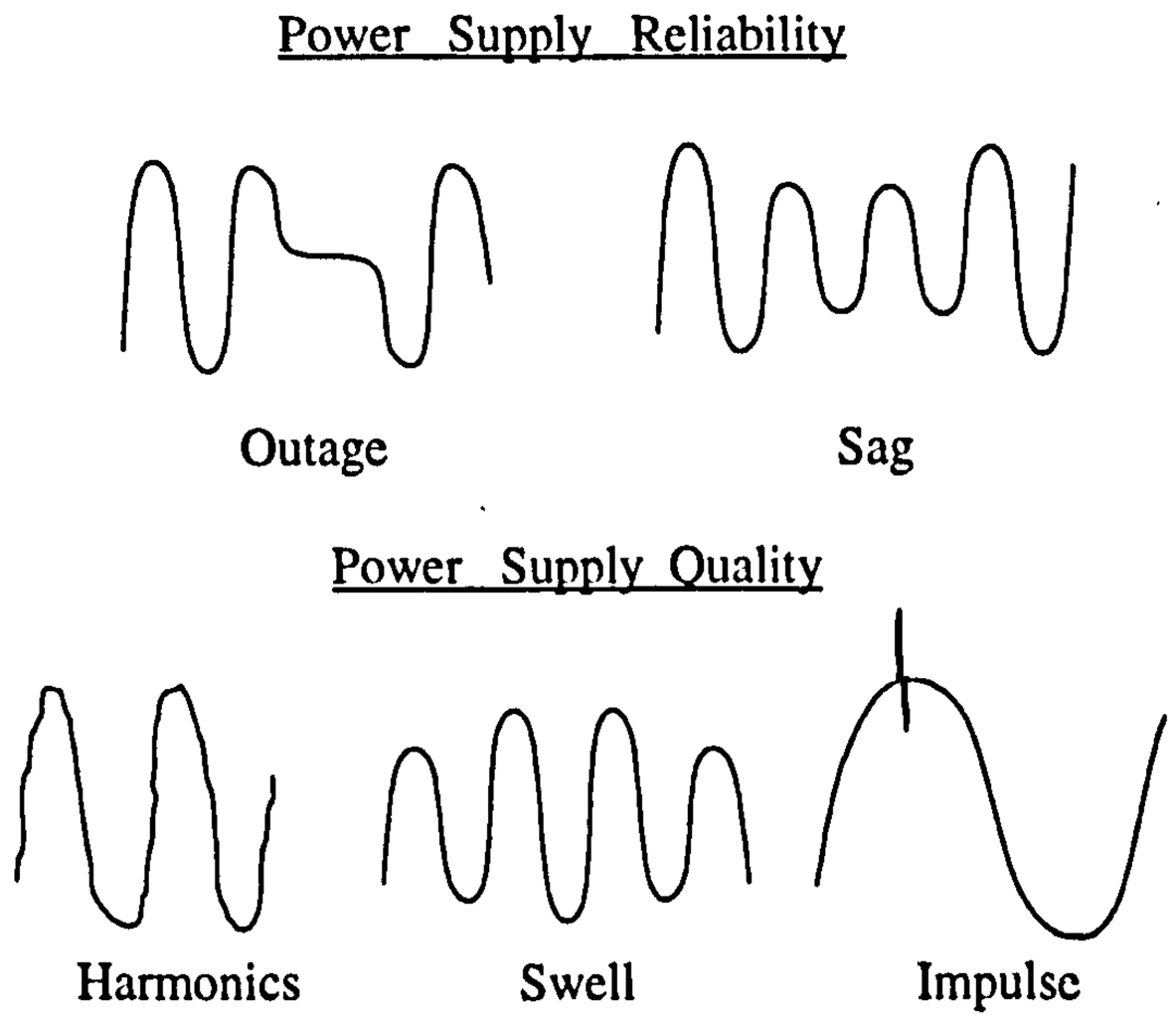

Figure 6.1: Disturbances in power supply systems [62].

The learning automata was used to initially test the power system control problem on a single machine infinite busbar system in a simulation study. Although this work had already been performed by Wu [28] it provided a foundation upon which further research was developed.

Form the initial single machine power system with infinite busbar a more complex power system was developed. This power system had three generators in the simulated power system network with two loads and using TD learning to update a parameterised fuzzy logic controller instead of a PID controller.

6.2 Power System Models

For initial tests a single-machine-infinite-busbar (SMIB) power system has been used to represent the reinforcement learning environment, Figure 6.2 shows the general SMIB.

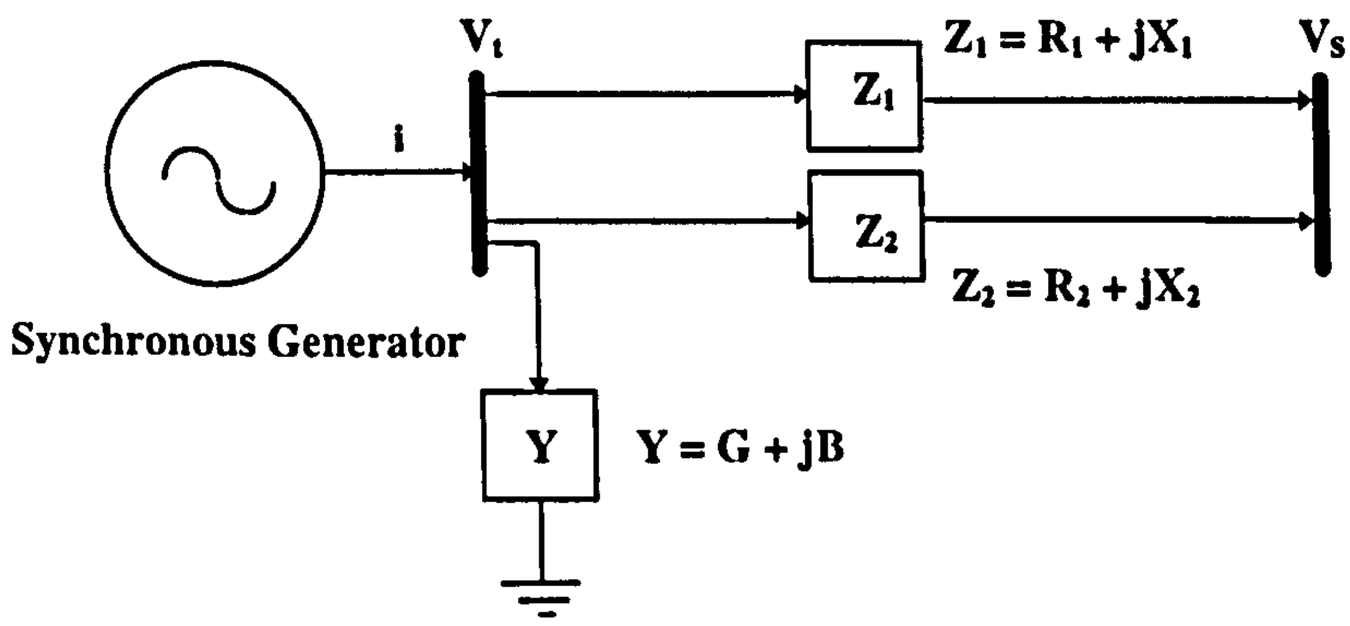

Figure 6.2: A single-machine-infinite-busbar power system [61].

Where i is the current, V_t is the terminal voltage Z_1 and Z_2 are the impedances of transmission lines 1 and 2 respectively. Y is the self admittance, G is the conductance and B is the susceptance. R_1 and R_2 are the resistances, X_1 and X_2 are the reactances in transmission lines 1 and 2 respectively. V_s is the voltage of a single-machine-infinite-busbar power system.

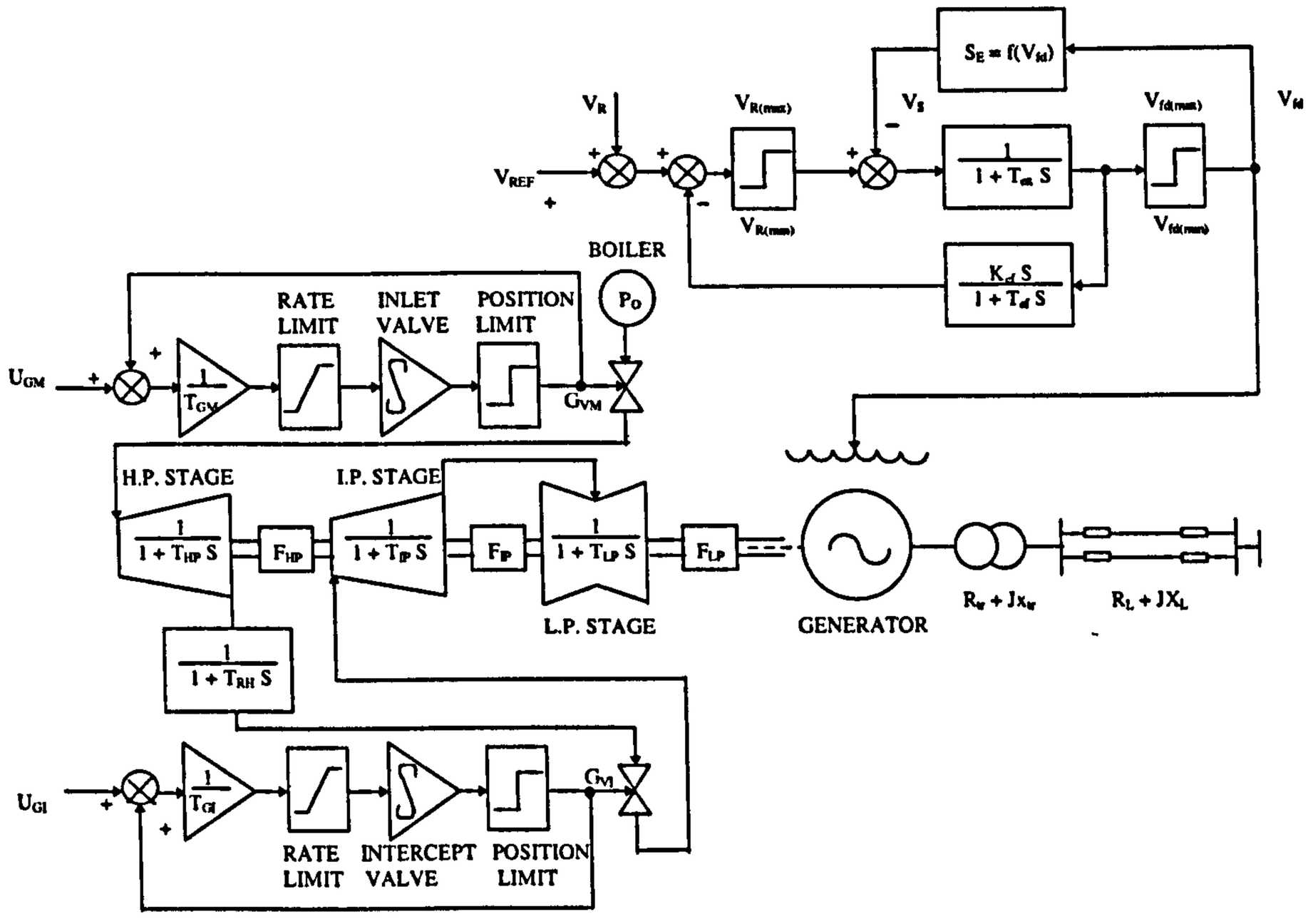

Figure 6.3: Turbo-generator system to be controlled.

6.3 Non-linear Model of Turbo-Generator

The development of a turbo generator model for simulation studies was a fundamental requirement. The basis for producing the turbo-generator model was provided by Wu [28], the model state equations for the various elements and their initial conditions are shown in the relevant subsections and are described later. A system block diagram of the turbo generator is shown in Figure 6.3.

The model is as realistic as possible and was used to test the optimisation of PID control parameters using a team of learning automata architecture.

The following is a list of symbols used in the model of the single-machine-infinite-busbar-system.

List of symbols

δ rotor angle with respect to infinite bus

ω rotor speed

I_{fd} field current

I_d, I_q stator currents in direct and quadrature-axis circuits, respectively

I_{kd}, I_{kq} damper circuit currents in $d-$ and $q-$axes, respectively

V_t generator terminal voltage

P_t, Q_t power and reactive power delivered at terminal, respectively

ψ flux linkages

V_R exciter voltage

X_{tr}, X_L transformer and transmission line reactances, respectively

R_{tr}, R_L transformer and transmission line resistances, respectively

R_a stator resistance

X_d, X_q synchronous reactances in $d-$ and $q-$axes, respectively

X_{ad}, X_{aq} $d-$ and $q-$axis mutual reactances, respectively

T_{ex} exciter time constant

T_{ef}, K_{ef} regulator stabilising circuit time constant and gain, respectively

T_{ac}, T_a compensation coefficients of automatic voltage regulator

$V_{R(\max)}, V_{R(\min)}$ maximum and minimum limitations of excitation voltage, respectively

V_c input to excitation system

T_e airgap torque

T_m generator shaft torque

H inertia constant

K_d mechanical damping coefficient

U_{GM}, U_{GI} actuating signal to governor on inlet and intercept valves, respectively

G_{VM}, G_{VI} position of inlet and intercept valves, respectively

P_0 internal boiler steam pressure

F_{HP}, F_{IP}, F_{LP} power fraction from HP, IP and LP stages of turbine

T_{HP}, T_{IP}, T_{LP} time constants associated with HP, IP and LP stages of turbine

T_{GVM}, T_{GVI} time constants of inlet and intercept valves, respectively

T_{RH} turbine reheat time constant

Δ deviation from steady-state value

6.3.1 Turbine and Boiler

The boiler is represented by an internal steam pressure Po and has been assumed to be a constant steam source. The steam raised by the boiler is used to drive a three stage turbine $H.P$ high pressure, $I.P$ intermediate pressure and $L.P$ low pressure respectively. The steam passes through each stage from high to low pressure turbines which drive a synchronous generator see Figure 6.3. The power fractions available from each stage are calculated from the following equations (6.3.1), (6.3.2) and (6.3.3). These were calculated continuously for the system dynamic response using the RungeKutta numerical method to observe the continuous output. Equation (6.3.3) was calculated every $20ms$ to give a feedback to the learning system for performance evaluation.

$$
\begin{aligned}
\dot{\delta} &= \Delta\omega \qquad (6.3.1)\\
\Delta\dot{\omega} &= \omega_0(T_m - T_e - K_d\,\Delta\,\omega)/2H\\
\dot{\psi}_{fd} &= \omega_0(V_{fd} - R_{fd}I_{fd})\\
\dot{\psi}_d &= \omega_0(V_d + \psi_q + I_d(R_a + R_e)) + \psi_q\,\Delta\,\omega\\
\dot{\psi}_{kd} &= -\omega_0 R_{kd} I_{kd}\\
\dot{\psi}_q &= \omega_0(V_q - \psi_d + I_q(R_a + R_e)) - \psi_d\,\Delta\,\omega\\
\dot{\psi}_{kq} &= -\omega_0 R_{kq} I_{kq}\\
T_e &= \psi_d I_q - \psi_q I_d
\end{aligned}
$$

$$\begin{bmatrix} \psi_{fd} \\ \psi_d \\ \psi_{kd} \\ \psi_q \\ \psi_{kq} \end{bmatrix} = \begin{bmatrix} X_{fd} & -X_{ad} & X_{ad} & & \\ X_{ad} & -(X_d+X_e) & X_{ad} & & \\ X_{ad} & -X_{ad} & X_{kd} & & \\ & & & -(X_q+X_e) & X_{aq} \\ & & & -X_{aq} & X_{kq} \end{bmatrix} \begin{bmatrix} I_{fd} \\ I_d \\ I_{kd} \\ I_q \\ I_{kq} \end{bmatrix} \quad (6.3.2)$$

$$\begin{aligned} V_{td} &= V_b \sin\delta + R_e I_d - (X_{tr}+X_L) I_q \quad (6.3.3) \\ V_{tq} &= V_b \cos\delta + R_e I_q - (X_{tr}+X_L) I_d \\ V_t &= (V_{td}^2 + V_{tq}^2)^{1/2} \\ P_t &= V_{td} I_d + V_{tq} I_q \\ Q_t &= V_{tq} I_d - V_{td} I_q \end{aligned}$$

where

$$R_e = R_{tr} + R_L, \quad X_e = X_{tr} + X_L$$

the parameters used in the simulation for the synchronous turbo-generator are given in equation (6.3.4).

$$\begin{aligned} H = 3.25, \quad & X_{ad} = 1.86 p.u., \quad X_q = 1.91 p.u. \\ R_{fd} = 0.0015 p.u., \quad & X_{aq} = 1.77 p.u., \quad X_{kd} = 1.94 p.u. \\ R_{kd} = 0.0078 p.u., \quad & X_{fd} = 1.97 p.u., \quad X_{kq} = 1.96 p.u. \quad (6.3.4) \\ R_{kq} = 0.0084 p.u., \quad & R_a = 0.005 p.u., \quad X_d = 2.0 p.u. \\ K_d = 0.025 & \end{aligned}$$

6.3.2 The Governor System

The governor controls (or regulates) the amount of steam pressure available from the boiler and previous turbine stages. In this system two governor control signals are present, the first U_{GM} controls the inlet from the boiler regulating steam pressure to the $H.P$ turbine stage, while the second U_{GI} controls the

intercept valves regulating steam pressure to the $I.P$ and $L.P$ turbine stages. The regulation of the steam pressure at a constant level controls the speed of the turbines, allowing the synchronous machine to generate electric power at a constant frequency.

$$\begin{aligned}
\dot{Y}_{HP} &= (G_{VM}P0 - Y_{HP})/T_{HP} \\
\dot{Y}_{RH} &= (Y_{HP} - Y_{RH})/T_{RH} \\
\dot{Y}_{IP} &= (G_{VI}V_{RH} - Y_{IP})/T_{IP} \\
\dot{Y}_{LP} &= (P_{IP} - Y_{LP})/T_{LP} \\
\dot{G}_{VM} &= (U_{GM} - G_{GM})/T_{GVM} \\
\dot{G}_{VI} &= (U_{GI} - G_{VI})/T_{GVI} \\
T_m &= F_{HP}Y_{HP} + F_{IP}Y_{IP} + F_{LP}Y_{LP}
\end{aligned} \tag{6.3.5}$$

where

$$0 < G_{VM} < G_{VM(\max)}, \quad 0 < G_{VI} < G_{VI(\max)}$$

$$\begin{aligned}
G_{VM(\min)} &\leq G_{VM} \leq G_{VM(\max)} \\
\dot{G}_{VI(\min)} &\leq \dot{G}_{VI} \leq \dot{G}_{VI(\max)}
\end{aligned} \tag{6.3.6}$$

the parameters used in the simulation study for the governor system are shown in equation (6.3.7).

$$\begin{aligned}
F_{HP} = 0.24, \quad T_{HP} = 0.3s, \quad T_{GVM} = 0.1s \\
F_{IP} = 0.34, \quad T_{IP} = 0.3s, \quad T_{GVI} = 0.01s \\
F_{LP} = 0.42, \quad T_{LP} = 0.72s, \quad T_{RG} = 10.0s \\
P_0 = 1.2
\end{aligned} \tag{6.3.7}$$

6.3.3 Excitation System

The excitation system provides a voltage V_{fd} to the generator which is used to control the magnetic field in the rotating machine generating a three phase

alternating-current output. The three phase output is mathematically represented by its transformed direct and quadrature components. The excitation voltage is used to calculated the magnetic flux in the generator, these flux calculations are further used to calculate the currents in the machine. Finally the terminal voltage can be determined after the generator output is changed by the transformer, for transmission through power lines to the consumer.

$$\begin{aligned} \dot{V}_R &= (K_a(V_c + T_{ac}\dot{V}_c) - V_R)/T_a \\ \dot{V}_{fd} &= (V_R - V_{ef} - V_{fd})/T_{ex} \\ \dot{V}_{ef} &= (K_{ef}\dot{V}_{fd} - V_{ef})/T_{ef} \end{aligned} \tag{6.3.8}$$

where

$$\begin{aligned} V_{R(\min)} &\leq V_R - V_{ef} \leq V_{R(\max)} \\ V_{fd(\min)} &\leq V_{fd} \leq V_{fd(\max)} \end{aligned}$$

the parameters for the excitation system used in the simulation study are shown in equation (6.3.9).

$$\begin{aligned} T_{ex} = 0.01s, \quad T_{ac} = 0.1514, \quad T_a = 0.0154 \\ T_{ef} = 0.3s, \quad K_{ef} = 0.15, \quad K_a = 0.05 \end{aligned} \tag{6.3.9}$$

where

$$-0.005 \leq V_{fd} \leq 0.005$$

6.3.4 Transmission System

The synchronous generator represented in this model has its output power controlled by the excitation system. The power output generated at the terminals must give a terminal voltage (V_t) equal to any losses in transmission line(s) plus the constant load voltage required by the consumer (Vs=1.0 p.u.).

The transmission line(s) characteristics need to be taken into account for losses in order for a correct value of terminal voltage (V_t) to be calculated.
the following are the parameters used to model the transformer (equation (6.3.10)) and transmission lines (equation (6.3.11)).

$$R_{tr} = 0.038 p.u., \quad X_{tr} = 0.1 p.u. \tag{6.3.10}$$

$$R_L = 0.025 p.u., \quad X_L = 0.35 p.u. \tag{6.3.11}$$

6.4 Applying Reinforcement Learning

The popular PID controller is used in a power system. The parameterised nature of the PID controller makes it a suitable candidate for the application of reinforcement learning. The methodical and automatic nature of improving the parameter action selection through continuous learning is shown in the following example using reinforcement learning of a PID controller to optimise the performance of a turbo-generator.

6.4.1 Parameter Optimisation of a Turbo-Generator PID Controller

A team of interconnected learning automata have been used to successfully optimise a PID controller in a turbo-generator system [28]. The team of interconnected learning automata (Figure 5.21)are able to learn optimal PID control parameters of an unknown turbo-generator system in a noisy environment without persistent excitation signals. Each automata in the team controls/optimises one parameter of a complex dynamic system, in this case the PID controller of a turbo-generator system. At every instant the internal states of each learning automata are updated according to some probability

distribution and an action (of a finite set) is taken. Each automaton has a uniform probability distribution initially, with all actions having an equal probability set by $p_n^j(k) = 1/r_j (n = 1, 2, ...r_j)$ and updated according to input β The response β from the environment is used in the reinforcement scheme (or learning algorithm) to update the state probabilities by giving identical pay-offs to every automaton in the team. The reinforcement scheme is described in equation (6.4.1) [28] and a flow chart, Figure 6.4 illustrates how the learning was implemented.

$$\begin{aligned} p_n^j(k+1) &= p_n^j(k) + \theta_1^j \beta(k)(1 - p_n^j(k)) - \theta_2^j(1 - \beta(k))p_n^j(k) \qquad (6.4.1) \\ &\text{when } m = n \\ p_m^j(k+1) &= p_n^j(k) - \theta_1^j \beta(k) p_m^j(k) + \theta_2^j(1 - \beta(k)) \left[\frac{1}{(r_j - 1)} - p_m^j(k)\right] \\ &\text{when } m \neq n \end{aligned}$$

Where $n \in 1, 2, ..., r_j$ and $m = 1, 2, ..., r_j$, $\beta(k) \in [0, 1]$, with $0 < \theta_1^j < 1$ and $0 < \theta_2^j < 1$ being the reward and penalty parameters, respectively. The inputs to the environment $\alpha = \{\alpha^1, \alpha^2, ..., \alpha^N\}$ represents a finite input set, where α^j is a subset of the input provided by the jth automaton Aj, $(j = 1, 2, ..., N)$. A reward probability of the environment corresponding to action $\alpha(k)$ defined by equation (6.4.2) [28]

$$s_{i_1, i_2, ..., i_N} = E\beta(k) \mid \alpha(k) = [\alpha_{i_1}^1, \alpha_{i_2}^2, ..., \alpha_{i_N}^N] \qquad (6.4.2)$$

The expected payoff at instant k is given by equation (6.4.3)

$$E(k) = \sum_{i_1, i_2, ..., i_N} p_{i_1}^1(k) p_{i_2}^2(k) ... p_{i_N}^N(k) s_{i_1, i_2, ..., i_N} \qquad (6.4.3)$$

Where $p_{i_j}^j(k)$ is the probability that A_j chooses action $\alpha_{i_j}^j$ at instant k. For systems with a wide range of parameter values such as the turbo-generator system, search with subsets of actions can be used [28]. The basic method

involves searching a subset of actions until some performance criterion is met and then expand or contract the subset according to the action probabilities, which are reset (back to initial uniform distribution) from time to time. This keeps the number of working values low for rapid search, as well as allowing for a global search to find the optimum parameter value.

The advantage of a reinforcement learning method such as the learning automata is that learning can be carried out directly (on-line), based on the control actions and system performance, thus direct knowledge of the system model is never required. For real turbo-generators, obtaining an "exact" system model can be difficult and expensive if not impossible in some cases.

A simulation study of a non-linear turbo-generator was used to test the reinforcement learning approach, although similar to the approach used in the previous there are some differences. The learning was carried out in a stochastic environment where only noise is present and without the use of a persistent excitation signal (such as the PRBS used previously. The PID control parameters were optimised by a team of interconnected learning automata searching for the optimum control actions. Each learning cycle is one second of real time. The parameters to be optimised were K_P, the proportional coefficient, K_D, the differential coefficient and γ, a stabilizing signal coefficient, the integration coefficient, K_I was kept constant since it was not sensitive to the control performance. The parameter ranges were set as $K_P \in [3,0]$, $K_D \in [0.2,0]$ and $\gamma \in [0,-0.2]$, a sampling interval of $\tau = 20ms$ was used and the control performance was evaluated during a period of $2s$.

Figure 6.5 shows an example of the learning, where the probability of the optimum control action approaches one and the other action probabilities vanishing to zero as learning progresses. The interesting fact that the optimum control parameters were obtainable in a noisy environment without the use of a persistent excitation signal and the ease of implementing the team of interconnected learning automata to any real unknown industrial turbo-generator when

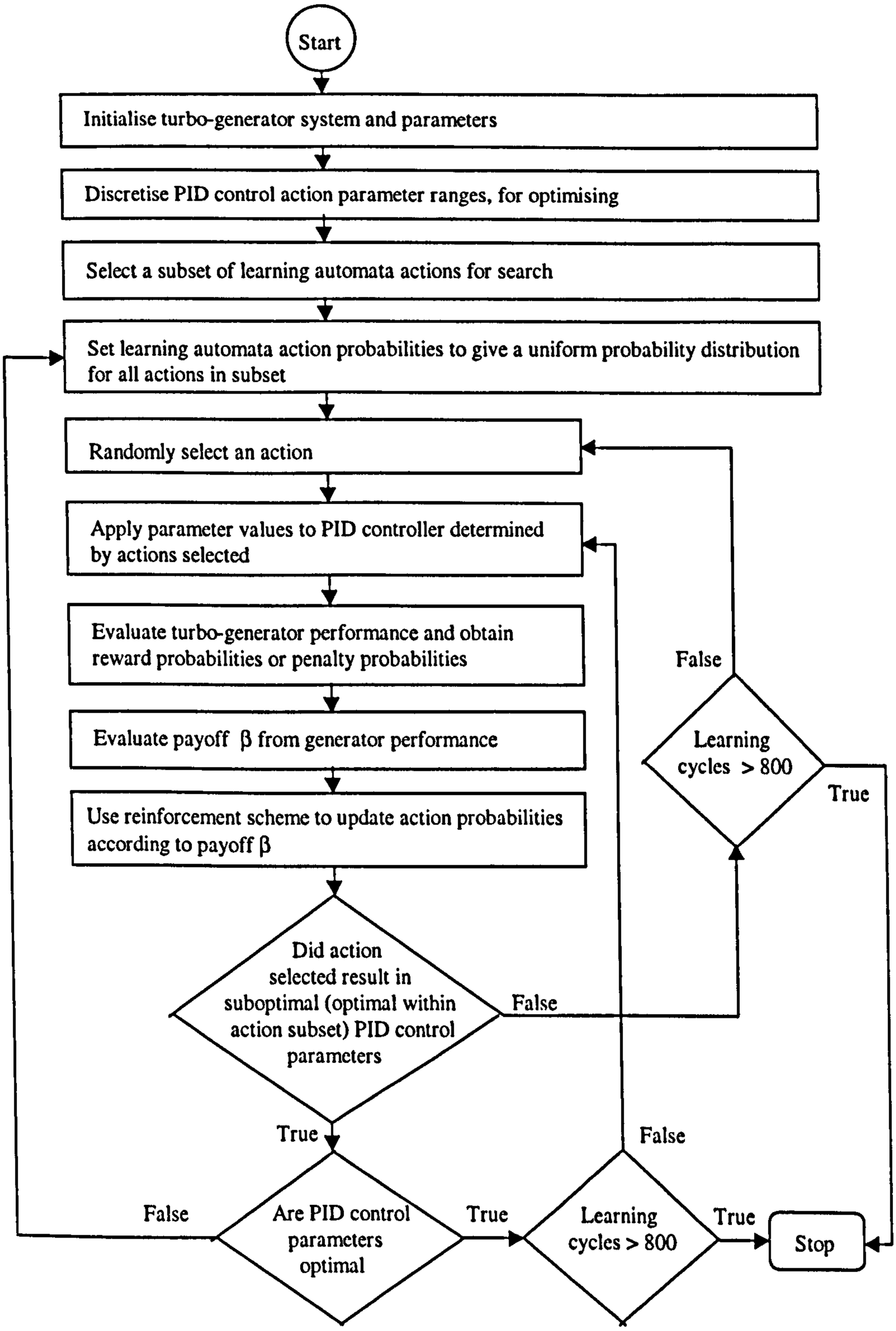

Figure 6.4: Flow chart of turbo generator system PID controller optimisation using a team of interconnected learning automata.

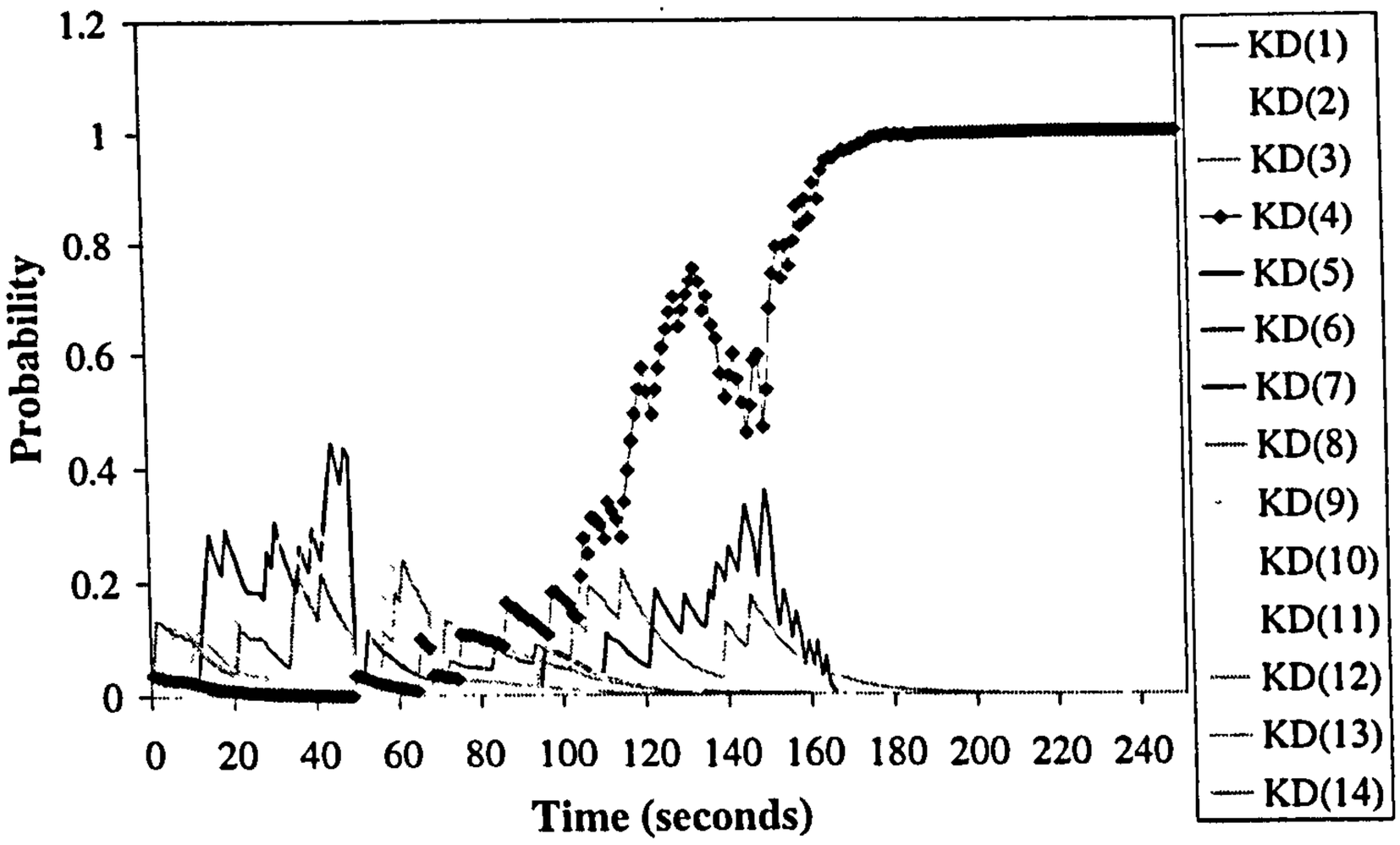

Figure 6.5: Optimisation of K_D parameter in a turbo generator PID controller using a team of interconnected learning automata.

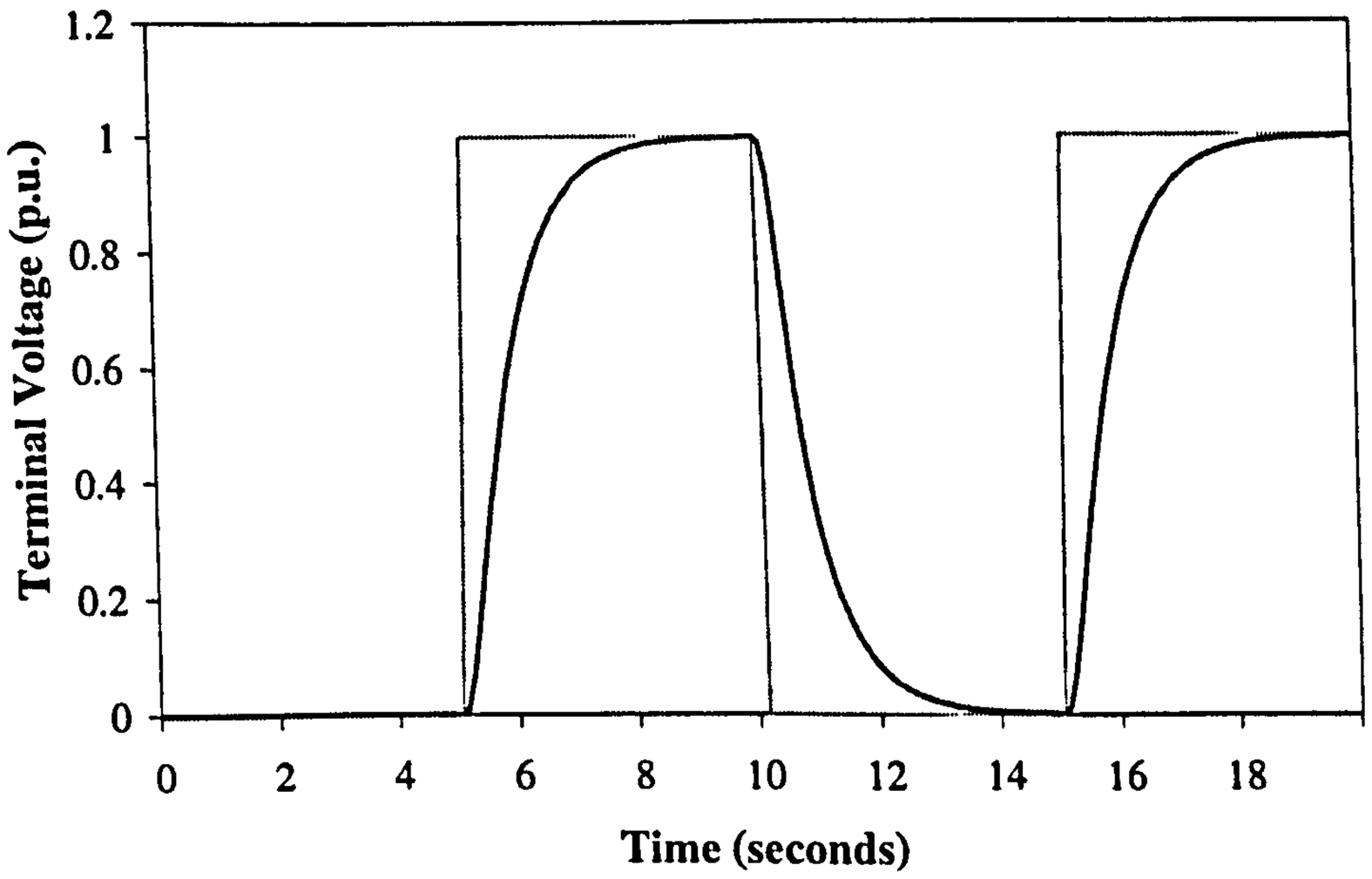

Figure 6.6: Control performance of team of interconnected learning automata in a power system optimisation problem.

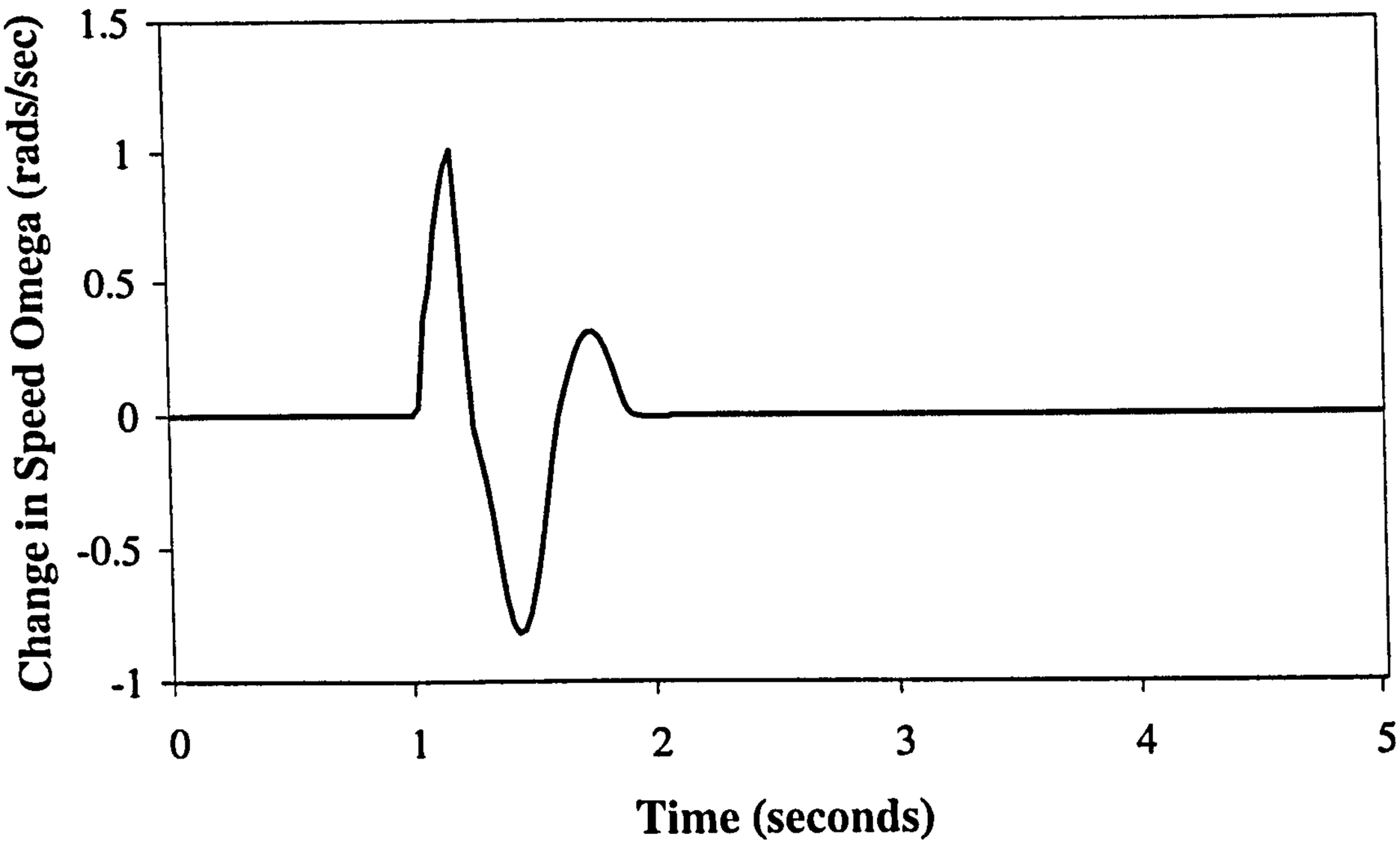

Figure 6.7: Stability response of the team of interconnected learning automata in a power system after recovering from a three phase short circuit.

required. This example illustrates the great potential of using reinforcement learning methods to complex engineering problems.

The final control performance of the system after training can be seen in Figures 6.6 and 6.7.

6.4.2 Learning Time

In most reinforcement learning methods there must be a balance between gaining new knowledge and using learnt knowledge for effective control performance. The fundamental limitation is the learning time required to make use of any information the environment can provide. In practice the time for simple system such as this single machine infinite busbar system the learning time is not a major concern when using modern computer processors which

typically have high clock speeds of hundreds of megahertz. For more complex systems the learning and computational time will grow accordingly. One obvious way in which to reduce these increased learning times is to use parallel learning processes. For the PID controller it is a simple matter to have a separate learning automaton optimising each PID parameter. Taking this idea one step further for multi-machine power systems for example then learning control need not be centralised but can be distributed as demonstrated in the next chapter using a multi-agent reinforcement learning scheme.

Chapter 7

Multi-Agent Learning for Control of Multi-Machine Power Systems

Being able to predict and control stochastic systems has always fascinated scientist and researchers. A lot of real world systems are stochastic such as the financial stock market, weather forecasting and information routing. One other important aspect of natural intelligent systems is their distributed nature and sometimes described as connectionist systems [66], [5]. The move from large complex centralised intelligence to simple distributed but connected intelligence was influenced by biological models of the brain, which are powerful, flexible and highly adaptive, but also not yet completely understood.

Early work with temporal difference learning for practical problem solving and control tasks, began with route planning and navigation for simulated robotic agents in maze solving problems. This evolved to encompass non stationary environments in order for the robotic agents to learn and adapt to changes in the environment [45], [67], [68]. The robot navigation idea has been taken one step further for information routing in the ever expanding world

of accessible information such as the internet or simply the telephone. The idea was to have an intelligent adaptive agent at each network node whose only concern is to route packets of information efficiently. However only local information was known to each reinforcement agent but the optimal solution, for routing information as quickly as possible from source to destination was a global one. It is this co-operation and co-ordination between agents which interests researchers and gives the information network an intelligent and adaptive capability.

By following the evolution the application of multi-agent reinforcement learning is explored for the control of large-scale systems. In particular the use of multi-agent learning (or distributed learning control) by means of TD reinforcement and the adaptive heuristic critic neuron like structures was used to optimise the performance of a multi-machine power system.

7.1 Multi-Machine Power System Learning Control

The usefulness of TD reinforcement learning for optimising control performance in general has been described by many examples in chapter 1 and demonstrated in chapter 5.

The concern here is to apply reinforcement learning for control of synchronous generators in a multi-machine power system. Previous studies used the learning automata-based reinforcement learning for controlling power systems [28] [69], the TD reinforcement learning method is proposed to optimise controller parameters, on-line in real time, in the large-scale power system. Two types of parameterised controller were evaluated, the PID controller and the fuzzy logic controller. The multi-agent reinforcement learning scheme has been evaluated in a simulation study. The simulated system is concerned with a

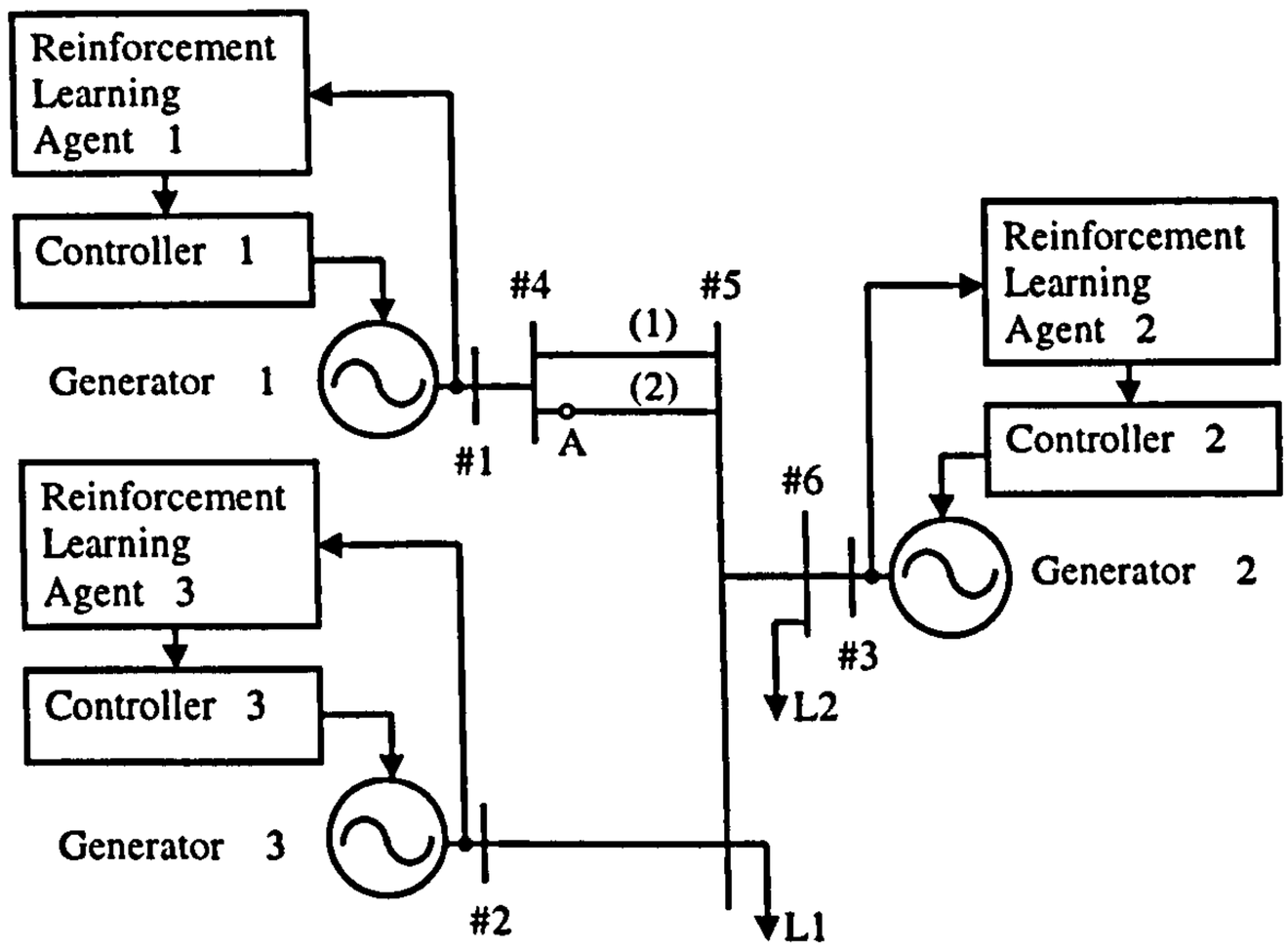

Figure 7.1: Optimisation of power system controllers using multi-agent learning [70].

three-machine power system which has multi-mode oscillations. The simulation results show that the proposed scheme has satisfactory learning performance and following a fault disturbance such as a three-phase-grounding short circuit, the learning controllers can damp out the multi-mode oscillations of the power system rapidly.

7.2 Architecture of network

List of symbols used

δ_i	=	rotor angle of $i-$th machine, in degrees
ω_i	=	rotor speed of $i-$th machine, in radian per second
ω_0	=	synchronous speed, in radian per second
$V_{di},\ V_{qi}$	=	stator voltages in d- and q$-axesof\ i-$th machine, in $p.u.$
$I_{di},\ I_{qi}$	=	stator currents in d- and q-axes of $i-$th machine, in $p.u.$
V_{ti}	=	generator terminal voltage of $i-$th machine, in $p.u.$
P_{mi}	=	input mechanical power of $i-$th machine, in $p.u.$
$P_{ei},\ Q_{ei}$	=	power and reactive power delivered at the terminal of $i-$th machine, in $p.u.$
E'_{qi}	=	internal transient voltage in q-axis of $i-$th machine, in $p.u.$
H_i	=	inertia coefficient of $i-$th machine, in seconds
D_i	=	damping power coefficient of $i-$th machine, in $p.u.$
$x_{di},\ x_{qi}$	=	synchronous d- and q-axis reactances of $i-$th machine, in $p.u.$
x'_{di}	=	transient reactance in d-axis of $i-$th machine, in $p.u.$
T_{doi}	=	the field winding time constant, in seconds
Y_{ij}	=	the $G_{ij} + j\ B_{ij}$ transfer admittance between buses i and j, in $p.u.$
Y_{ii}	=	the $G_{ii} + j\ B_{ii}$ self-admittance of bus i, in $p.u.$
$G_{ij},\ B_{ij}$	=	transfer conductance and susceptance between buses i and j, in $p.u.$
y_{ij}	=	the magnitude of Y_{ij}, in $p.u.$
α_{ij}	=	$\pi/2-\ \arctan(-B_{ij}/G_{ij})$
u_i	=	excitation control of $i-$th machine, in $p.u.$
0	=	subscript, representing steady state of variables

L1 and L2 in Figure 7.1 are the system load admittances given by, $L1 = 8.6 - 6.88j$ and $L2 = 9.8 - 7.8j$ The operating conditions for the generators are shown in Table 7.5. All the power system parameters are presented in Tables 7.1, 7.2, 7.3, 7.4.

7.2.1 Multi-Machine Power System Model

The three generators in the power system network were each simulated using a fifth-order model given by equations (7.2.1).

$$\begin{aligned}
\frac{d(\delta)}{dt} &= \omega - \omega_0 \\
\frac{d(\omega)}{dt} &= \frac{\omega_0}{2H}(P_m - P_e) \\
\frac{d(E_q')}{dt} &= \frac{1}{T_{d0}'}[E_{fd} - E_q] \\
T_{d0}''\frac{d(E_q'')}{dt} &= -E_q'' + E_q' - (X_d - X_d'')I_d + \\
&\quad T_{d0}''\frac{d(E_q')}{dt} \\
T_{q0}''\frac{d(E_d'')}{dt} &= -E_d'' + (X_q' - X_q'')I_q \\
U_d &= -RI_d + X_q''I_q + E_d'' \\
U_q &= -RI_q - X_d''I_d + E_q'' \\
P_e &= U_dI_d + U_qI_q
\end{aligned} \tag{7.2.1}$$

All generators come equipped with AVRs but only generator 2 is equipped with a governor. The AVR action is determined by the following equation (7.2.2),

$$E_f = \frac{K_A}{1 + T_{AS}}(V_{\text{ref}} - V_t) \tag{7.2.2}$$

and the governor is modelled using equation (7.2.3).

$$g = \left[\frac{(a+b)}{(1+T_{gs})}\right]\frac{d(\omega)}{dt} \tag{7.2.3}$$

Parameter values for the multi-machine power system model

Loads (admittances) in *p.u.*

$$L1 = 8.6 - 6.88j, \qquad L2 = 9.8 - 7.8j$$

Node No.	Impedance
1-3	0.015+0.10j
4-5(1)	0.075+0.50j
4-5(2)	0.1125+0.75j
2-5	0.060+1.40j
5-6	0.225+1.50j
3-6	0.025+0.15j

Table 7.1: Transmission line parameters in *p.u.*

7.3 Multi-Agent Learning PID control

The adaptive heuristic critic (AHC) was used as the basis for optimising the control parameters in this study of TD reinforcement learning. The adaptive capability of the neuron like structures enable learning and hence intelligent actions to be performed. Each of the control parameters is optimised using an *adaptive critic element* (ACE) and an *associative search element* (ASE). As learning proceeds, the ASE constructs associations between inputs and outputs by searching under the influence of a reinforcement feedback signal r which gives an immediate indication of how good the chosen actions are. Figure 7.2 shows how the neurons are configured into a neural network for optimising the control parameters of a single PID controller. Each of these TD neural networks acted as a learning agent (see Figure 7.1). The idea is that for a large-scale system these learning agents can be distributed through the

Parameters	Unit 1	Unit 2	Unit 3
x_d	1.0260	0.1026	0.1026
x_q	0.6580	0.0658	0.0658
x'_d	0.3390	0.0339	0.0339
x''_d	0.2690	0.0269	0.0269
x''_q	0.3350	0.0335	0.0335
T'_{d_0}	0.3670	0.3670	0.3670
T''_{d_0}	0.0314	0.0314	0.0314
T''_{q_0}	0.0623	0.0623	0.0623
H	2.8000	28.000	28.000

Table 7.2: Parameters of the generators in *p.u.*

Parameters	Unit 1	Unit 2	Unit 3
T_g	0.250000	0.250000	0.250000
a	-0.001328	-0.00015	-0.00015
b	-0.170000	-0.01700	-0.01700

Table 7.3: Parameters of the governors.

system at relevant points to solve a local optimisation problems. By solving their own local optimisation problem they also effectively coordinate with each other to solve a global optimisation problem. The TD reinforcement learning agents do not share knowledge with each other, and effectively they are unaware of each others existence. One advantage is that they can still solve a global optimisation problem if one of them breaks down, in effect this principle has built in redundancy. For any size system it is a simple case of scaling the required number of reinforcement learning agents to suit the task being solved.

Figure 7.3 shows how the TD neural network is implemented with a PID

Parameters	Unit 1	Unit 2	Unit 3
K_A	30	30	30
T_A	0.01	0.01	0.01
K_e	1	1	1
T_e	0	0	0

Table 7.4: Parameters of AVRs and exciters.

Generator	$P(p.u.)$	$Q(p.u.)$	$V(p.u.)\angle\theta$(degree)
1	0.7564	1.0930	$1.300\angle 5.0$
2	7.5537	9.0095	$1.250\angle 6.0$
3	9.2769	9.6574	$1.100\angle 0.0$

Table 7.5: Generator operating conditions.

controller in order to optimise the system performance. The PID controller used in this simulation study is given by equation (7.3.1).

$$\begin{aligned} u(t) &= u(t-1) + K_P[y(t) - y(t-1)] + \\ &\quad K_I y(t) + K_D[y(t) - 2y(t-1) + y(t-2)] \end{aligned} \tag{7.3.1}$$

The predictions for the AHC using TD learning is the same as that used in the $3rd$ order dynamic system in chapter 5 equation (4.2.2) as were the ACE weights, ASE weights and trace decays being updated using equations (4.2.3) (4.2.6) respectively. The $\hat{r}(t)$ outputs from the ACE was calculated from equation (4.2.5) and the action selection output for the ASE was given by equation (4.2.8) with a sigmoidal activation function determined by equation (5.2.5).

The reinforcement feedback $r(t)$ was formulated as a cost function using equation (7.3.2) and used to measure the performance of the controller after

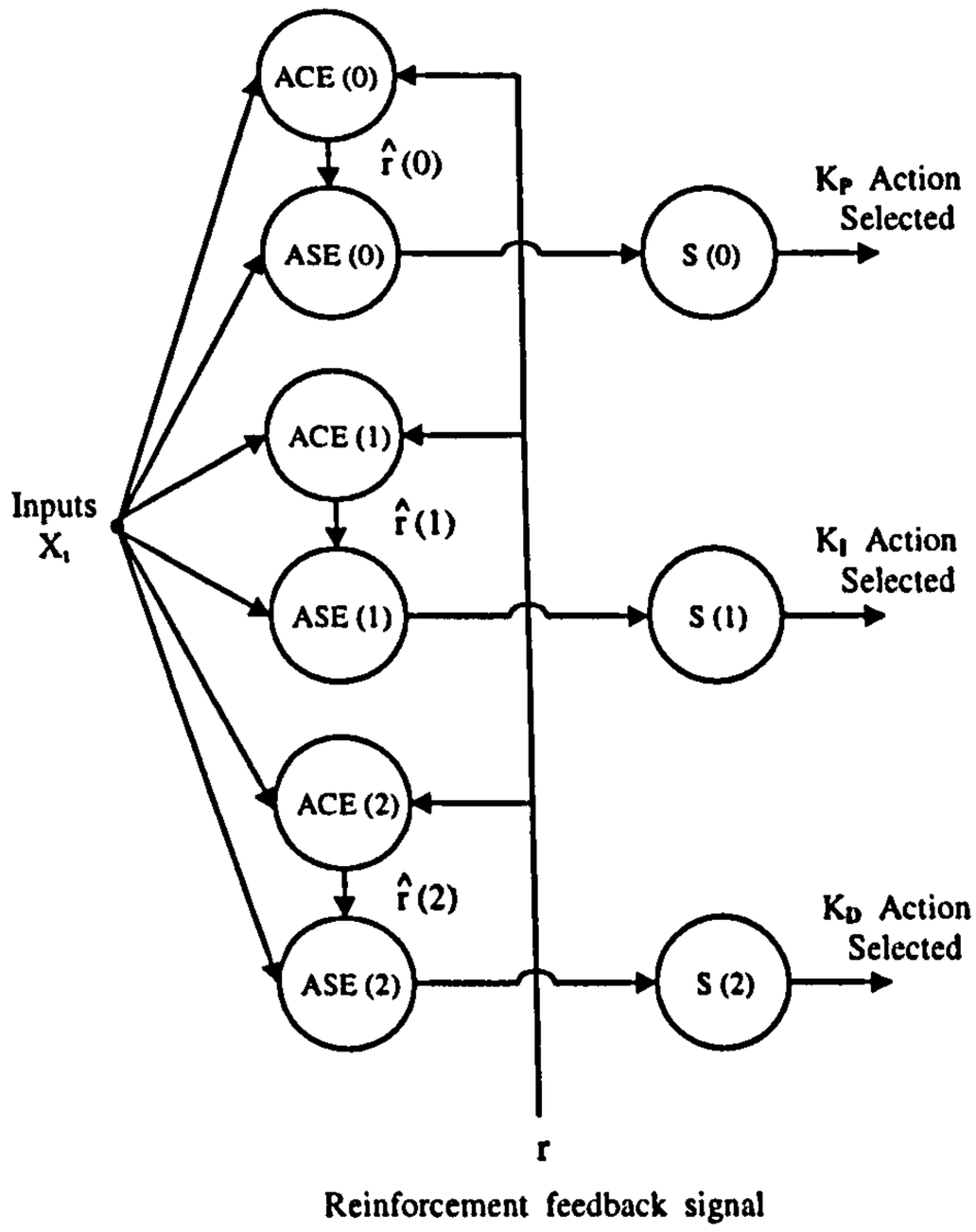

Figure 7.2: The TD neural network architecture used to optimise a PID controller [70].

each K_P, K_I and K_D action selection.

$$r(t) = \frac{1}{N}\sum_{i=1}^{N}(y_{\text{ref}}(t) - y(t))^2 \tag{7.3.2}$$

where N is the window size or sampling interval for calculating the mean and covariance inputs x_i to the TD neural network from the measured output $y(t)$.

7.3.1 Simulation Results

The power system investigated is divided into three areas. Each area serves its own load with only a small load transferred through the transmission line. Both generators 1 and 2 serve load $L1$, while generator 3 serves load $L2$, with only a small load transferred through the transmission line. The power system

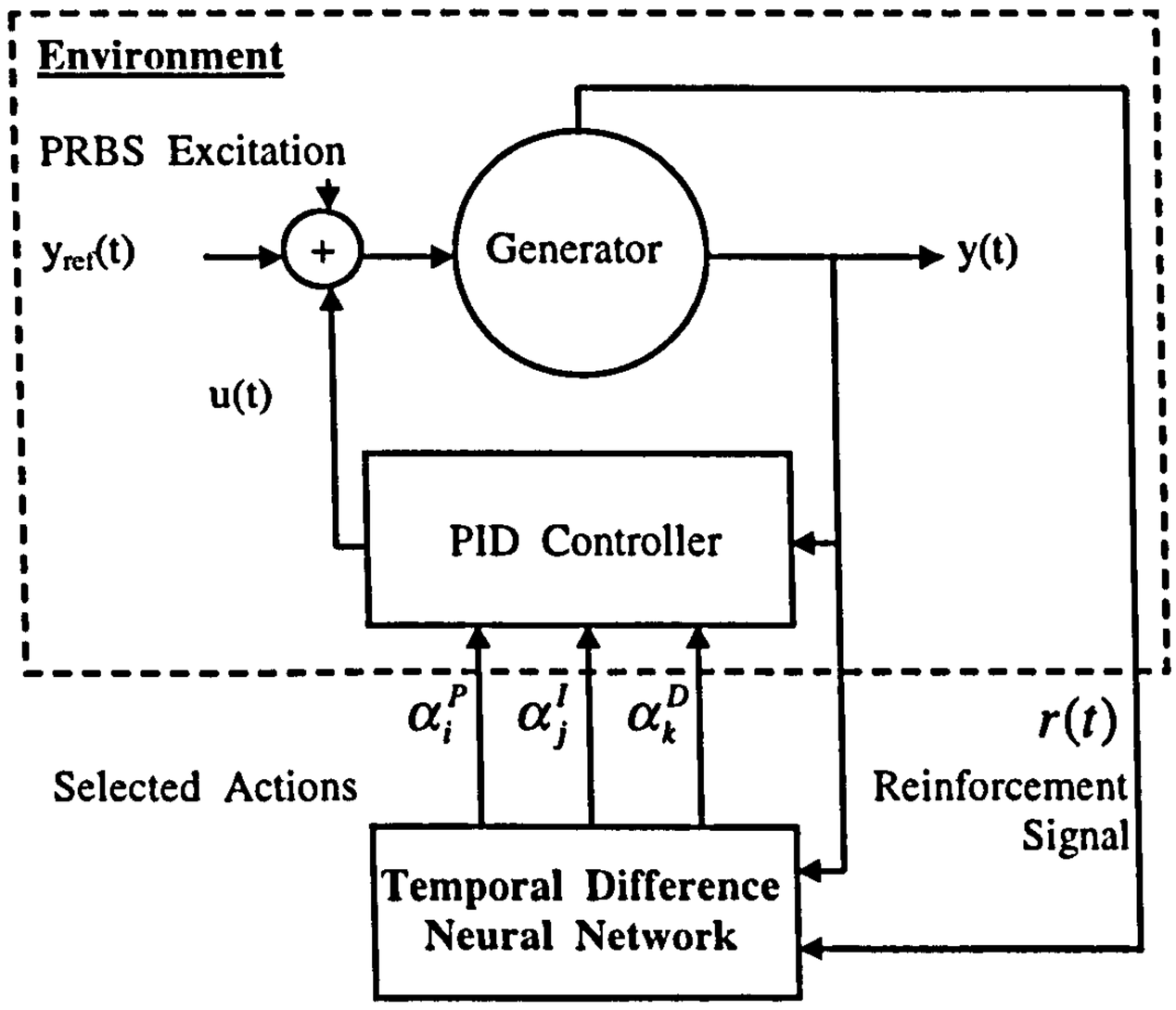

Figure 7.3: Using TD reinforcement learning to optimise a PID controller for each generator in the multi-machine power system [70].

possesses a nature of multi-mode oscillations [71], which can be seen in Figure 7.4. The inertia constant for generator 1 was 10% of that of generators 2 and 3. The outputs of all exciters and controllers are limited to 7 *p.u.*. Each machine is controlled by a learning PID controller. A three-phase-to-ground short circuit at one of the double transmission lines, at point A, as illustrated in Figure 7.1, was simulated. The transmission line is switched off at $t = 0.3\ ms$ and is switched back on at $t = 0.7\ ms$ when the fault is cleared.

The PID control parameters are optimised using a TD learning neural network. Each PID parameter is set a range and within this range TD learning is used to try and find the best value for each PID parameter through a process of selection and then observing the results of the multi-machine performance after a three phase short circuit. As each action α_i^P, α_j^I and α_k^D (corresponding to a parameter value) is chosen the probability of it being chosen is also updated, to the effect that if the action gives a good performance then the probability of choosing that action again is increased. After many search and observe train-

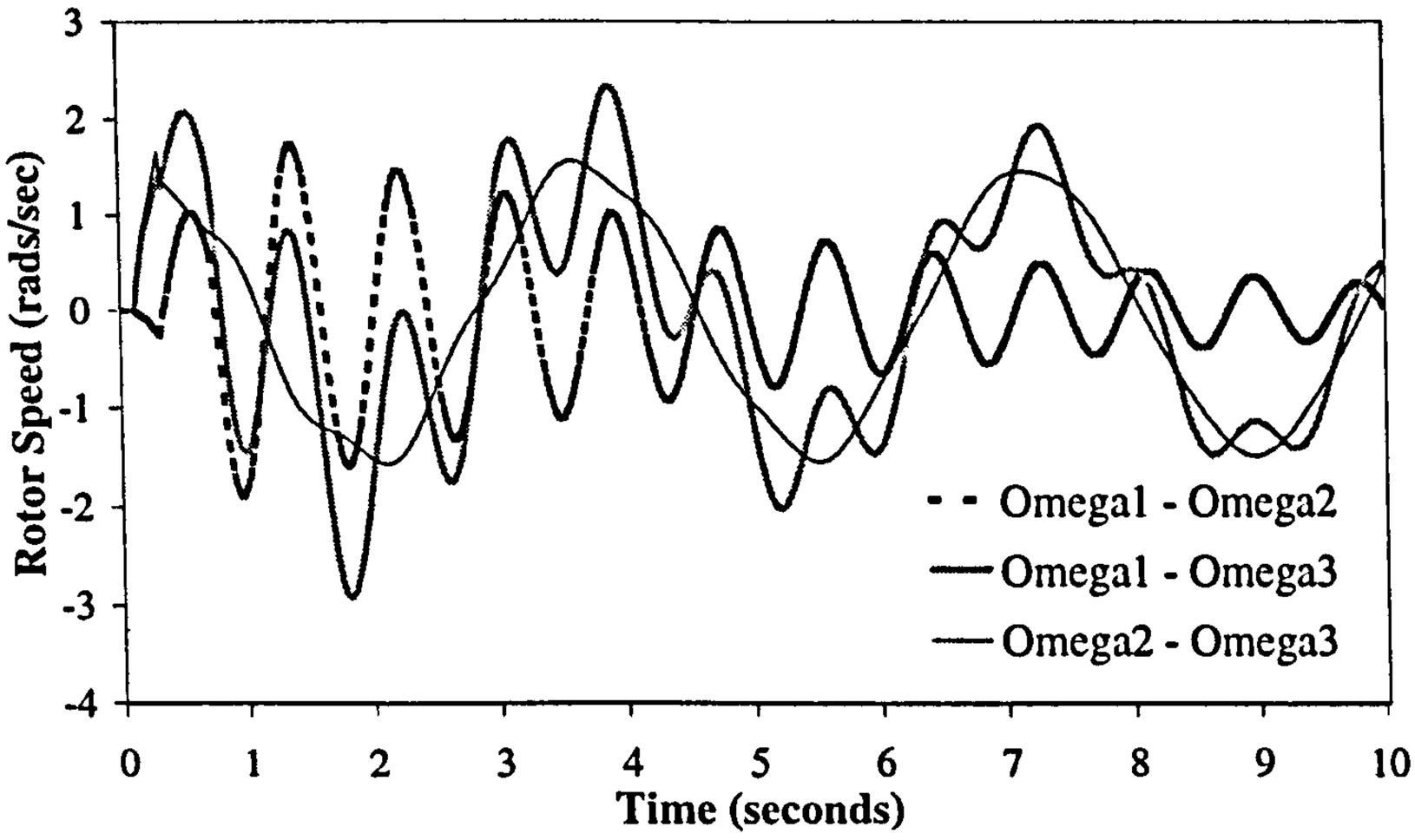

Figure 7.4: Stability response of the multi-machine power system after recovering from a three phase short circuit without control.

ing sequences the best control parameters will be the ones whose probabilities are closest to one. These should then be the optimal control parameters for controlling the multi-machine system in the event of a three phase short circuit fault. The range of values available to each PID control parameter K_p, K_i, K_d, were [1000, 3000], [1000, 2000] and [100, 200] respectively, with a choice of ten actions available in the range set.

In general for a desired learning performance, the larger the window size N is in equation (7.3.2), the better $r(t)$ can be formulated and in our tests $N = 2000$ was used throughout. $y(t)$ and $y_{\text{ref}}(t)$ are the changes in angular velocity $\Delta\omega$ of the generator rotor with a reference of $\Delta\omega_{\text{ref}} = 0\ rads\ s^{-1}$. $\Delta\omega$ was used in this simulation as our performance parameter to be optimised. However, other outputs from the generators could be used also, the only requirement is that the evaluative feedback can show how good the control action is so that a useful $r(t)$ can be inferred from the measurement.

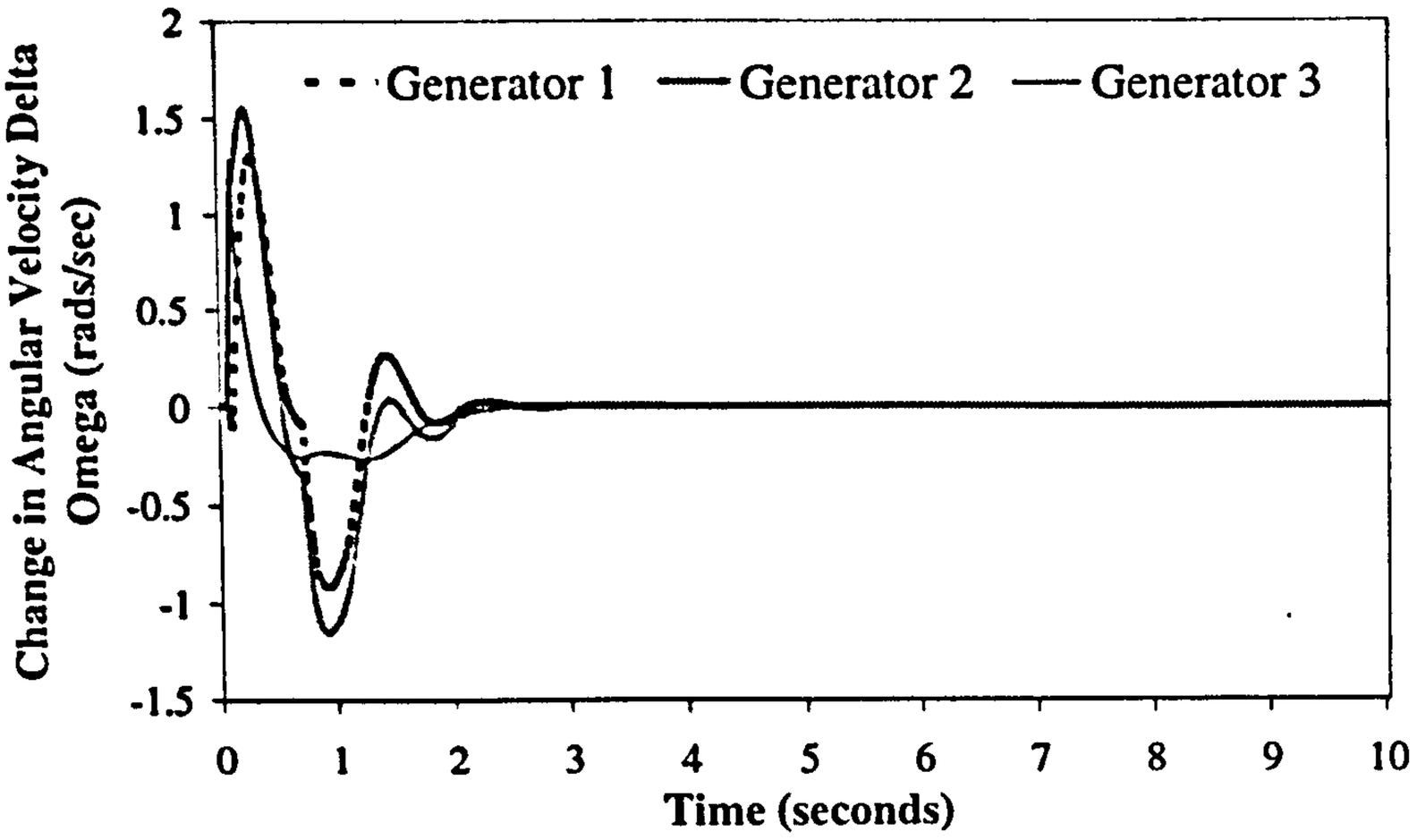

Figure 7.5: Stability response of the multi-machine power system after recovering from a three phase short circuit with multi-agent control.

The multi-agent learning system can successfully learn to optimise the controller parameters. Figure 7.5 shows the deviation of the angular speed $\Delta\omega$ of the generators controlled by the learning PID controller equipped on each machine in the power system. The simulation runs for 10000 samples with a time step interval of 0.001 corresponding to $10ms$ of time. Training of the TD neural network is achieved by constantly running the program, after 100 runs of 10000 samples we observe the accumulated number of times each action has been selected, the ones chosen most often should then be the optimal control parameters. Each controller is then set to the best parameters found during learning and the response of $\Delta\omega$ observed. The PID control output can be seen in Figure 7.6 showing the control outputs settling, during the transient process, which provide optimal control to the multi-machine power system. The terminal voltages of each generator also recovers quickly after the three-phase-short circuit fault, which is illustrated in Figure 7.7.

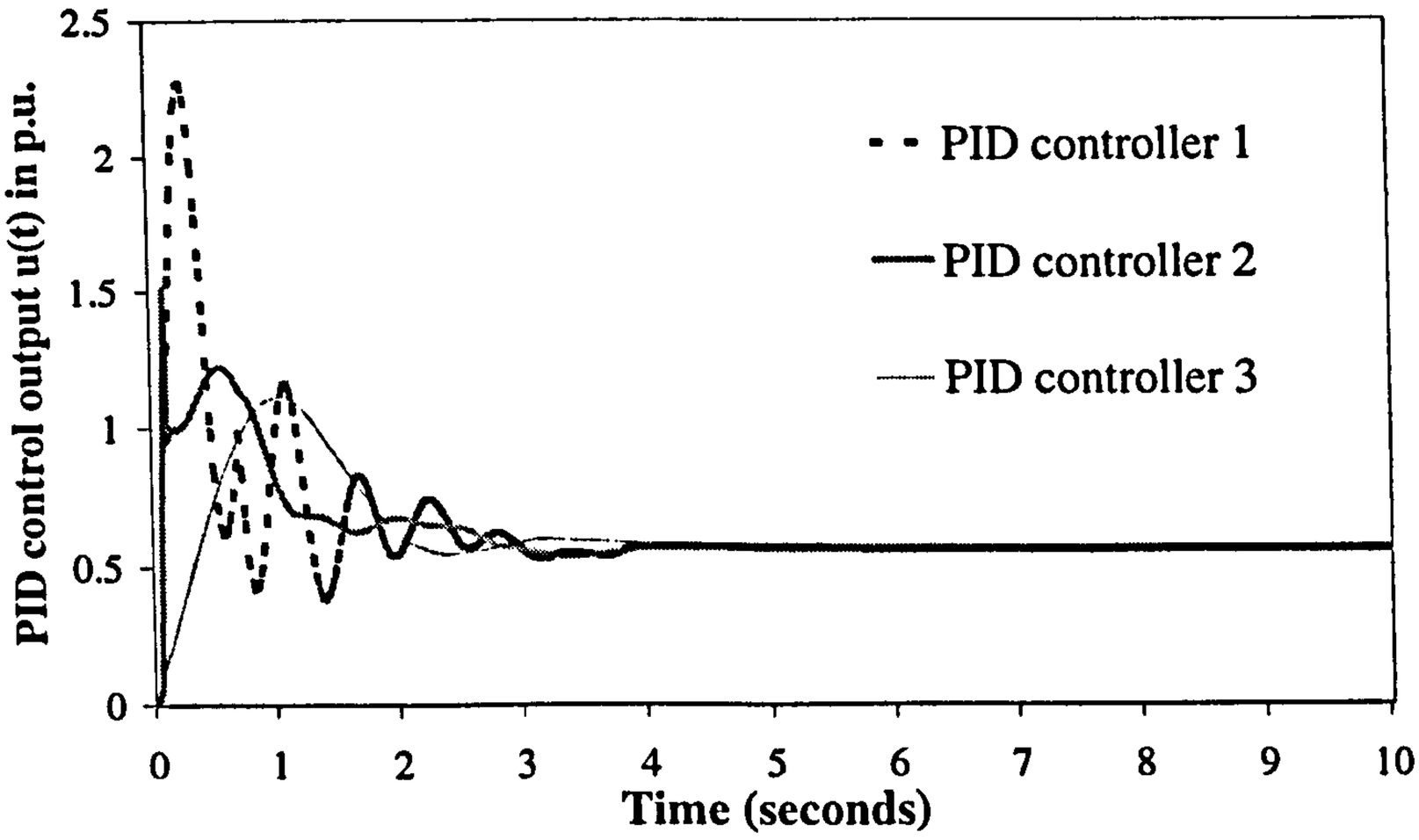

Figure 7.6: PID control outputs optimising the multi-machine power system performance after a three phase short circuit.

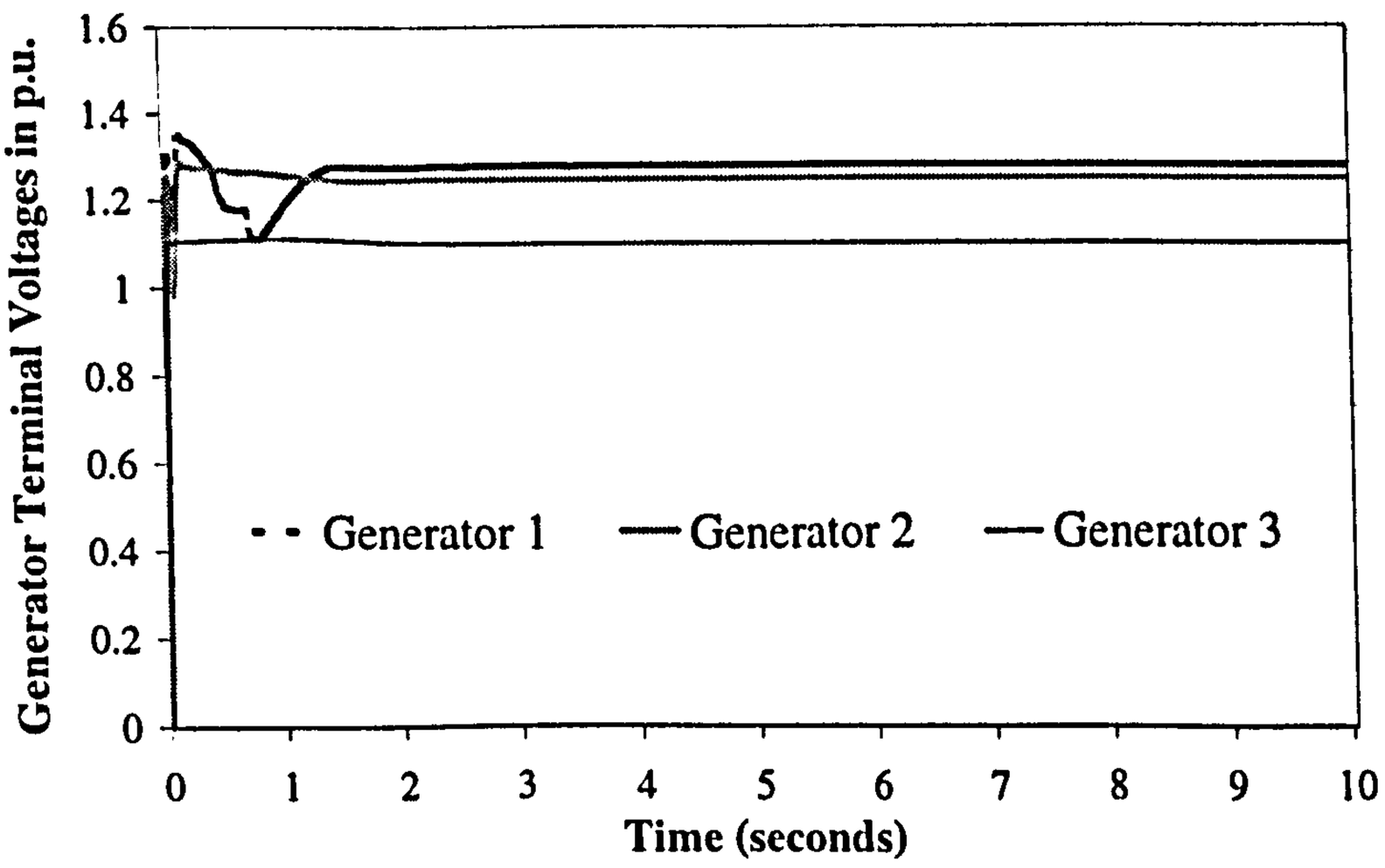

Figure 7.7: Terminal voltage recovery for the multi-machine power system after a three phase short circuit.

The simulation results show that the presented scheme had a satisfactory learning performance and after a fault disturbance, such as a three-phase-grounding short circuit, the learning PID controllers successfully damp out the multi-mode oscillations of the power system rapidly.

7.4 Multi-Agent Learning Fuzzy Control

The next simulation involved the application of *Temporal Difference* (TD) learning to optimise fuzzy logic controllers in a distributed power system. As used in the evaluation of the PID controller previously, the three machine power system is simulated to evaluate the dynamic performance of the generators as well as observing the multi-agent learning control performance. The parameters for each of the fuzzy logic controllers are optimised in parallel using TD reinforcement learning agents. The fuzzy control parameters are updated independently and simultaneously.

The same TD learning neural network and multi-machine power system (Figure 7.1) was used, but a fuzzy logic controller was used instead of a PID controller. The control actions from each TD learning neural network that makes up a learning agent is shown in Figure 7.8. As for the PID simulation study the cost function to evaluate $\hat{r}(t)$, $r(t)$ and output function were updated in the same manner using the same equations, as were the AHC weights and trace decays. A flow chart, Figure 7.9 describes how the learning fuzzy control was implemented.

Figure 7.10 shows how the TD neural network is implemented with a fuzzy logic controller in order to optimise a single synchronous machine.

For design of adaptive fuzzy logic controllers an important aspect to consider is the choice or shape of the membership functions. A non-linear continuous fuzzy membership function of the form shown in Figure 7.11 was employed for the fuzzy logic controller. This type of membership function is useful for

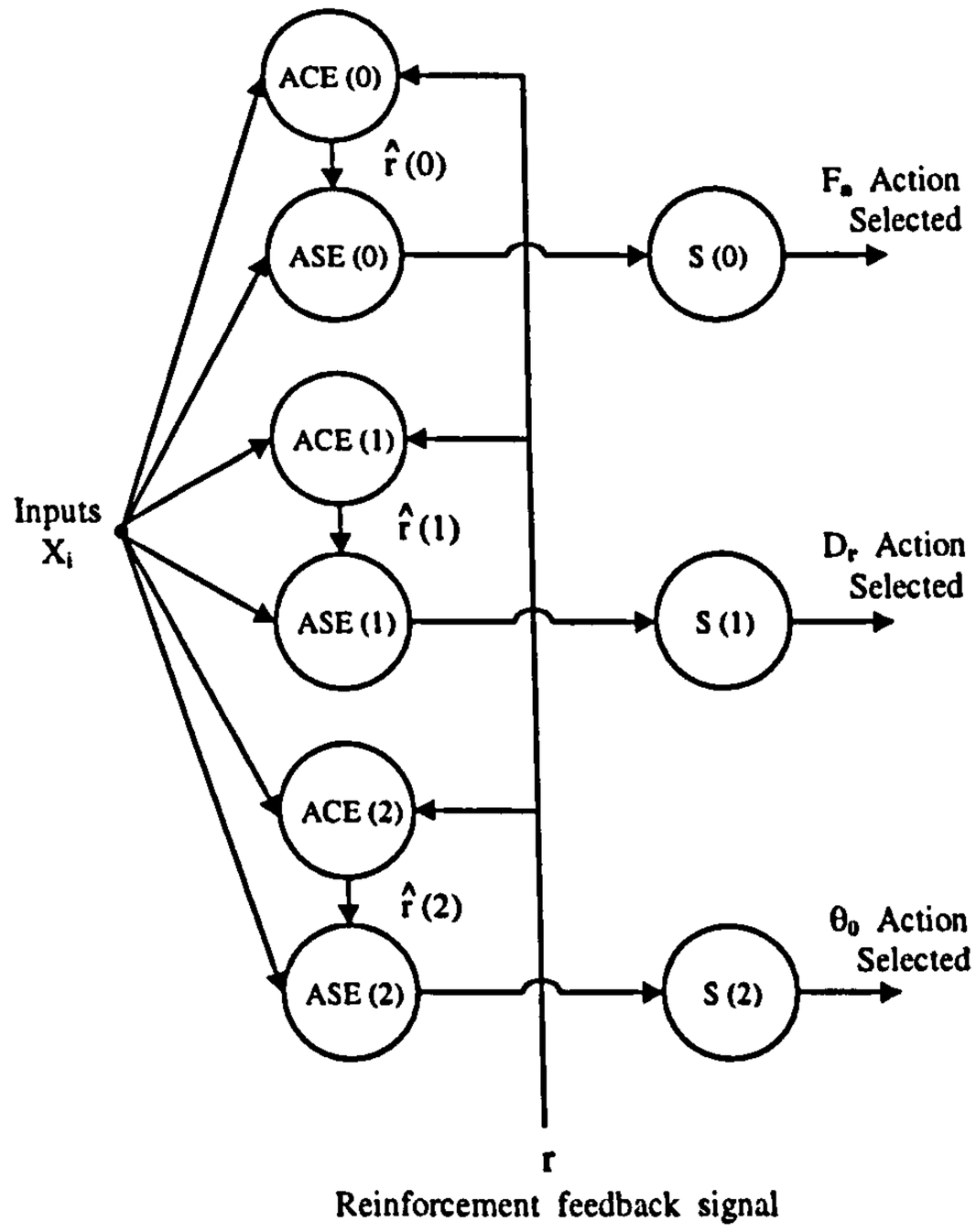

Figure 7.8: The temporal difference neural network architecture for optimising a fuzzy logic controller [72] [73].

determining the accelerating and decelerating power needed to control each synchronous generator as it deviates from normal operating speeds. These fuzzy membership functions $N(\theta)$ and $P(\theta)$ (Li et al [69] and Hassan et al [74]) are described by equations (7.4.1) and (7.4.2). It can be seen that the function $S(\theta, a, b, c)$ has limits $[0, 1]$ and that changes in a, b, c as variables in θ give us the desired result of changing the shape of the membership function in Figure 7.11. In order to determine a control, the synchronous generator speed deviation is sampled to give an indication of the rate of change of $\Delta\omega$ over a sampling interval T_s.

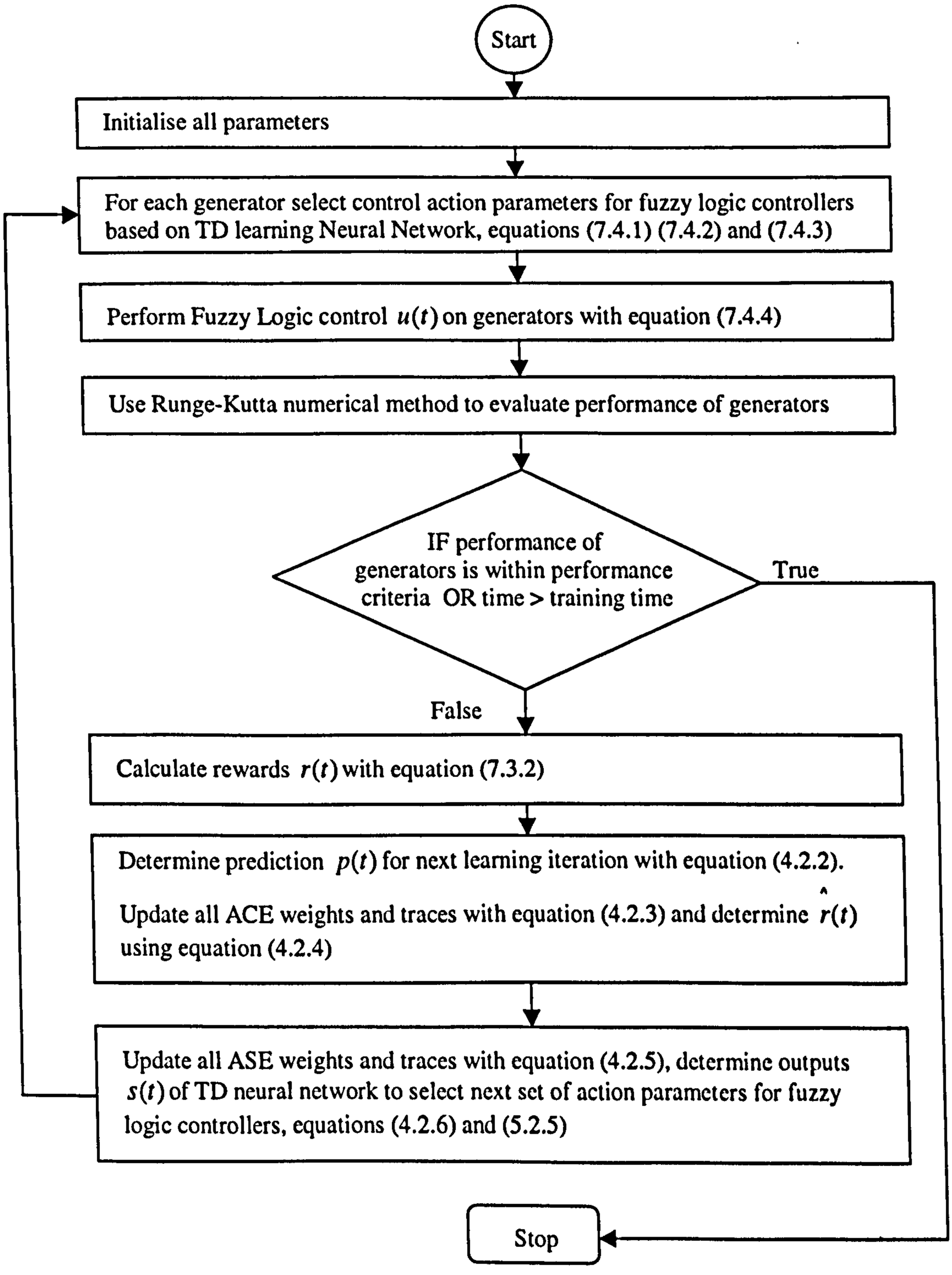

Figure 7.9: Flow chart for multi-machine learning fuzzy logic control with TD learning neural networks.

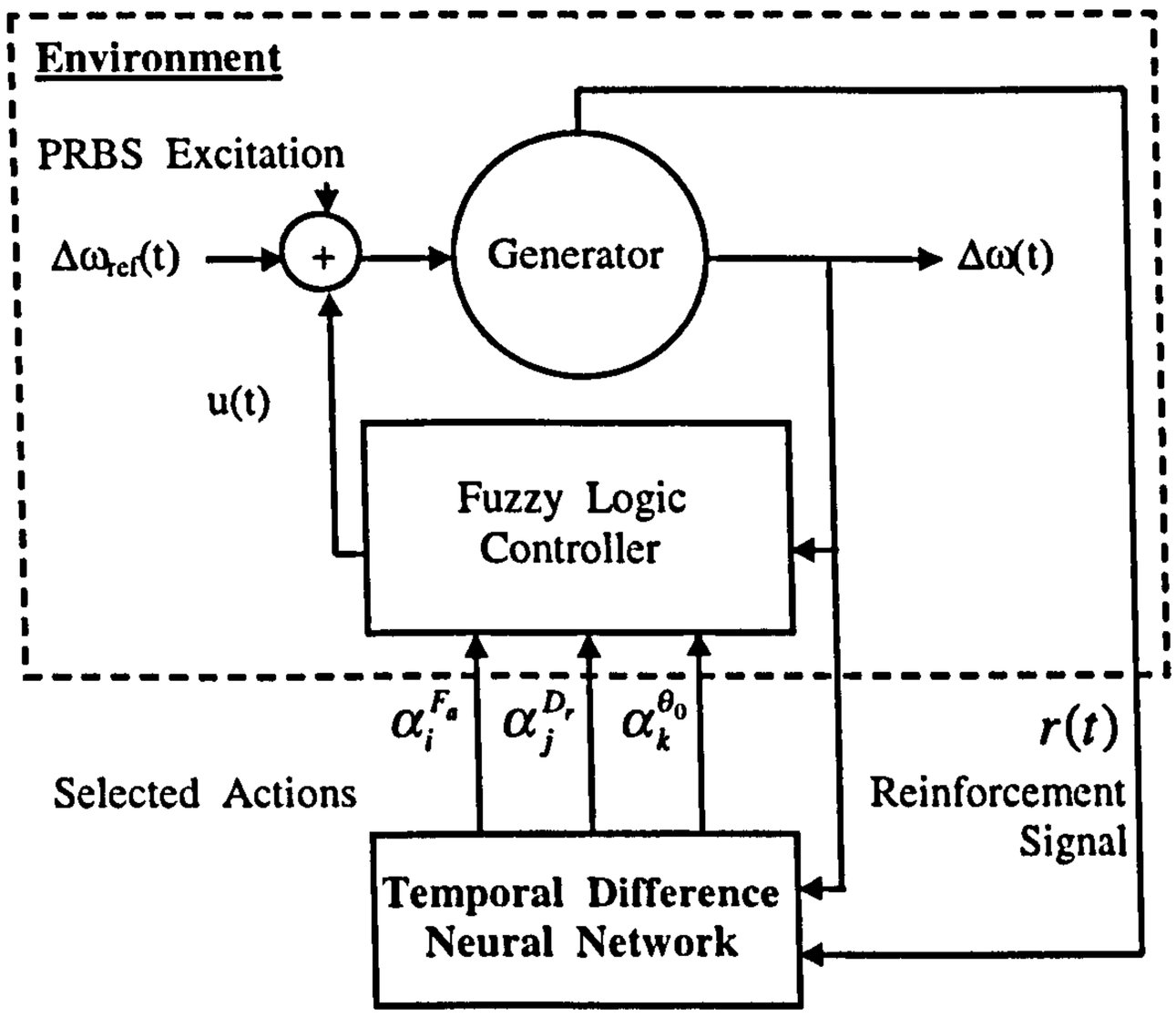

Figure 7.10: Using TD reinforcement learning to optimise a fuzzy logic controller [72] [73].

$$
\begin{aligned}
N_s(\theta) &= \begin{cases} 1 - S(\theta; \theta_0, \theta_{m1}, \theta_m) & \theta \leq \theta_m \\ S(\theta; \theta_m, \theta_{m2}, 2\pi) & \theta > \theta_m \end{cases} \qquad (7.4.1) \\
P_s(\theta) &= 1 - N_s(\theta)
\end{aligned}
$$

where

$$
\begin{aligned}
\theta_m &= (2\pi + \theta_0)/2 \qquad (7.4.2) \\
\theta_{m1} &= (\theta_0 + \theta_m)/2 \\
\theta_{m2} &= (2\pi + \theta_m)/2
\end{aligned}
$$

and

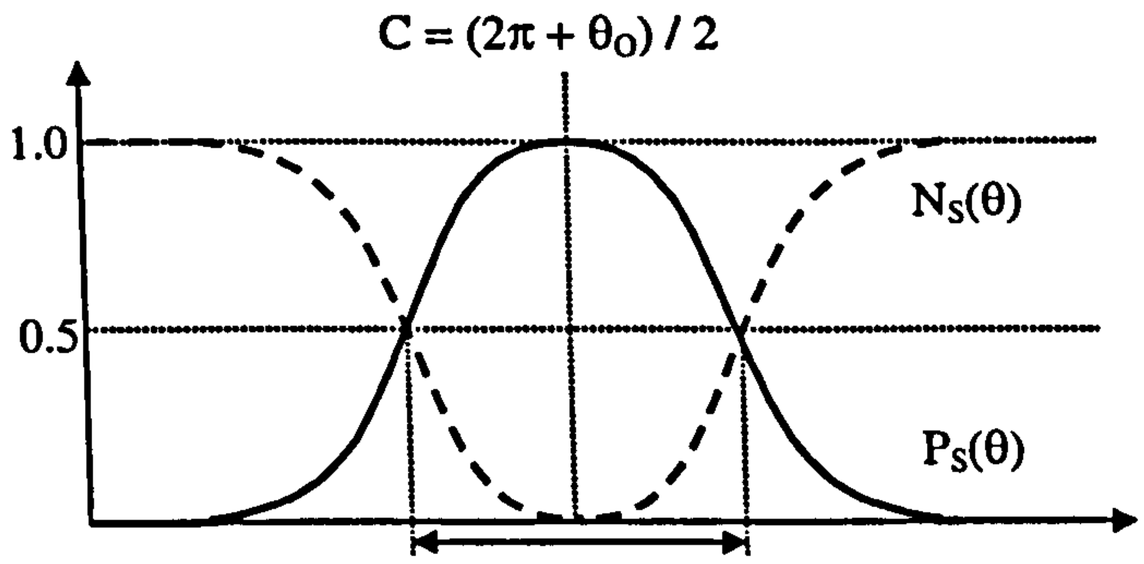

Figure 7.11: Fuzzy membership function [72] [73] [74] [69].

$$S(\theta; a, b, c) = \begin{cases} 0; & \theta \leq a \\ 2((\theta - a)/(c - a))^2; & a < \theta \leq b \\ 1 - 2((\theta - c)/(c - a))^2; & b < \theta < c \\ 1; & \theta \geq c \end{cases} \tag{7.4.3}$$

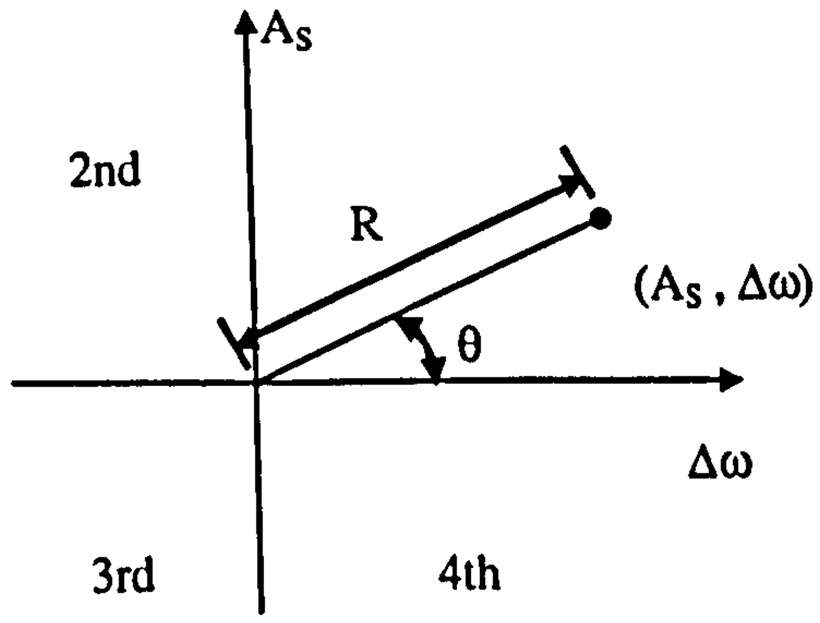

Figure 7.12: The rotor angle position on the phase plane [72] [73] [74] [69].

The variable θ can be determined from Figure 7.12, where A is the rotor acceleration, $A = (\Delta\omega(t) - \Delta\omega(t-1)/T_s)$; A_S is a scaled acceleration proportional to a scaling factor F_a, $A_s = F_a A$. Both A_s and $\Delta\omega$ are used to compute $R(t)$,

where $R(t) = \sqrt{(A_s^2 + \Delta\omega^2)}$. The determination of θ can be obtained using $\sin(\theta) = \mathrm{A_s/R(t)}$ for $0 < \theta \leq 2\pi$. The control action from the fuzzy logic controller is given by equation (7.4.4)

$$u(t) = G_c(t)[N_s(\theta) - P_s(\theta)]u_{\max} \tag{7.4.4}$$

where G_c is a ramp function $G_c(t) = R(t)/D_r$ for $R(t) < D_r$, $G_c(t) = 1$ when $R(t) \geq D_r$, and $u_{\max}$ is a coefficient limiting the maximum output control of the fuzzy logic controller. In order to change the distribution of the fuzzy membership function for improved control performance, it is observed that the three parameters, θ_0, F_a and D_r can be adjusted and optimised using the temporal difference reinforcement learning.

7.4.1 Simulation Results

All power system parameters and initial conditions were left unchanged as for the early study with the PID controller. The outputs to all exciters and controllers were limited to $7p.u.$ Each machine is controlled by a learning fuzzy logic controller, at point A in Figure 7.1 a three-phase-to-ground short circuit fault was introduced. The transmission line was switched off at $t = 0.3ms$ and then switched back on at $t = 0.7ms$ when the fault had been cleared.

The fuzzy logic control parameters are optimised using a temporal difference learning neural network. Each fuzzy control parameter is set a range and within this range temporal difference learning is used to try and find the best value for each parameter. The process of selection and then observing the results of the multi-machine performance after a three phase short circuit is used to evaluate the actions selected. As each action $\alpha_i^{\theta_0}$, $\alpha_j^{F_a}$ and $\alpha_k^{D_r}$ (corresponding to a parameter value) is chosen the probability of it being chosen is also updated, to the effect that if the action gives a good performance then the probability of choosing that action again is increased. After many search and observe training

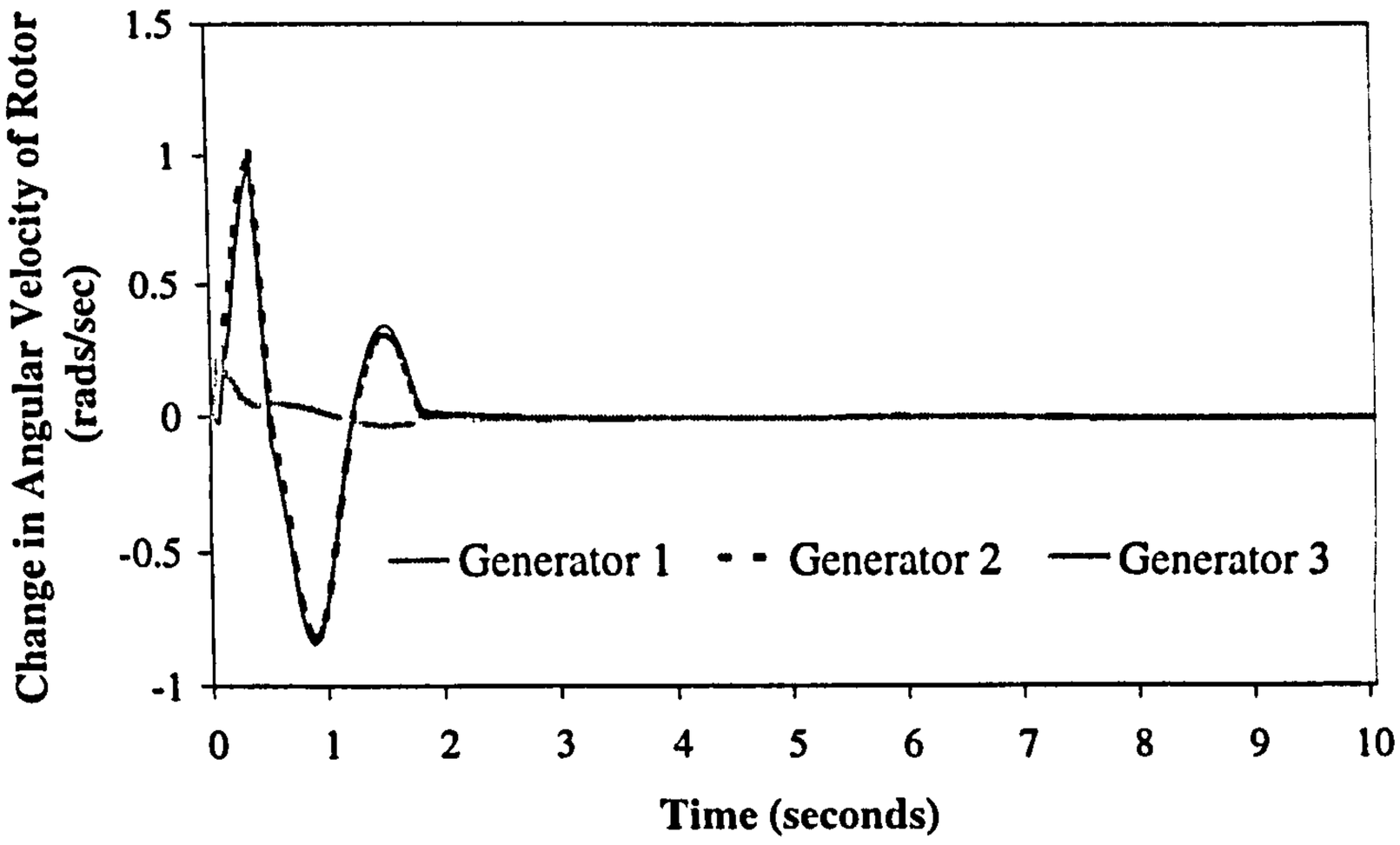

Figure 7.13: Generator speed responses to a three-phase-ground short circuit with multi-agent fuzzy control.

sequences the best control parameters will be the ones whose probabilities converge closest to one. These should then be the optimal control parameters for controlling the multi-machine system in the event of a three phase short circuit fault.

After initial evaluations the range of values for each control parameter parameter was set. Fuzzy logic controller one, corresponding to generator 1 was set the range $\theta_0 \in [1.5, 2.0]$ radians, $F_a \in [0.005, 0.01]$ and $D_r \in [0.01, 0.1]$. For controllers 2 and 3 a suitable range was found to be $\theta_0 \in [1.5, 2.0]$ radians, $F_a \in [0.01, 0.1]$ and $D_r \in [0.05, 0.5]$, all ranges were equally divided to give ten available actions within the set range.

The multi-agent learning system can successfully learn to optimise the control parameters. A comparison between the system without control, Figure 7.4 and with the fuzzy controllers optimised, Figure 7.13 shows that temporal difference learning can successfully select parameters that will lead to good

control performance of the multi-machine power system. The simulation runs for 10000 samples with a time step interval of 0.001 corresponding to 10*s* of time. Training of the temporal difference neural network is achieved by constantly running the program, after 1000 runs of 10000 samples we observe the accumulated number of times each action has been selected, the ones chosen most often should then be the optimal control parameters. Each controller is set to the current best parameters found during learning and the response of $\Delta\omega$ observed. Figure 7.14 illustrates the learning history for the θ parameter of the fuzzy logic controller.

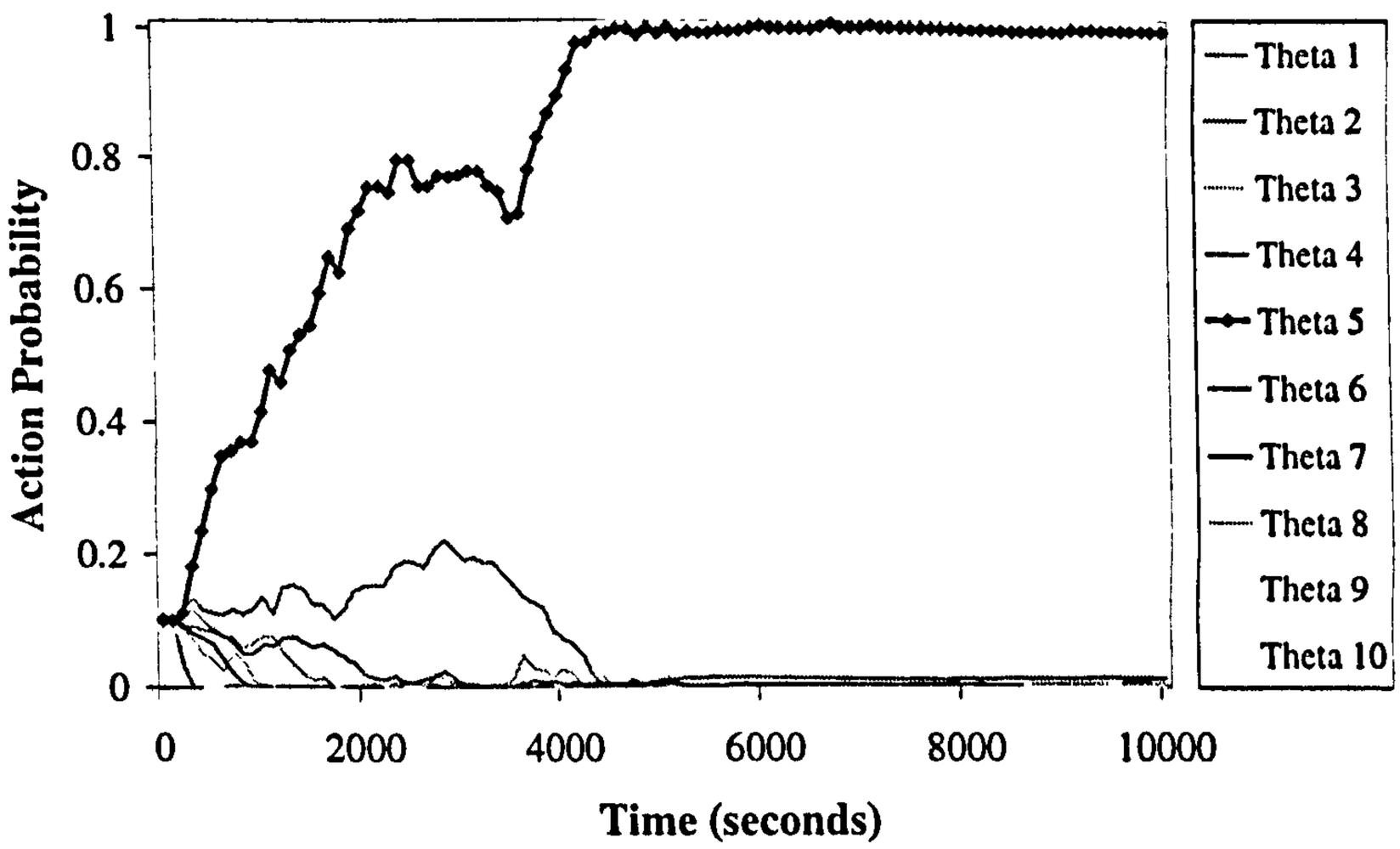

Figure 7.14: Optimisation of θ action parameters using temporal difference reinforcement learning.

As observed although training takes a long time to converge to probability one, the early trend shows that the TD learning agent can select optimal control actions quite quickly. The complete training time may not be required and may the use of an existing optimal (or even suboptimal) action selected may still be beneficial in a control situation, this will also enable reinforcement learning to

be used in applications that require quick or even real time response. Further the use of the chosen fuzzy membership function was due to its well established history in past control problems ([74] [69]). Perhaps the more common types of membership functions, such as trapezoidal or triangular membership functions may improve upon the control performance. The combination of reinforcement learning with fuzzy control as demonstrated here may provide a bridge between artificial intelligence and natural intelligence. Fuzzy logic already allows this to some extent but with the inclusion of a learning method that works by trial and error may lead to a more natural form of control which is more intuitive and easier to understand in human terms. From the other perspective this methodology also allows artificially intelligent machines to exhibit behaviour which seems more natural.

Chapter 8

Conclusion

8.1 Summary

The use of reinforcement learning to control a large-scale system such as a multi-machine power system has been demonstrated. The success of this method is highlighted partially by the learning automaton approach and mainly by the temporal difference learning approach and neural network configuration using adaptive heuristic critics working in parallel, providing robust control to dynamic systems.

Reinforcement learning control and multi-agent reinforcement learning control has been shown to be a valid method of optimising dynamic systems, this has been achieved by the following simulation studies.

1. Developing and using a temporal difference learning neural network in a $3rd$ order dynamic system for optimising PID control parameters

2. Using a team of learning automata to optimise PID control parameters in a $3rd$ order dynamic system.

3. Using a team of learning automata to optimise PID control parameters in a single-machine-infinite-busbar power system.

4. Using multi-agent temporal difference learning neural networks in a multi-machine power system to optimise PID control parameters.

5. Using multi-agent temporal difference learning neural networks in a multi-machine power system to optimise fuzzy logic control parameters.

8.2 General remarks

This study has involved multi-agent learning to co-ordinate and solve global dynamic optimisation problems. The learning agents are able to do this using only local limited knowledge and without the need to communicate this knowledge to each other has been demonstrated. Individually these learning agents can solve useful learning problems but they also provide a flexible way to expand to much larger scale problems, so that the number of learning agents can be increased or decreased to suit the tasks involved. Multi-agent learning results for the fuzzy logic control and PID control show that temporal difference learning can be applied to complex systems on-line, providing continuous and parallel learning to improve a controllers performance.

The advantages of using well known simple learning agents as building blocks for a distributed hierarchical structure is the robust nature of the learning system. If one learning agent fails then the rest can compensate within limits without it, since they will re-learn to optimise the same situation but with one less learning agent in the architecture.

The application of reinforcement learning to optimise parameterised controllers is both intuitive and flexible in learning control of dynamic systems. This method of learning and control has great potential for future engineering applications as well as being a useful tool for control of large-scale power systems. The co-ordination between learning agents in a multi-agent learning architecture also warrants further investigating.

8.3 Limitations of approach

Reinforcement learning is best applied to systems that will benefit from continuous learning control that works in the background continuously sampling the environment for changes, the controller parameters are set for general control but are then fine tuned and reset when an un-desired disturbance occurs to regain control of the system.

As with all learning approaches to problem solving the main criticism lies in the amount of time that is required for learning (or training). The flexibility of reinforcement learning control as it learns by trial and error is offset by the fact that, in order to come up with an optimal solution a price needs to be paid as it searches and explores the potential possibilities before settling and converging on the final solution. Without containing any preset information to improve its initial search the price in learning time can be expensive. This is somewhat offset by modern computational processing power and trends suggest a rapid increase in processing power in the future. Some applications requiring critical real time performance may not benefit so well from reinforcement learning control, while a model or some initial knowledge of a system may help, this will only limit the learning agent for that given situation or environment, thus removing much of the flexibility of learning by experience. As with all control methods some applications will be better suited for reinforcement learning than others.

Another limit found in the simulation studies was the fact that depending upon the preset parameter ranges for the action selection process, learning a parameter sometimes resulted in two or more actions having similar or equal chances of being selected and so no one unique solution was found. In general these actions or parameter values tended to be close to one another (having a similar value) and each provided reasonable control performance when evaluated individually in the final control test after learning. This indicates that

the learning agent may be having problems withg local optima as examplified in the results seen in Chapter 5, Figure 5.14

The solution may be to provide a continuous action space, unlike the discretised one used in the simulation study or just provide for a longer learning time, but as mentioned above increased learning time may not be so practical. Using a continuous action space new methods of selecting an action will need to be implemented so that the learning agent can expand the action range for general search then decrease and home in on a particular action value when it has identified the region the optimal action resides in.

8.4 Recommendations for further study

The emphasis in co-ordinated control using multi-agent reinforcement learning relies upon the co-operation between learning agents, each solving it's own local optimisation problem without impeding the progress of the others. Another approach to learning as in biological systems is competitive learning or survival of the fittest. A method proposed by Humphrys [26] and based on Q-learning follows this approach is one existing example and similar method may be devised in order to solve an optimisation problem based on competing learning agents.

PID and fuzzy logic controllers have been used to control the excitation system in the power system control problem, alternative types of control can be implemented. In the literature the use of *flexible A.C transmission systems* (FACTS) provide another application of reinforcement learning control in power systems. Many FACTS devices exist [63] [64] [65] providing a large scope for further investigation.

From power systems to other large-scale control problems in engineering further studies in distributed learning and alternative architectures for arranging the multi-level agents can be improved upon. The use of such multilevel

learning systems, accelerates the learning process (more so for the initial rate of learning) and simplifies their realisation. Many of the easily identifiable distributed systems have been investigated, traffic light control [51]; packet routing in information networks [53]; channel allocation in cellular phone systems [44] and improving elevator performance in large office buildings [43]. One idea suggested by Narendra and Thathachar [17], but so far has not been studied in the reinforcement learning context, mentions potential applications regarding priority assignment in a queuing system. Priority queuing could be applied to task scheduling which is important in industry for maintaining inventory and maximising efficiency in production.

Reinforcement learning is a fundamental and essential process in all animals and has helped us achieve things that were not possible until tried (and tried and tried). This concept has been imitated to enable machines (or learning agents) to exhibit intelligent behaviour. The scope of reinforcement learning encompasses not only engineering but also science, psychology, biology and maths and may some day include other disciplines that so far seem unrelated as we seek to improve our knowledge of reinforcement learning both natural as well as artificial.

Bibliography

[1] CINLAR E. *Introduction to Stochastic Processes.* Prentice-Hall, 1975.

[2] POZNYAK A.S. and NAJIM K. *Learning Automata and Stochastic Optimisation.* Springer, 1997.

[3] SUTTON R.S. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[4] HAYKIN S. *Neural Networks: A Comprehensive Foundation.* MacMillan College Publishing, 1994.

[5] THATHACHAR M.A.L. and PHANSALKER V.V. Convergence of teams and hierarchies of learning automata in connectionist systems. *IEEE Transactions on Systems Man and Cybernetics*, 25(11):1459–1469, Nov 1995.

[6] BARTO A.G. and ANANDEN P. Pattern-recognising stochastic learning automata. *IEEE Transactions on Systems Man and Cybernetics.*, 3:360–375, May/Jun 1985.

[7] BELLMAN R.E. *Dynamic Programming.* Princeton University Press, 1957.

[8] BERTSEKAS D.P. *Dynamic Programming and Stochastic Control.* Academic Press, 1976.

[9] DENARDO E.V. *Dynamic Programming: Models and Applications.* Prentice-Hall, 1982.

[10] HOWARD R.A. *Dynamic Programming and Markov Processes.* The Technology Press of MIT and John Wiley and Sons inc, 1960.

[11] BARTO A.G. BRADTKE S.J. and SINGH S.P. Learning to act using real-time dynamic programming. *Artificial Intelligence.*, 72:81–138, 1995.

[12] KOSKO B. *Neural Networks and Fuzzy Systems.* Prentice-Hall, 1992.

[13] BRADTKE S.J. Incremental dynamic programming for on-line adaptive optimal control. *CMPSCI Technical Report.*, pages 62–94, 1994.

[14] GULLAPALLI V. Learning control under extreme uncertainty. *Advances In Neural Information Processing Systems, IIANSON S.J. COWAN J.D. and GILES C.L. (Ed)*, 5, 1993.

[15] LIN C.J. and LIN C.T. Reinforcement learning for an art-based fuzzy adaptive learning control network. *IEEE Transactions on Neural Networks*, 7(3):709–731, May 1996.

[16] SUTTON R.S. and BARTO A.G. *Reinforcement Learning.* MIT Press, 1998.

[17] NARENDRA K.S and THATHACHAR M.A.L. *Learning Automata, an introduction.* Prentice Hall International Editions, 1989.

[18] LANCTOT J.K. and OOMMEN B.J. Discretised estimator learning automata. *IEEE Transactions on Systems Man and Cybernetics*, 2(6):1473–1483, Nov/Dec 1992.

[19] RAJARAMAN K. and SASTRY P.S. Finite time analysis of the pursuit algorithm for learning automata. *IEEE Transactions on Systems Man and Cybernetics - part B: Cybernetics*, 26(4):590–598, Aug 1996.

[20] BRADTKE S.J. and DUFF M.O. Reinforcement learning methods for continuous-time markov decision problems. *NIPS-94 Conference, Denver CO.*, Nov 1994.

[21] KAELBLING L.P. LITTMAN M.L. and MOORE A.W. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[22] SUTTON R.S. and BARTO A.G. Time derivative models of pavlovian reinforcement. *Learning and Computational Neuroscience: Foundations of Adaptive Networks, MIT Press, GABRIEL M. and MOORE J. (Ed)*, pages 497–537, 1990.

[23] RUMMERY G.A. Problem solving with reinforcement learning. Master's thesis, University of Cambridge, 1995.

[24] SUTTON R.S. (Ed). . *Machine Learning: Special Issue on Reinforcement Learning*, 8(3/4), May 1992.

[25] WATKINS C.J.C.H. and DAYAN P. Technical note q-learning. *Machine Learning*, 8:279–292, 1992.

[26] HUMPHRYS M. Action selection methods using reinforcement learning. Master's thesis, The University of Cambridge, Trinity Hall, 1996.

[27] THATHACHAR M.A.L. and SASTRY P.S. Learning optimal discriminant functions through a cooperative game of automata. *IEEE Transactions on Systems Man and Cybernetics*, 17(1):73–85, Jan/Feb 1987.

[28] WU Q.H. Reinforcement learning control using interconnected learning automata. *International Journal of Control*, 62(1):1–16, 1995.

[29] WU Q.H. PUGH A.C. Reinforcement learning control of unknown dynamic systems. *IEE Proceedings-D*, 140(5):313–322, Sept 1993.

[30] FROST G.P. HOWELL M.N. GORDON T.J. and WU Q.H. Dynamic vehicle roll control using reinforcement learning.

[31] HOBDAY S. WU Q.H. and GORDON T.J. A learning automata based fuzzy logic controller. *IFAC, 13th Triennial World Congress, San Francisco, USA*, pages 411–416, July 1996.

[32] GULLAPALLI V. A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, 3:671–692, 1990.

[33] NAJIM K. CHTOUROU and THIBAULT J. Neural network synthesis using learning automata. *Journal of Systems Engineering*, 2:192–197, 1992.

[34] BARTO A.G. SUTTON R.S. and ANDERSON C.W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems Man and Cybernetics.*, 13(5):834–846, Sept/Oct 1983.

[35] JERVIS T.T. and FALLSIDE F.F. Pole balancing on a real rig using reinforcement learning. Master's thesis, Cambridge University Engineering Dept, Cambridge, Dec 1992.

[36] TESAURO G. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3), Mar 1995.

[37] YEE R.C. SAXENA S. UTGOFF P.E. and BARTO A.G. Explaining temporal differences to create useful concepts for evaluating states. *AAAI-90 Proceedings eighth National Conference on Artificial Intelligence*, 2:882–888, 1990.

[38] BARTO A.G. SUTTON R.S. and WATKINS C.J.C.H. Learning and sequential decision making. *COINS Technical Report.*, pages 89–95, Sept 1989.

[39] SUTTON R.S. Generalisation in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems, MIT Press*, pages 1038–1044, 1996.

[40] THAM C.K. and PRAGER R.W. Reinforcement learning for multi-linked manipulator control. *from FTP site for downloading reinforcement learning papers, ftp://ftp.cs.umass.edu/pub/anw/pub/sutton/*, .

[41] LIN C.T. and KAN M.C. Adaptive fuzzy command acquisition with reinforcement learning. *IEEE Transactions on Fuzzy Systems*, 6(1):102–121, Feb 1998.

[42] BOYAN J.A. Modular neural networks for learning context-dependent game strategies. Master's thesis, University of Cambridge, 1992.

[43] CRITES R.H. and BARTO A.G. Improving elevator performance using reinforcement learning. *Advances in Neural Information Processing Systems, TOURETZKY D.S. MOZER M.C. and HASSELMO M.E. (Ed)*, 8, 1996.

[44] SINGH S.P. and BERTSEKAS D. Reinforcement learning for dynamic channel allocation in cellular telephone systems. *from FTP site for downloading reinforcement learning papers, ftp://ftp.cs.umass.edu/pub/anw/pub/sutton/.*

[45] KOENIG S. and SIMMONS R.G. Complexity analysis of real-time reinforcement learning applied to finding shortest paths in deterministic domains. Master's thesis, Carnegie Mellon University: School of Computer Science, 1992.

[46] DUFF M.O. Q-learning for bandit problems. *from FTP site for downloading reinforcement learning papers, ftp://ftp.cs.umass.edu/pub/anw/pub/sutton/.*

[47] JERVIS T.T. Connectionist adaptive control. Master's thesis, Trinity Hall, Cambridge, 1993.

[48] SINGH S.P. and SUTTON R.S. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.

[49] THAM C.K. and PRAGER R.W. A modular q-learning architecture for manipulator task decomposition. *from FTP site for downloading reinforcement learning papers, ftp://ftp.cs.umass.edu/pub/anw/pub/sutton/,* .

[50] CLAUSEN C. and WECHSLER H. Quad-q-learning. *IEEE Transactions on Neural Networks.*, 11(2):279–294, Mar 2000.

[51] PIATER J. Controlling traffic signals. *from FTP site for downloading reinforcement learning papers, ftp://ftp.cs.umass.edu/pub/anw/pub/sutton/.*

[52] SUTTON R.S. Planning by incremental dynamic programming. *Proceedings ninth Conference on Machine Learning*, pages 353–357, 1991.

[53] BOYAN J.A. and LITTMAN M.L. Packet routing in dynamically changing networks: A reinforcement learning approach. *from FTP site for downloading reinforcement learning papers, ftp://ftp.cs.umass.edu/pub/anw/pub/sutton/,* .

[54] KAELBLING L.P. (Ed). *Machine Learning: Special Issue on Reinforcement Learning*, 22(1/2/3), Jan/Feb/Mar 1996.

[55] SUTTON R.S. and SINGH S.P. On step-size and bias in temporal-difference learning. *Proceedings of the eighth Yale Workshop on Adaptive and Learning Systems*, pages 91–96, 1994.

[56] CHRISMAN L. Reinforcement learning with perceptual aliasing: The

perceptual distinctions approach. *Proceedings of the tenth National Conference on Artificial Intelligence, CA, AAAI Press*, pages 138–188.

[57] McCALLUM R.A. Hidden state and reinforcement learning with instance-based state identification. *IEEE Transactions on Systems Man and Cybernetics - part B: Cybernetics*, 26(3):464–473, Jun 1996.

[58] MOORE A.W. and ATKESON C.G. Prioritised sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130.

[59] SUTTON R.S. Integrated architectures for learning, planning and reacting based on approximating dynamic programming. *Proceedings of the seventh International Conference on Machine Learning*, pages 216–224, 1990.

[60] SUTTON R.S. Dyna, an integrated architecture for learning, planning and reacting. *Working Notes of the 1991 AAAI Spring Symposium*, pages 151–155, 1991.

[61] YU Y.N. *Electric Power System Dynamics.* Academic Press, Inc, 1983.

[62] HINGORANI N.G. Introducing custom power. *IEEE Spectrum*, pages 41–48, Jun 1995.

[63] HAMMAD A.E. Analysis of power system stability enhancement by static var compensators. *IEEE Transactions on Power Systems*, PWRS-1(4):222–227, Nov 1986.

[64] WANG H.F. and SWIFT F.J. A unified model for the analysis of facts devices in damping power system oscillations part ii: Multi-machine power systems.

[65] WANG H.F. and SWIFT F.J. A unified model for the analysis of facts devices in damping power system oscillations part i: Single-machine infinite-bus power systems.

[66] CRITES R.H. Large-scale dynamic optimisation using teams of reinforcement learning agents. Master's thesis, The University of Massachusetts Amherst, 1996.

[67] KOENIG S. and SIMMONS R.G. Unsupervised learning of probabilistic models for robot navigation. *Proceedings of the International Conference on Robotics and Automation (ICRA-96)*, 1996.

[68] MITCHELL T.M. and THRUN S.B. Explanation-based neural network learning for robot control. *Advances In Neural Information Processing Systems, MOODY J.E. HANSON S.J. and LIPMAN R.P. (Ed)*, 5, 1993.

[69] LI B.H. WU Q.H. WANG P.Y. and ZHOU X.X. Learning coordinated fuzzy logic control of dynamic quadrature boosters in multi-machine power systems. *Proc. IEE. Part C, Generation, Transmission and Distribution*, 146(6):577–585, Feb 1999.

[70] CHAN K.H. JIANG L. TILLOTSON P.R.J and WU Q.H. Reinforcement learning for the control of large-scale power systems. *Proceedings of the Second International Symposium Engineering of Intelligent Sys (EIS 2000)*, Jun 2000.

[71] CAO Y.J. WU Q.H. JIANG L. and CHENG S.H. Nonlinear control of power system multi-mode oscillations. *International Journal of Electrical Power & Energy Systems*, 20(1):61–68, 1998.

[72] CHAN K.H. JIANG L. TILLOTSON P.R.J and WU Q.H. Learning fuzzy logic control of synchronous generators in multi-machine power systems. *Proceedings of the 35th Universities' Power Engineering Conference (UPEC 2000)*, Jun 2000.

[73] CHAN K.H. JIANG L. TILLOTSON P.R.J and WU Q.H. Reinforcement

learning for fuzzy logic control of large-scale power systems. *IEE Control Seminar, Learning Systems for Control*, page 3/1 3/5, May 2000.

[74] HASSAN M.A.M. MALIK O.P. HOPE G.S. A fuzzy logic based stabilizer for a synchronous machine. *IEEE Transactions on Energy Conversion*, 6(3):407–413, Sept 1991.