Reinforcement Learning & Artificial Neural Networks

The Optimal Control of the Water Vessel Process

Sirous Kavehercy

May 1996

Abstract

In this master's thesis we are going to study the use of artificial neural networks in the control of a water vessel process. Water vessel process is highly nonlinear and conventional methods of control from control theory do not yet provide optimal solutions to the control of such processes. The ability of artificial neural networks to approximate nonlinear functions could therefore be effectively used in the optimal control of nonlinear dynamic systems, where the relationship between the state of the system and the control signals is mostly nonlinear and unknown.

As will be explained later, in this study a simple version of water vessel process is considered in which only one vessel is to be controlled. The task is to learn the function of the controller using artificial neural networks. Two types of neurocontrollers are discussed, linear and nonlinear. To learn the mappings of these neurocontrollers the method of reinforcement learning is used in combination with temporal difference learning. In addition to the experiments with neurocontrollers, also experiments were performed using conventional P- and PI-Controllers.

Acknowledgments

This research was performed in the Autonomous Systems Group at the University of Amsterdam headed by Prof. F.C.A. Groen. The idea of the study came from the real system of water vessels in Delft University of Technology, Department of Electrical Engineering where they experiment with other methods of keeping the water in the vessels. There have been lots of people involved in this research, either directly or indirectly and I would like to thank as many as I can remember.

I would like to thank my research supervisor dr. ir. Ben Kröse especially for his comments on the thesis. I also thank dr. ir. A.J.J. van den Boom for providing the information on the water vessel system. I want to thank Gerard Schram whose previous work on reinforcement learning [22] was my starting point. I found our talks helpful, although they were few.

I am grateful to Hakan (Huseyin Yakali) for being an enormous help throughout the whole research. Thanks for taking the time to explain basics of control theory to me, giving me matlab tips, discussing PC hardware and most of all for being a nice colleague. I am really thankful to Sander Bosman for our talks, giving me useful feedback on parts of the thesis and mainly for just being there. Good luck with your work, although you probably won't need it. I would also like to thank Albert (A.J.N. van Breemen) for good water vesselly discussions and being a good friend. I want to thank René Brohm for asking me to explain the goal of my research to him at the very beginning. He might not know it but that long discussion was quite useful for me. The following friends have been indirectly involved in my research and I want to thank them all: Ahmed, Antonios, Behnam, Drona, Fred, Harold, Kemal, Mehran and Mike.

The illustration of the water-vessel on the cover is copyright of Gary Cooper, used with his kind permission.

Most of all I want to express my gratitude to my parents Parvin and Iraj, and my sisters Parisa, Nadia, Nazy and Mahsa. They have always been there to support and encourage me. I owe them a lot. I want to thank my uncle Touraj for a great laptop and the printer. And last but not least, I want to thank Anke for being the greatest companion of all. She has shared the good and the bad times with me and could probably write her psychology master's thesis on *The Optimal Control of Multitempered Personalities Under Pressure.* Thanks for everything.

Sirous Kavehercy
Amsterdam, May 1996

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 General Introduction

A conceptual basis for **artificial neural networks** (also known as *parallel distributed processing*) was formed in the 1940s by the researchers who were studying the biological structure of the human brain. The idea emerged that the network of interconnected neurons in the brain could be simulated by electronic circuits able to perform simple computational tasks. The research in this field resulted in building simple artificial neural networks called *perceptrons* in late 1950s (Rosenblatt [20]). However, near the end of 1960s the limitations on the performance of perceptrons on one hand (Minsky & Papert [17]) and the lack of proper technology on the other, caused the research to slow down in this field. Little progress was made until 1980s, when the availability of powerful microprocessors and important theoretical breakthroughs opened new doors for artificial neural networks. Since then this field has been the subject of a great deal of research which has extended the applicability of neural networks to different classes of learning problems.

An artificial neural network could be classified as a computational model of the human brain. However, in contrast with the way a computer is "programmed" to perform a certain task, an important property of an artificial neural network is that it can "learn" and "adapt" itself. In an artificial neural network the nodes represent the neurons in the brain and the arcs (called *weights*) between them the synapses between the neurons. The computation in each node takes place as a response to its inputs. This resembles the reaction of the neurons of the brain to electrochemical signals sent to them from other neurons through the synapses. The learning involves adjusting the weights of the neural network as a response to the data presented to it. For a detailed discussion of different designs and learning methods of neural networks the following books are good sources: Haykin [12]; Hertz, Krogh & Palmer [13]; Rumelhart & McClelland [21].

Some of the well-known characteristics of artificial neural networks are their ability to learn mappings based on input-output examples and to generalize upon them. Robustness and fault-tolerance are also some other useful properties of neural networks. As outlined in Rumelhart & McClelland [21] artificial neural networks are suitable for learning problems involving pattern association, auto association and classification. Despite successful applications of artificial neural networks to different learning tasks it should be added that no matter how "intelligent" the behaviour of neural networks might seem, they are still far from being a complete model of the brain.

In recent years there has been a growing interest in neural networks by the people of control community. The ability of neural networks to approximate nonlinear functions has made them an effective tool in optimal control of nonlinear dynamic systems. The conventional methods do not have solutions to the optimal control of nonlinear systems in a well-understood manner as they do for linear systems. The optimal control tasks involving nonlinear dynamic systems are usually faced with two problems:

- Unavailability of a proper system model due to the highly nonlinear and complex dynamics

of the system.

- The mostly time-varying relationship between control signals and the dynamics of the process.

Artificial neural networks could be used as means of modeling the nonlinear system or/and the nonlinear controller which maps the dynamics of the process to control signals. The term *neurocontroller* is used to refer to a neural network based controller used in a control process. The research has shown that neurocontrollers are able to perform well in the control of an unknown process under noise and are adaptive to dynamical changes in the system. Sofge and White discuss several types of these neurocontrollers in their excellent book [32].

1.2 Water Vessel Process

An example of a nonlinear dynamic process is the so called *water vessel process*. The water vessel process consists of four vessels connected with each other as shown in Figure 1.1. The water

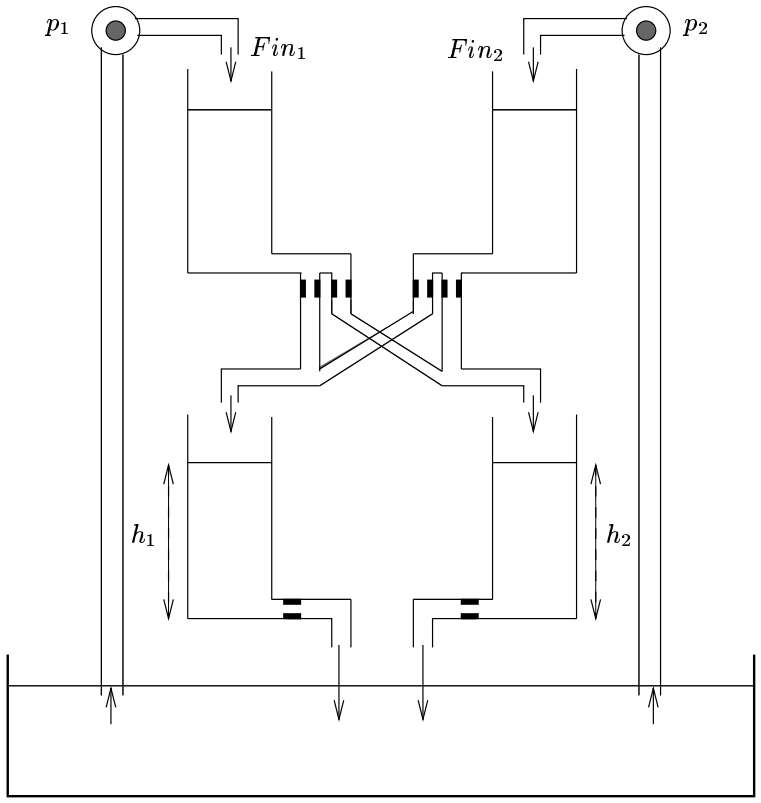

Figure 1.1: Dual cascaded water vessel system.

from a water reservoir is pumped into the top vessels by roller pumps p_1 and p_2. The goal is to control the level of water h_1 and h_2 in the lower vessels by controlling the inflow Fin_1 and Fin_2 to each of the top vessels produced by the pumps. The output restrictions of all the four vessels remain constant during the process. The water vessel process is a highly nonlinear and dynamic process. There are different aspects of this process which make it interesting as a control problem, to mention:

1. Nonlinearity in the process.
2. Nonlinearity due to the pump flow range, since it has a maximum and minimum flow capacity.

3. Variable delay caused by the transfer time of water to each of the top or lower vessels.

In this thesis we will be dealing with a simplified version of the water vessel process consisting only of a single vessel as shown in Figure 1.2.

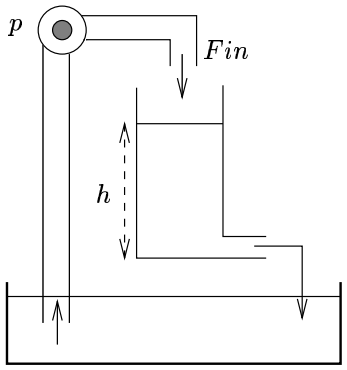

Figure 1.2: Single water vessel system.

Boom [7] discusses a simple nonlinear approximation of the water vessel process which is used in our experiments and will be explained in Section 4.2.

The goal is to design a neurocontroller that learns to control the level of water in the vessel without using a model of the process and just by gaining experience over time through interaction with the environment. This method is called *reinforcement learning* (RL for short). The power of RL lies in its ability to produce an estimate of the future performance which adaptively improves with time. At each time-step a control signal is produced which is expected to result in the improvement of the performance.

1.3 Overview

In Chapter 2 we give a short introduction on control theory and some control laws. In Chapter 3 we discuss reinforcement learning and some of its methods. In the same chapter we will discuss the CMAC neural networks which are used within the RL-Controller. In Chapter 4 the experimental setup will be explained and the results of the experiments with different conventional controllers will be presented and compared to a linear RL-Controller. In Chapter 5 the results of the experiments with a nonlinear RL-Controller are presented. And finally in Chapter 6 we conclude the research and describe possible future work.

Chapter 2

Control Theory

2.1 Introduction

The purpose of this chapter is to provide some basic knowledge of control theory. It starts with the definitions of some elementary terms from control theory. Then in Section 2.7 some control laws are presented and discussed.

The contents of this chapter are mainly based on the material from a number of books. These are: Amerongen et al. [10], Narendra & Annaswamy [19], Leigh [15], Burghes & Graham [5] and Dorst [8]. Of course, explicit references will be provided in the text when necessary.

2.2 System

A system is defined as an aggregation of objects united by some kind of interaction or interdependence. When one or more aspects of the system change with time, it is generally called a *dynamical* system [19]. To analyze a system, we should first make sure what its relevant inputs and outputs are and how they are related to each other. The behaviour of a dynamic system is usually characterized by differential equations that are derived from the properties of the components involved and their interactions. We will talk more about these differential equations and the dynamics of the system in Section 2.5.

2.3 Control Systems

A *control system* is defined as a system consisting of at least a controller and a process, in which a certain goal is to be achieved. The process is the part that needs to be controlled. By control of a process, we mean trying to keep its relevant output within some desired limits. In a very simple form the system would look like the one in Fig.2.1. The controller uses the value of $y(t)_{desired}$ and produces a control signal $u(t)$ for the process. The goal is to try to drive the process such that it's output $y(t)$ equals $y(t)_{desired}$. The controller uses some *a priori* knowledge to determine the control signal and is not affected by the actual output of the process. This is called *driving* the process.

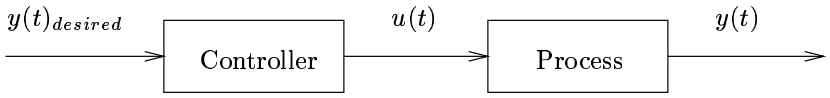

Figure 2.1: A control system which drives the process (also known as open-loop).

2.3.1 Feedforward Control

Driving is one way of 'control' which performs well when the process specifications are known prior to control and there is no noise in the system. Unavailability of the process specifications and the presence of noise could cause the process to produce undesirable output. In situations where the noise is known we could improve the performance by using a different form of driving called *feedforward* control. Suppose that noise $n(t)$ is measured at time t then Fig.2.2 shows how a feedforward controller might be used to get a better result.

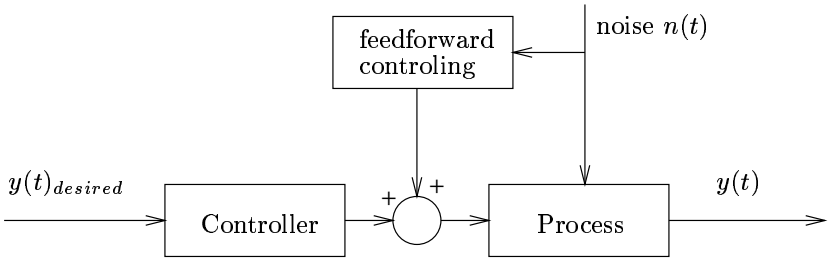

Figure 2.2: A feedforward controller

In this case we actually use the measured noise to influence the output of the process. However, just knowing the amount of the noise signal is usually not enough. In order to get satisfactory results by using feedforward controlling it is necessary to know exactly how the system is influenced by the noise. Therefore, dealing with a large number of noise signals requires also a large number of sensors to measure their influences on the system. This makes the use of feedforward control costly. Besides, it is not always easy to determine the amount and the effect of the noise precise enough for the controlling purposes.

2.3.2 Feedback Control

One way of getting around the problem of noise is to use the principle of *feedback* control. A simple feedback control system is shown in Fig.2.3 assuming that the noise is included in the process. Using feedback principle we make the system less sensitive to noise or changes in other parameters of the system. Here the error in the performance is calculated using the actual output of the process and its desired output also called *setpoint*. This error is passed to the controller which then produces the control signal. Feedback control systems are also called *closed-loop* as opposed to *open-loop* control systems shown in Fig.2.1.

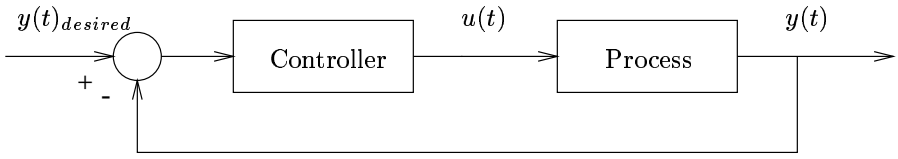

Figure 2.3: A feedback controller (also known as closed-loop)

2.4 Regulators and Servosystems

It was mentioned in 2.3 that the aim of control is to keep the relevant output of the process within certain limits. If the setpoint (desired output value of the process) is constant the problem is one of *regulation* of the system around this setpoint. But if the setpoint is time-varying then we refer to the problem as *tracking*. In the former case, the system is referred to as *regulator system* and in the latter *servosystem*.

In servosytems, besides coming to rest at the setpoint it is also very important how that has been achieved. In other words, how fast a new reference point is reached and how much *overshoot* was involved during the transition. We will talk more about these aspects in Section 2.7 when we discuss different control laws.

2.5 Linearity and Nonlinearity

Before defining the linearity of a system let's see when a function is called linear. A function f is said to be linear if the following conditions hold [15]:

1. Homogeneity Condition:

$$f(\alpha x) = \alpha f(x) \qquad \text{for any } x \text{ in the domain of } f \text{ and for any } \alpha; \tag{2.1}$$

2. Additivity Condition:

$$f(x_1 + x_2) = f(x_1) + f(x_2) \qquad \text{for any } x_1,\, x_2 \text{ in the domain of } f. \tag{2.2}$$

It should be clear that the same conditions could also define the linearity of differential equations and thus the systems they represent. In that case we say that the system obeys *superposition principles* (referring to conditions (2.1) and (2.2)).

Next, two examples of processes represented by linear differential equations are given.

- Radioactive Decay (1^{st} order): The number of radioactive particles that are deteriorated per time-step is proportional to the number of already existing particles: $\frac{\Delta N}{\Delta t} = -\lambda N$ $(\lambda > 0)$ where N is the number of particles, t the time and λ the constant of proportionality. The differential equation representing this decay process is therefore a linear first order equation given as:

$$N' + \lambda N = 0 \qquad (\lambda > 0) \tag{2.3}$$

  Now $N(t)$ could be easily calculated for the process.

- Object-Spring (2^{nd} order): Another example of a linear system is an object hanging from a spring moving only in vertical direction. Call the vertical axis as x with positive direction downwards and its origin 0 at balanced position of the object. Figure 2.4 shows how this system might look like and:

$$Mx'' + fx' + kx = 0 \qquad (M > 0, f > 0, k > 0). \tag{2.4}$$

  is its differential equation which is linear and of the second order, where M represents the mass of the object, f the friction with air and k the constant of the spring. We can easily calculate $x(t)$ from Equation(2.4) using known methods and use it for control purposes.

If the differential equation of a system is not linear the system is said to be nonlinear, and the superposition principles do not apply to it. Linear systems are the ones that are best understood and used in control theory. For nonlinear systems superposition is invalid by definition and that precludes the direct application of well known linear principles for their control. While the same type of controllers could be designed for nonlinear systems, no coherent theory of control design exists at present for any reasonable class of nonlinear systems.

The nonlinear differential equation of the single water vessel system when linearized is reduced to a first order equation as shown by Boom [7].

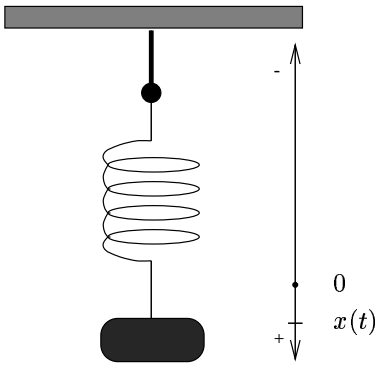

Figure 2.4: A system containing an object hanging from a spring.

2.6 Adaptive Systems

Many of the real-world processes are of a nonlinear nature. For example in most of the mechanical devices this is caused by highly nonlinear friction forces. The exact parameters of these processes are usually not known to us. The behaviour of the process changes with time due to numerous environmental causes which are not always as easy to measure or detect. Conventional control theory even when used to design controllers for such systems, may be inadequate for achieving satisfactory performance of the system for its whole range of parameters' values. Adaptive control refers to the control of this partially known systems. The term *adaptive system* was introduced by Drenick & Shahbender [9] in control theory to represent control systems that monitor their own performance and adjust their parameters in the direction of better performance. The following definition of an adaptive system is taken from Eveleigh [11]:

Adaptive System is a system which is provided with a means of continuously monitoring its own performance in relation to a given figure of merit or optimal condition and means of modifying its own parameters by a closed-loop action so as to approach this optimum.

2.7 Control Laws

The *controller* or *control law* contains the information about how the control processor produces the control signals from its input signals. In the following sections some well-known control laws are presented. These control laws have been primarily developed for controlling linear systems. They could also be used to control nonlinear systems, in which case the system is linearized first. As we mentioned earlier, the direct application to nonlinear systems is not well-studied and might not produce satisfactory results.

In the followings, y_d and y, represent, respectively the setpoint and the current output value of the plant to be controlled and u is the control signal generated by the controller.

2.7.1 P-Controller

A P-Controller generates a control signal which is *proportional* to the amount of error. Such a controller is called *proportional controller*, or P-Controller for short. The control law will be as follows:

$$\mathrm{P}: \quad u = K_p * (y_d - y) \tag{2.5}$$

where K_p is the proportional gain.

When dealing with linear systems of the second order it is possible to control the behaviour of the system by selecting appropriate values for K_p. Without going into the detail of how it is done I will just mention the three possible cases:

- *Damping*
  A small K_p results in *damping* in which the y_d is reached asymptotically and very slow.

- *Critical Damping*
  There is a value of K_p for which y_d is reached at the shortest possible time, without overshooting the value.

- *Overshoot*
  For large values of K_p the system gets rapidly close to the desired value, but oscillates around it. The relative amplitude of the first oscillation is often called the overshoot.

Note that the value of K_p only controls whether or not the system is critically damped, but the amount of damping is not affected by that and is always equal to a *damping constant*. One major drawback of the P-Controllers is that they do not necessarily result in a zero error state. The reason for this is that naturally when the error is zero the proportional control action will be also zero which is not always desired. We will see how this problem could be solved when we discuss PI-Controllers in Section 2.7.3.

2.7.2 PD-Controller

In P-Controller case, we could not control the damping. By adding a term to the controller proportional to $\dot{y}$ (the derivative of y) we will be able to control the damping. Such a controller is called *proportional plus derivative* controller, or PD-Controller for short. This new controller will generate the control signal as follows:

$$\text{PD}: \quad u = K_p * (y_d - y) - K_d * \dot{y} \tag{2.6}$$

where K_d is called the *differential gain*. Now: if the error is large and the speed at which the desired value is being reached is slow, the control signal will be high, so the system accelerates; if the error is small and the speed is high, the control signal will be negative, causing the system to decelerate. This way the system rapidly homes in on the desired value.

2.7.3 PI-Controller

A problem with P-Controllers is that the asymptotic value (steady state) of the system is not necessarily equal to the desired value. This is called the *steady-state error* problem. Since the problem involves the accumulation of the error we can overcome this by designing a new controller which generates a control signal dependent also on the integral of the error. This is called *proportional plus integral* controller, or PI-Controller for short. Its feedback law to generate the control signal is:

$$\text{PI}: \quad u = K_p * (y_d - y) + K_i * \int_0^t (y_d - y(\tau))d\tau \tag{2.7}$$

where K_i is the gain for the integral factor and t the current time of the system. Now if the P-part of the controller causes a steady state error making $y > y_d$ then the I-part is negative and generates a control signal which decreases the error to 0.

2.7.4 PID-Controller

If we combine the properties of PI and PD controllers we will get *proportional plus integral plus derivative* controller, PID-Controller for short. This controller would generate the control signal as follows:

$$\text{PID}: \quad u = K_p * (y_d - y) - K_d * \dot{y} + K_i * \int_0^t (y_d - y(\tau))d\tau \tag{2.8}$$

Chapter 3

Reinforcement Learning

3.1 Introduction

In this chapter the method of reinforcement learning is discussed. We first give a short history on the psychological background and then go on to explain the characteristics of this type of learning in Section 3.3. In Sections 3.4 and 3.5 we explain how reinforcement learning could be implemented by discussing some popular approaches. In Section 3.7 finally the CMAC function approximator is discussed in some detail due to its importance for this project.

This chapter is by no means a complete introduction to all different methods of reinforcement learning. The main objective is to give an overview of this learning paradigm and discuss some of its major designs.

3.2 Animal Learning

In the context of experimental studies related to animal learning the terms reinforcement and punishment have been used to refer to the events that could affect the stimulus-response relationships and therefore learning. To see what this means it might be helpful to recall the Thorndikes's "Law of Effect" from 1911 [25]:

> Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond.

Although later results convinced Thorndike that punishment is rather ineffective in weakening responses but he strongly believed that reinforcement is critical to learning. Despite all the controversies, this theory has been one of the most influential learning theories in psychology because of its common-sense approach.

The term *Reinforcement Learning* however, is never used in psychology. We could trace its usage back to Minsky [18], Waltz and Fu [26] in their studies of learning in the fields of AI and control theory, inspired by animal learning. In the rest of this chapter this approach to learning will be explained in relation with artificial neural networks. For further discussions on reinforcement learning, the following material could be used: Barto [3], [2] and Williams [33].

3.3 Reinforcement Learning

Supervised learning, *learning with a teacher*, is probably the most common learning method used in the field of artificial neural networks. In this approach a certain input-output mapping is learned by some examples which are presented to the network by the teacher. The method of reinforcement learning could be best explained on the basis of its difference with supervised learning. We will not however, get into the details and formulations since that will be done in the next section. This is just to give you an idea of what kind of problems reinforcement learning has to deal with.

It might be helpful to use an analogy to make the idea of reinforcement learning more understandable. Consider a learning task in which a student is supposed to improve his grade in an exam. In a supervised learning method the teacher would give the student the information on his/her grade plus the feedback on each of the questions, also the correct answers. The student then uses this information to improve his/her performance in the corresponding areas . On the other hand, in a reinforcement learning method the teacher would provide *only* the grading information. Now the student is faced with the problem of finding out his/her mistakes and the correct solutions solely based on a single grade, evaluating the overall performance in each of the consecutive exams.

Usually, a system is said to be learning when it improves its performance based on a certain performance measure. Suppose that the performance measure is calculated as a function of the parameters of the learning system which represent its current state. For instance, in water vessel process these parameters could be e the error between the current and the destination height, h_d the destination height, u the inflow signal sent to the pump and maybe some other sensory information. If we visualize the performance measure as a surface, we can assign to each state of the system $(e, h_d, u, \ldots)$ a point on that surface $f(e, h_d, u, \ldots)$ where f is the performance-measure function. Now if the system is to improve its performance, the point corresponding to its state on the performance surface should move towards higher points [3].

The difference between learning tasks is in the type of information available to the system about the characteristics of this performance surface. For example in the tasks involving supervised learning the learning system has the gradient information of its performance surface at the point corresponding to the current state of the system. So the supervised learning system receives information on how it should adapt itself in order to improve its performance. Notice that the term "gradient" is used because it makes better sense in combination with performance "surface", but in general the information feedback to the supervised learning system does not necessarily have to be of a gradient nature. The key point is that the existence of (input,target-output) examples provided by the teacher plays a crucial role in supervised learning.

However, the tasks involving reinforcement learning are faced with a problem concerning the evaluation of their state. The only feedback the system receives from its environment is the value of the performance measure corresponding to the current state of the system. This scalar value is usually called *(primary) reinforcement signal*. It should be mentioned that this reinforcement signal could also be a vector of values, for example in the reinforcement learning tasks concerned with more than one performance measure. But in both cases the problem remains that the reinforcement signal evaluates the performance but does not on its own provide the learning system with the direct-information on how it should change itself to improve its performance. Therefore, the main task of the reinforcement learning system is to somehow produce an estimate of the gradient using the changes in reinforcement during learning.

There are two main types of reinforcement learning tasks as Sutton [23] distinguishes:

- *Nonassociative RL*, in which the task of the learning system is to find a single optimal action with reinforcement signal being the only piece of information the learning system receives from the environment and no other sensory information are available to it. An example would be a robot trapped in a room trying to free itself by performing a certain action not known to it. The only input to the robot is a reinforcement signal 0 or 1 representing the door being closed or open. This signal is presented to the robot after performing an action.

- *Associative RL*, in which the task of the system is to associate different actions to different

system states. In this case the system receives some additional sensory information from the environment regarding the state of the system. Associative reinforcement learning is closer to Thorndike's idea and has been the subject of a lot of research in the field of artificial neural networks.

We will be dealing mainly with the associative reinforcement learning in this chapter since it is also closely related to the optimal control theory.

Further, it is also useful to divide RL-tasks into two more classes. One in which the reinforcement signal is present right after the system has performed an action. Another in which the reinforcement signal is delayed because it depends on a series of actions performed by the system. The latter usually occurs when the task involves the optimal control of dynamic systems. In the rest of the chapter we will be concerned with the delayed RL.

In application to the optimal control problems RL could be formulated in a way that the long-term consequences of actions are taken into account since in most of the cases the goal is to design a controller with an optimal long-term performance. The RL-Controller is then designed to receive a reinforcement signal from the controlled process based on its performed action and the state of the process. The objective of the learning system is then to either minimize or maximize the amount of reinforcement signals accumulated in the future, depending on what the signal represents, cost or benefit. This performance measure is often calculated as a discounted sum of the future reinforcement signals in which the earlier ones are weighed more.

In the following sections we will describe how this is done. To relate the discussion to our experiments we will assume that the reinforcement signal represents a certain cost(in contrast to benefit). And of course we assume that no model of the process is available and the learning occurs real-time without it.

The material used in the following two sections are based on the works of Sathiya & Ravindran [14], Sutton [23][24] and Watkins [27]. References will be provided explicitly where necessary.

3.4 Evaluation Function

Recall that the goal of reinforcement learning is to find a policy for selecting actions in a way that the selected sequence of actions will be optimal according to a certain performance measure. Now we are going to formulate these ideas.

Consider the RL-Controller is used to optimize the control of a discrete-time dynamical system. Let $x(t)$ represent the system state at time t and $u(x(t))$ the selection policy for actions based on the state of the system. We will be only concerned with a *stationary* policy meaning that the actions are selected based on the current state of the system and not the previous ones. In other words we assume that the transition probability of the current state x to y in the next time-step is only dependent on the current state and action.

Suppose the system starts at $t = 0$. Then $q(x(t), u(x(t)))$ represents the cost (reinforcement) received by the learning system after performing action $u(x(t))$ at state x. We will take the performance measure to be the discounted sum of the future costs which should be minimized for the system to perform optimal. From now on we call the function representing this performance measure the *evaluation function* and define it for a given policy u as:

$$J^u(x) = E\left[\sum_{t=0}^{\infty} \gamma^t q(x(t), u(x(t)))|x(0) = x\right] \tag{3.1}$$

The summation term is called the *cumulative discounted cost*. E is the expectation operator and γ the discount factor that can take a value in $0 \leq \gamma < 1$, typically 0.95. $\gamma = 1$ is not used because of the problems it might cause with the convergence of the sum. The value of γ represents to which extent the learning system is concerned with the consequences of the control actions.

The optimal evaluation function J^* could be calculated by:

$$J^*(x) = \min_u J^u(x) \quad \forall x \tag{3.2}$$

In order to calculate J^*, we first need a method of approximating J^u. Since the actual outputs of the evaluation function involve future data not available to the learning system immediately, we require learning methods specialized to deal with delayed rewards. We discuss these next.

3.4.1 Approximating Evaluation Function

The goal is to find an approximation of $J^u(.)$ in (3.1). We call this approximation $\hat{J}(.;W)$, where W is the parameter set of the function approximator. A good approximation of $J^u(.)$ is important since it could be used to check the optimality of the policy u and if necessary adapt it to get a better policy, as we will see later. In this section we are going to discuss three methods of approximating the evaluation function.

We take τ to represents the number of time-steps elapsed after the system was in state x. For the clarity of writing we use $q(t)$ instead of $q(x(t), u(x(t)))$.

N-step Truncated Return

We define the following:

$$J^u_{[n]}(x) = \sum_{\tau=0}^{n-1} \gamma^\tau q(\tau) \tag{3.3}$$

The summation term is the n-step truncation of the sum in Eq. 3.1. Now the approximation $\hat{J}$ of J^u could be calculated by

$$\hat{J}(x;W) := E(J^u_{[n]}(x)) \quad \forall x \tag{3.4}$$

where E is the expectation operator. In cases where a system model is not present or the calculation of the expectation is costly we could simply use

$$\hat{J}(x;W) := J^u_{[n]}(x) \tag{3.5}$$

where $J^u_{[n]}(x)$ is calculated using real-time data over n time-steps. However, notice that the expectation in (3.4) covers all possible states that could be reached in n time-steps and the method used in (3.5) is restricted to only a specific set of states visited in on-line experiments. Therefore, $\hat{J}(x;W)$ in (3.5) will not be a good approximation of (3.4) unless it is averaged over a large number of trials. One way of achieving this averaging is to use the following learning rule based on the supervised-learning update rule

$$\hat{J}(x;W) := \hat{J}(x;W) + \alpha[J^u_{[n]}(x) - \hat{J}(x;W)] \tag{3.6}$$

with α being the learning rate.

Corrected N-step Truncated Return

A variation of the method mentioned above is to add a correction term to $J^u_{[n]}(x)$ to make up for the terms left out due to truncation.

$$J^u_{(n)}(x) = \sum_{\tau=0}^{n-1} \gamma^\tau q(\tau) + \gamma^n \hat{J}(x(n);W) \tag{3.7}$$

where $x(n)$ represents the state of the system after it has passed state x, n time-steps. Also notice that the rest-term added to the sum for correction involves the approximator $\hat{J}$ that we actually are aiming to find. $\hat{J}$ is always available. Although it might not be a good approximation for J^u at the beginning, as learning progresses this approximation improves. $J^u_{(n)}$ is considered to be more appropriate than $J^u_{[n]}$ and this could be justified by looking at the following two cases:

- $\hat{J}$ is a good approximation of J^u. Then $J^u_{(n)}$ will be an ideal choice to use with a small n.

- $\hat{J}$ is not a good approximation of J^u. Then we have to use a large n in both $J^u_{(n)}$ and $J^u_{[n]}$ to get good results. When n is large the rest-term $\gamma^n \hat{J}$ in (3.7) is negligible meaning that both approaches will produce the same results.

Ideally, the learning rule would be:

$$\hat{J}(x;W) := E(J^u_{(n)}(x)) \quad \forall x \tag{3.8}$$

Following the same line of arguments used in Section 3.4.1, for obtaining the approximation in (3.8) based on averaging the value of $J^u_{(n)}(x)$ over a number of trials, we come to the following on-line learning rule:

$$\hat{J}(x;W) := \hat{J}(x;W) + \alpha[J^u_{(n)}(x) - \hat{J}(x;W)] \tag{3.9}$$

with α being the learning rate.

However, the question is how large n used in (3.9) should be. If $\hat{J}$ is a good approximation of J^u then a small n would be fine. If that's not the case then a large n must be used. Therefore, for better performance using this method the value of n has to be changed during learning based on how well $\hat{J}$ is learned.

Temporal Difference Learning

Sutton [24] suggested the popular learning method called *temporal difference* especially applicable to learning of delayed rewards. This method uses an estimate of the sum in Eq. 3.1 based on geometrically averaging $J^u_{(n)}(x)$ from Section 3.4.1. We define J^u_λ as follows:

$$J^u_\lambda(x) = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} J^u_{(n)}(x) \tag{3.10}$$

with $(1-\lambda)$ being a normalizing term and $0 \leq \lambda \leq 1$. Notice that the $J^u_{(n)}$ terms with smaller values for n are weighed more in the averaging process. This makes sense since the terms with a large n rely more heavily on future data and therefore should be weighed less in the average. To make the effect of λ more clear let's expand (3.10) using (3.7):

$$\begin{aligned} J^u_\lambda(x) &= (1-\lambda)\big[J^u_{(1)}(x) + \lambda J^u_{(2)}(x) + \lambda^2 J^u_{(3)}(x) + \ldots\big] \\ &= q(0) + \gamma(1-\lambda)\hat{J}(x(1);W) + \\ &\quad \gamma\lambda\big[q(1) + \gamma(1-\lambda)\hat{J}(x(2);W) + \\ &\quad\quad \gamma\lambda\big[q(2) + \gamma(1-\lambda)\hat{J}(x(3);W) + \\ &\quad\quad\quad \ldots \end{aligned} \tag{3.11}$$

Since $q(0) = q(x, u(x))$ we could rewrite (3.11) recursively as:

$$J^u_\lambda(x) = q(x,u(x)) + \gamma(1-\lambda)\hat{J}(x(1);W) + \gamma\lambda J^u_\lambda(x(1)) \tag{3.12}$$

with $x(1)$ being the system state one time-step after x. Using (3.12) we get:

$$\begin{aligned} \lambda = 0 \quad &\rightarrow \quad J^u_0(x) = q(x,u(x)) + \gamma\hat{J}(x(1);W) = J^u_{(1)}(x) \\ \lambda = 1 \quad &\rightarrow \quad J^u_1(x) = q(x,u(x)) + \gamma J^u_1(x(1)) = J^u_{(\infty)}(x) \end{aligned}$$

In other words, in order to calculate the discounted sum J^u, J^u_0 makes use of the immediate cost within one time-step plus the *approximation* of the rest of the sum. J^u_1 on the other hand, relies only on the *actual* costs to achieve the same goal. At the end of this section we will come back to this discussion once more.

The learning method using J^u_λ is called $TD(\lambda)$, with TD being the short form for temporal difference. In a while we will see the justification for the use of this name.

Let's first define the learning rule similar to the one in (3.9), now using J^u_λ:

$$\hat{J}(x;W) := \hat{J}(x;W) + \alpha[J^u_\lambda(x) - \hat{J}(x;W)] \tag{3.13}$$

To be able to use this learning rule we need a way of calculating $J^u_\lambda(x)$ on-line, without requiring a system model or costly operations for computing $J^u_\lambda(x)$. Next we explain a solution to this problem which makes the on-line learning possible.

As we saw in (3.1), the evaluation function is defined as the discounted sum of the future costs. We could easily derive the relation between two consecutive evaluations as:

$$J^u(x) = q(x, u(x)) + \gamma J^u(x(1)) \tag{3.14}$$

with $x(1)$ being the system state one time-step after x. But, the same relation should also hold for the predictions of $\hat{J}$ if it is a good approximation of J^u. If that's not the case then the difference between these predictions could be used to adapt $\hat{J}$. Now, we define $\varepsilon(.)$ the *temporal difference* between two successive predictions of the evaluation function:

$$\varepsilon(x) = \big[q(x, u(x)) + \gamma\hat{J}(x(1);W)\big] - \hat{J}(x;W) \tag{3.15}$$

$\varepsilon(.)$ could be calculated using the temporal sequence of data available in each time-step. This also justifies why this method is called temporal difference learning. Further expansion of (3.11) results in:

$$\begin{aligned} J^u_\lambda(x) &= q(0) + \gamma\hat{J}(x(1);W) - \gamma\lambda\hat{J}(x(1);W) + \\ &\quad \gamma\lambda\big[q(1) + \gamma\hat{J}(x(2);W) - \gamma\lambda\hat{J}(x(2);W) + \\ &\quad\quad \gamma\lambda\big[q(2) + \gamma\hat{J}(x(3);W) - \gamma\lambda\hat{J}(x(3);W) + \\ &\quad\quad\quad \dots \\ &= q(0) + \gamma\hat{J}(x(1);W) + \\ &\quad (\gamma\lambda)\big[q(1) + \gamma\hat{J}(x(2);W) - \hat{J}(x(1);W)\big] + \\ &\quad\quad (\gamma\lambda)^2\big[q(2) + \gamma\hat{J}(x(3);W) - \hat{J}(x(2);W)\big] + \\ &\quad\quad\quad \dots \end{aligned} \tag{3.16}$$

Now we add $-\hat{J}(x;W)$ to both sides and use (3.15) knowing that $q(0) = q(x, u(x))$ to get the following error term:

$$J^u_\lambda(x) - \hat{J}(x;W) = \varepsilon(x) + (\gamma\lambda)\varepsilon(x(1)) + (\gamma\lambda)^2\varepsilon(x(2)) + \dots \tag{3.17}$$

Eq.(3.17) defines the error to be used in the learning rule (3.13) as a weighted sum of the temporal differences computed at each of the visited states. Temporal differences are weighed exponentially with the earlier ones weighed more. Still, we could not use this value in the learning rule (3.13) since the calculation of all the terms except the first one involves data only available in the future. There are different ways to deal with this problem, two of which will be mentioned here.

1. In (3.17) we could truncate the terms on the right hand side to select only the first N terms. This means that we calculate each term as the required information becomes available. The function approximator $\hat{J}$ stays unchanged for N time-steps till the required error is accumulated, after which it will be used to update $\hat{J}$ using (3.13).

2. Another approximation is to select $\lambda = 0$ and therefore it will be called TD(0). The error term in (3.17) is then reduced to its first term. Now we could update $\hat{J}$ at each step using the temporal difference in (3.15). This is an extreme case of TD(λ) learning in which we continuously adapt $\hat{J}$ at each time-step using the error in consecutive predictions.

It is useful to mention another extreme case in which $\lambda = 1$. As Sutton [24] shows, in this case the adaptation is simply an equivalent of supervised learning method which uses the difference

between the predictions of $\hat{J}$ and the actual outcome of the evaluation function J^u. This will make more sense if we assume that after a finite number of time-steps, the performed actions result in the final state, and a reinforcement signal is provided by the environment. This reinforcement is then the actual outcome of the evaluation function and could be used for adaptation. Taking the infinite discounted sum of future costs as the outcome of the evaluation function is then a generalization of this case, in which any intermediate state also produces a certain reinforcement signal $q(.,.)$.

The truncation method when combined with a proper λ speeds up the learning procedure since less updates take place. On the other hand, TD(0) slows down learning due to its updates per time-step, but because of its simplicity it is widely used and could perform well by choosing an appropriate α. However, choosing an appropriate λ instead of simply $\lambda = 0$ or $\lambda = 1$ could improve the computational efficiency a great deal (Sutton [24]). One method is to start with $\lambda = 1$ and decrease it to 0 as $\hat{J}$ improves its approximation.

The advantage of temporal-difference methods of learning lies in their sensitivity to temporal changes between successive predictions of the evaluation function and not so much the actual error between the predictions and the target-output. This is the reason why they are so well applicable to delayed-RL tasks involving prediction problems.

Now we are going to discuss some popular designs of RL-Controllers.

3.5 Reinforcement Learning Methods

3.5.1 Adaptive Critic Method (discrete)

Adaptive critic was an early method of reinforcement learning presented by Barto, Sutton and Anderson [4] which they used in their classic task of pole-balancing.

The adaptive critic learning systems contain two parts. The 'critic' which has the task of predicting future costs. And the 'actor' which produces actions at each state x. It is assumed that the state and action space are both finite, hence the term discrete in the title. Critic receives the reinforcement signal and gives an estimate of the future cost, at the same time it adapts the actor policy of selecting action for better performance next time. Figure(3.1) shows an adaptive critic based reinforcement-learning control system. The critic is adapted by the method of TD(0) explained earlier.

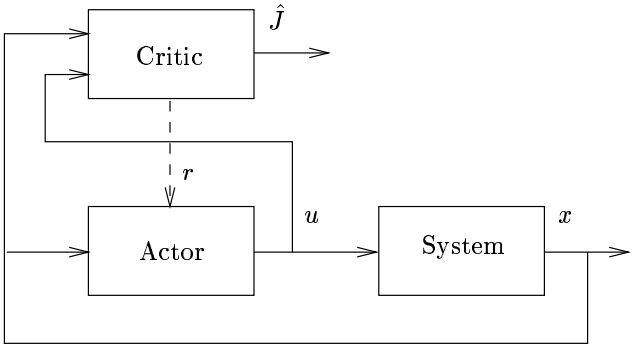

Figure 3.1: Adaptive-critic reinforcement learning scheme. x represents the state of the system, J the prediction of the future costs and u the selected action for the current state. Critic produces the reinforcement signal r for the actor to adapt its policy.

Suppose that the action space contains m actions which we represent by a_1, a_2,... a_m. Let $\hat{u}(.;V) : X \rightarrow \mathcal{R}^m$ be the approximation of the actor's selection policy $u(.)$, with X being the action space and V the parameters of the approximator. $z = \hat{u}(x;V)$ gives a vector of merits of the feasible m actions at state x. Assume that z_k denotes the k-th component of z corresponding

to action a_k. The idea is then to select an action a at a each state according to the max selector,

$$a = a_k \qquad \text{where} \qquad z_k = \max_{1 \le i \le m} z_i \tag{3.18}$$

After each action is selected, critic would produces an evaluation of the resulting state which is then used to adapt the actor's policy. If the critic is to give a good evaluation, the action space should be explored well such that all the actions get a chance of being evaluated by the environment. The max selector of (3.18) is in that case not a good choice, especially if the actor is not trained enough to produce the optimal merits for each action. To make the exploration of the action space possible, the action selection takes place according to a stochastic policy. A popular stochastic action selector is based on the Boltzmann distribution,

$$p_i(x) = \text{Prob}\{a = a_i | x\} = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \tag{3.19}$$

where T is a nonnegative real number (temperature) that controls the stochasticity of the action selector. For large values of T all actions would be as probable to be chosen at a state x. As $T \to 0$ the stochastic action selector approaches the max selector.

The temporal difference $\varepsilon(x)$ as calculated in (3.15) is used to adapt the actor. Let a_i be the selected action at state x. Then the following learning rule adapts the actor's policy for action a_i:

$$\hat{u}_i(x; V) := \hat{u}_i(x; V) + \beta\varepsilon(x) \tag{3.20}$$

where β is the learning rule. Intuitively, we could say that the actor is *punished* if the generated action a_i has increased the cost and *rewarded* if it reduced the cost. Punishment and reward would adapt the actor in the direction of respectively decreasing or increasing the probability of action a_i to be chosen in the future at state x. This method closely resembles the 'Law of Effect' of Thorndike from animal learning discussed earlier. The learning rule simply adapts the policy by updating the probabilities for the selected actions based on their produced outcome. During learning the actor learns which actions produce the least cost when selected at a certain state. This method would initially generate actions randomly and approach an optimal action selector as learning progresses. Look-up tables are the most common approach used to store the state-action mappings.

In the next section we extend the method to include the continuous state and action spaces.

3.5.2 Adaptive Critic Method (continuous)

We assume that the functions approximating critic and actor are both differentiable. To optimize a control process it is useful to have the derivative of the desired performance in respect with the control action $\frac{\partial J}{\partial u}$, since then it is possible to change u in the direction of improving the performance. Therefore the critic in this approach receives the control action at its input to be able to obtain this derivative. We use TD(0) to find the error needed to update the critic at each time-step, but we use a gradient based learning rule. The learning rule could be derived from minimizing $\frac{1}{2}\varepsilon^2(x)$. Define

$$E = \frac{1}{2}\varepsilon^2(x) = \frac{1}{2}\left[\left[q(x, u(x)) + \gamma\hat{J}(x(1), u(x(1)); W)\right] - \hat{J}(x, u(x); W)\right]^2 \tag{3.21}$$

Now we are going to use the method of *gradient descent* to adapt the parameters of the critic. The adaptation takes place by changing the parameters by an amount proportional to the derivative of E for the corresponding state with respect to each parameter:

$$\Delta W = -\alpha\frac{\partial E}{\partial W} = -\alpha\frac{\partial E}{\partial \hat{J}(x, u(x); W)}\frac{\partial \hat{J}(x, u(x); W)}{\partial W} \tag{3.22}$$

where α is the learning rate of the critic. The first part of the derivative could be calculated as follows:

$$\begin{aligned}\frac{\partial E}{\partial \hat{J}(x,u(x);W)} &= 2 * \frac{1}{2} * \varepsilon(x) * \frac{\partial[q(x,u(x)) + \gamma \hat{J}(x(1),u(x(1));W)] - \hat{J}(x,u(x);W)}{\hat{J}(x,u(x);W)} \\ &= \varepsilon(x) * (0 + 0 - 1) \\ &= -\varepsilon(x) \end{aligned} \tag{3.23}$$

And finally, using (3.22) and (3.23) we get

$$\Delta W = \alpha \varepsilon(x) \frac{\partial \hat{J}(x,u(x);W)}{\partial W} \tag{3.24}$$

Now we can derive the learning rule that adapts the parameters of the critic in the direction of minimizing E:

$$W := W + \alpha \varepsilon(x) \frac{\partial \hat{J}(x,u(x);W)}{\partial W} \tag{3.25}$$

Assuming that the critic is well-trained in predicting the future costs now we can calculate the gradient of critic in respect with the control action in the direction of minimizing the critic's output and use it to update the actor. Suppose that actor is approximated by $\hat{u}(x;V)$. Then we could find the amount of update for each actor's parameter by:

$$\Delta V = -\beta \frac{\partial \hat{J}(x,u(x);W)}{\partial \hat{u}(x;V)} \frac{\partial \hat{u}(x;V)}{\partial V} \tag{3.26}$$

where β is the learning rate of the actor. This would result in the following learning rule for updating the parameters of the actor:

$$V := V - \beta \frac{\partial \hat{J}(x,u(x);W)}{\partial \hat{u}(x;V)} \frac{\partial \hat{u}(x;V)}{\partial V} \tag{3.27}$$

A more detailed discussion of gradient based methods is given in Werbos [29]. Sofge & White [30] have successfully applied a gradient method to optimizing a manufacturing process.

3.5.3 Q-Learning

This method was first introduced by Watkins [27]. For a more detailed discussion of Q-learning see also Watkins & Dayan [28].

In Q-Learning unlike the adaptive critic approach we learn only one function. This function is called Q, or action value function. For a given state and action the optimal Q-function gives the estimated future cost when that action is performed and the same policy is used in the future. Using the notations we used in Section 3.4 we define the Q-function as follows:

$$Q^u(x,a) = q(x,a) + \gamma J^u(y) \tag{3.28}$$

where y is the state resulting from applying action $a = u(x)$ to state x. Suppose Q^* represents the optimal Q-function. It is clear to see that:

$$J^*(x) = \min_{a \in A(x)} Q^*(x,a) \tag{3.29}$$

where $A(x)$ represents the possible actions at state x. Now we can define the optimal Q-function using (3.28 and 3.29):

$$Q^*(x,a) = q(x,a) + \gamma \min_{b \in A(y)} Q^*(y,b) \tag{3.30}$$

In other words $Q^*(x,a)$ gives us the amount of future costs when action a is applied to the current state x and the performance is optimal afterwards. In a while, we will say a bit more about action selection at each state and the advantages of Q-function.

The goal is now to find an approximation $\hat{Q}(.,.;V)$ of $Q^*(.,.)$. We could use the methods explained in Section 3.4.1 to do this by replacing J with Q. Using the TD(0) learning method we define the following on-line learning rule for $\hat{Q}$:

$$\hat{Q}(x,a;V) := \hat{Q}(x,a;V) + \alpha \left[q(x,a) + \gamma \min_{b \in A(y)} \hat{Q}(y,b;V) - \hat{Q}(x,a;V) \right] \tag{3.31}$$

In the beginning the value of $\hat{Q}(.,.;V)$ would be chosen at random for (state,action) pairs. Initially an stochastic action selection policy could be used which makes the exploration of the action-space possible. Multiple training trials should be done to make sure that all the actions and states are visited frequently. And as the learning progresses the prediction would improve and the learning rate is reduced to zero.

The advantage of the Q-Learning is that once the Q values are learned, we can simply find the optimal actions at each state using the greedy policy in (3.32).

$$u(x) = \arg \min_{a \in A(x)} \hat{Q}(x,a;V) \tag{3.32}$$

3.6 About Our Approach

The continuous mappings seemed the most appropriate to be used for the water vessel process since a continuous action-space (inflow signals) would be the best choice to optimally control the height. Therefore, the reinforcement learning method of continuous adaptive critic explained in Section 3.5.2 has been used for our experiments.

Further we have used CMACs for approximating critic in both experiments and also for the actor in the second approach. The setup for each experiment will be given in their corresponding chapters.

It seemed appropriate to devote a complete section to the discussion of CMAC function approximator due to its importance in our project and in on-line control in general. This discussion comes next.

3.7 CMAC

3.7.1 Introduction

CMAC (Cerebellar Model Articulation controller) is a type of neural network introduced by Albus [1] to explain information-processing characteristics of the cerebellum in the brain, which is the part of the brain controlling motor activities of the body. CMAC learns the association between its multidimensional input and a single output value by using a kind of look-up table with local interpolation. The fast convergence and high accuracy of the CMAC have proven it to be quite successful when used in on-line learning controlling systems (Miller [16]). In Section 3.7.4 more of these characteristics will be discussed. But we will start with explaining the CMAC's structure in some more detail.

3.7.2 Discrete CMAC

Let's define a function $f : \mathcal{R}^k \longrightarrow \mathcal{R}$. CMAC learns the function $f(s)$ by mapping different input vectors s to their corresponding output $f(s)$. In the simplest case one could think of CMAC having one weight for each possible value of input vector s. However, this is not usually done for the following reason. Suppose that each element of the input vector could take on p different values, then p^k weights would be needed which in the case of large problems would require a lot

of memory. So often the input is mapped to a certain set of weights by methods such as hashing, which reduces the number of weights required. Figure(3.2) shows how this is done. After the

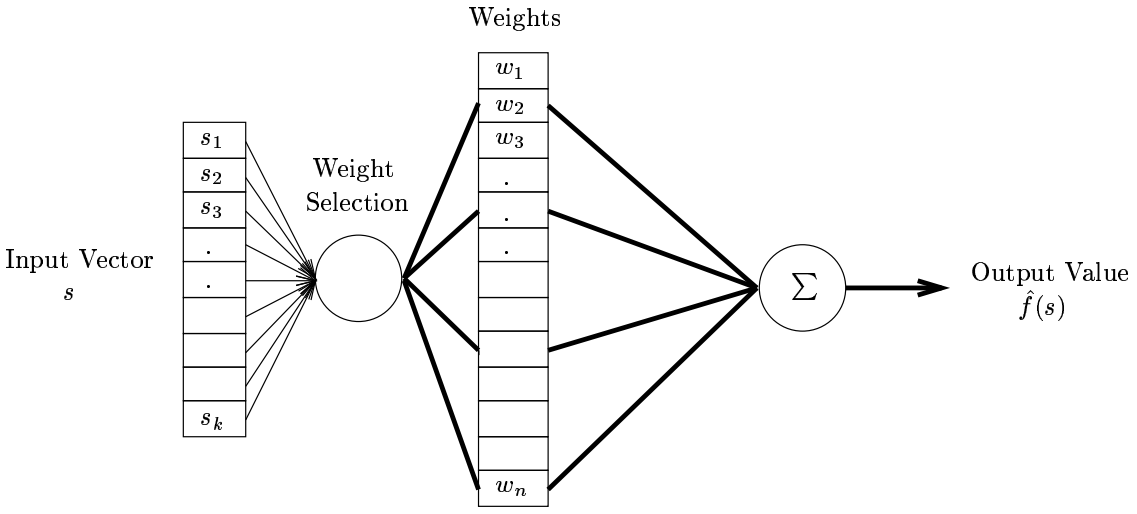

Figure 3.2: The standard CMAC structure of Albus. The weights represented by thick lines are selected to calculate the output.

weight selection has taken place the output of the CMAC is calculated using the values of the selected weights. In its simplest form the output of the CMAC is calculated by summing up the values of the selected weights. Suppose that input s is mapped into m weights represented by $w_i, i = 1, 2, \ldots, m$. The CMAC's output $\hat{f}(s)$ could be calculated as:

$$\hat{f}(s) = \sum_{i=1}^{m} w_i \tag{3.33}$$

We could simply use delta rule to adjust the selected weights. The error e is found using equation below:

$$e = f(s) - \hat{f}(s) \tag{3.34}$$

Now the following equation could be used to update each of the selected weights:

$$w_i(t+1) = w_i(t) + \alpha \frac{e}{m} \qquad \text{for} \quad i = 1, 2, \ldots, m \tag{3.35}$$

with $w(t)$ being the weight at time t and α the learning rate.

Analyzing the CMAC in Figure(3.2) is not easy since the relations between the input and the weight-space are nontrivial mappings like hashing etc. So, we will introduce a weight selection strategy which makes the use of CMAC's local response and adjusting ability easier. We choose a direct mapping between input and weight space, both having the same dimensions. In this case the components of the input vector will be treated as coordinates to find a corresponding weight for each input point in the weight space (an example will be given shortly). Further, since we usually have a limited number of points to use for training, also to reduce the time needed for training the CMAC we introduce the concept of generalization. Generalization simply states that not only the weight corresponding to a particular input is adjusted but also a group of neighbouring weights. This makes sense since similar inputs usually result in similar outputs and different inputs result in different outputs. A generalization factor g is usually used to define a generalization region having g weights in each dimension with the input point mapped at its center. The weights in the generalization region are then used to produce the output. Let's make this clear using an example.

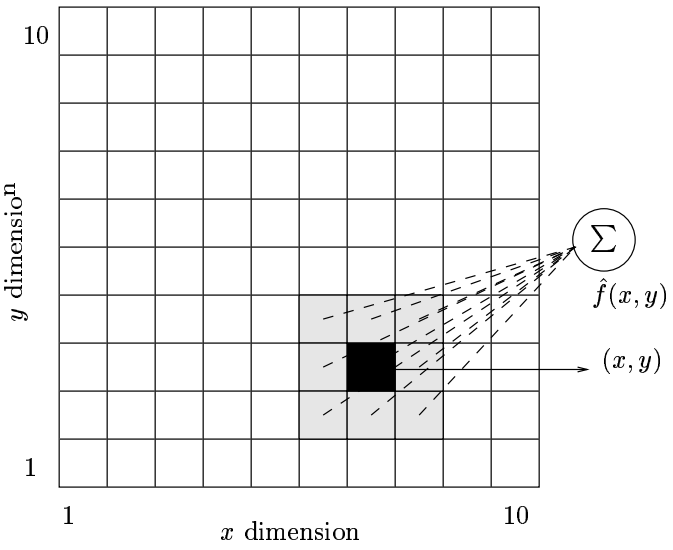

Figure 3.3: A 2-D discrete CMAC with $g = 3$. The point (x, y) represents the weight the input $(6.7, 2.8)$ is mapped to, and $f(x, y)$ is the CMAC's corresponding output.

Example of a Discrete CMAC

Consider a CMAC used to learn the function $f : \mathcal{R}^{+2} \longrightarrow \mathcal{R}$. Based on what we just discussed we choose a 2-D CMAC. We take each dimension to have 10 weights as shown in Figure(3.3). Also assume that our inputs are scaled such that each of the components has a value in the range $[1, 10]$, this could be easily done provided we know the actual ranges for each of the coordinates. This setup could be shown more precisely as follows:

$$\begin{aligned} \text{Input Space}: \quad & (s_1, s_2) \quad \text{where} \quad s_1, s_2 \in [1, 10] \\ \text{Weight Space}: \quad & w_{xy} \quad \text{for} \quad x, y = 1, 2, \ldots, 10 \end{aligned}$$

The mapping procedure of input points to the weights of the CMAC could be then explained as follows:

$$(s_1, s_2) \longrightarrow w_{xy} \qquad \text{where} \qquad x = ceiling(s_1), y = ceiling(s_2) \tag{3.36}$$

with function $ceiling(z)$ being defined as the smallest integer larger than z. Notice that when the input is real-valued, weight selection is not possible because the weight-space is discrete. According to (3.36) we have for instance $(4, 6) \longrightarrow w_{46}$. An example of a real-valued input is $(6.7, 2.8) \longrightarrow w_{73}$ as shown in Figure(3.3). The black cell is the actual weight to which the input is mapped and the weights in the shaded area are the ones selected by generalization factor $g = 3$. In general the output of the CMAC for an input mapped to the weight w_{xy} is given by (3.37):

$$\hat{f}(x, y) = \sum_{i,j=x-1,y-1}^{i,j=x+1,y+1} w_{ij} \qquad \text{for} \qquad g = 3 \tag{3.37}$$

The major drawback to the mapping produced by this CMAC is that it contains numerous step-functions. These step-functions will bring discontinuity in the mapping over the input-space, resulting in a nondifferentiable mapping. We will explain shortly why a differentiable mapping is necessary for our experiments. The proof of nondifferentiablity could be given using the definition of the derivative of a function at a point. In order for $\hat{f}(x, y)$ to be partially differentiable with respect to x at the point (x_1, y_1), it is necessary that the limit in (3.38) exists and is finite.

$$\lim_{h \to 0} \frac{\hat{f}(x_1 + h, y_1) - \hat{f}(x_1, y_1)}{h} \tag{3.38}$$

It could be easily seen that this is not the case:

$$\frac{\partial \hat{f}(x,y)}{\partial x}(x_1,y_1) = \begin{cases} \textit{undefined} & \text{where} \quad x_1 \in \mathcal{N} \\ 0 & \text{otherwise} \end{cases} \tag{3.39}$$

since

$$(3.36) \text{ and } x_1 \in \mathcal{N} \Rightarrow \begin{cases} \lim_{h\downarrow 0} \hat{f}(x_1+h,y_1) = \hat{f}(x_1+1,y_1) \neq \hat{f}(x_1,y_1) \\ \lim_{h\uparrow 0} \hat{f}(x_1+h,y_1) = \hat{f}(x_1,y_1) \end{cases} \tag{3.40}$$

from (3.40) we see that the limit in (3.38) takes on two different values $\lim_{h\downarrow 0} = \infty \neq \lim_{h\uparrow 0} = 0$ and the function is not continuous at $x_1 \in \mathcal{N}$ therefore not differentiable.

The reason for this discontinuity lies in the fact that the real values at the input points are stripped off their fractional parts by *ceiling*(.) and transformed into integers which results in step-wise mapping of the discrete CMAC.

To use CMAC as critic in our adaptive critic system we need a differentiable mapping. Then the derivative of CMAC could be calculated with respect to its input, the output of the controller, such that we can propagate the error through the controller network for its adaptation. A CMAC architecture with continuous differentiable mapping is introduced by Sofge & White [31] and explained below.

3.7.3 Continuous CMAC

We could extend the localized mapping explained above in order to achieve real-valued mappings in the weight space. To do this, we use the fractional values of the input points to select the corresponding weight without having to transform them to integers first. Let's consider the example given in Section 3.7.2. The mapping rule in (3.36) could be now rewritten as:

$$(s_1, s_2) \longrightarrow w_{s_1 s_2} \tag{3.41}$$

So from now on we will use s_1 and s_2 also as the coordinates of the selected weight in the weight space. In Figure(3.4) the same mapping of Figure(3.3) is shown in a continuous CMAC with $g = 3$.

Now we have two types of weights in the shaded generalization area. The weights which are fully located in the area are selected and updated fully and the ones which only overlap the generalization area are fractionally selected and updated. The new generated mapping is more accurate and uses the weights in a more effective way than the discrete approach explained above did. It does so by making use of the actual real-valued inputs to select weights. Of course, the main advantage is that as a result we have a continuous and differentiable mapping produced by the CMAC.

This CMAC makes use of a response function R to calculate the fraction of each weight used at each mapping, called *response-value* of each weight. We could define R as follows:

$$R = \begin{cases} 1 & \text{weight fully inside of the response area} \\ 0 & \text{weight fully outside of the response area} \\ z \in (0,1) & \text{weight partially in the response area} \end{cases} \tag{3.42}$$

The output of a continuous 2-D CMAC is then calculated as follows:

$$\hat{f}(s_1,s_2) = \frac{\sum_\beta \bar{R}^\beta(s_1,s_2)\bar{W}^\beta}{\sum_\beta \bar{R}^\beta(s_1,s_2)} \tag{3.43}$$

where s_1 and s_2 are the input values, β represents the generalization area, and $\bar{W}$ and $\bar{R}$ are resp. weights in the selected area β and their corresponding response-values. There are different response functions available, Gaussian function is the one used in our case and is discussed next.

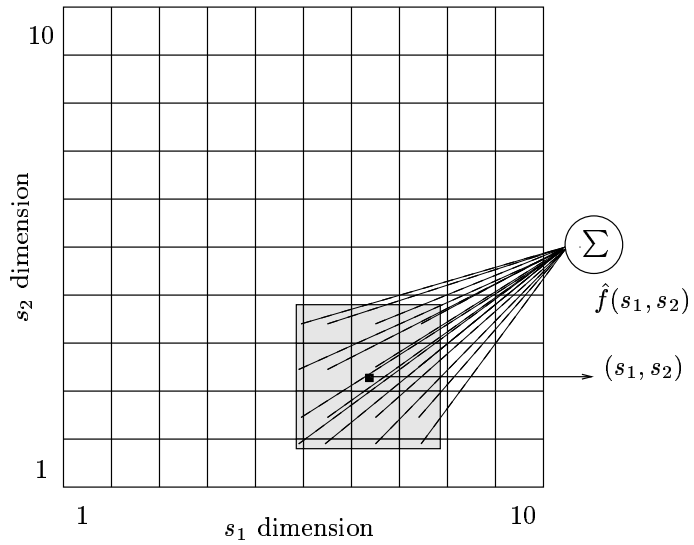

Figure 3.4: The weight space of a 2-D continuous CMAC with $g = 3$. The point (s_1, s_2) represents where the input $(6.7, 2.8)$ is mapped to, and $f(s_1, s_2)$ is the CMAC's corresponding output. Notice that the representations are changed to (s_1, s_2) since now the actual values of input are also used in the weight space.

Gaussian CMAC

In this section we will discuss a learning method for a 2-D CMAC which uses a Gaussian response function.

A Gaussian response function, when centered at the input point produces the response area and the corresponding response-values for the neighbouring weights. Using a Gaussian response function all the weights of the CMAC take part in calculating the output, with their response increasing as they get closer to the mapped input.

To make this more clear let's consider the following 2-D CMAC with N weights in each dimension. It is used to approximate a function $f : \mathcal{R}^{+2} \longrightarrow \mathcal{R}$. The output of the CMAC is calculated by:

$$\hat{f}(s_1, s_2) = \frac{\sum_{i,j=1}^{N} R_{ij}(s_1, s_2) * w_{ij}}{\sum_{i,j=1}^{N} R_{ij}(s_1, s_2)} \tag{3.44}$$

where s_1 and s_2 are the input values. The Gaussian R function is defined as:

$$R_{ij}(s_1, s_2) = \frac{1}{2\pi * v_{s_1} v_{s_2}} * \exp(-\frac{1}{2}\left[\frac{(i - s_1)^2}{v_{s_1}^2} + \frac{(j - s_2)^2}{v_{s_2}^2}\right]) \tag{3.45}$$

v_{s_1} and v_{s_2} are called variance terms. Each dimension has it's own variance term which determines the amount of generalization along that dimension (equivalent of generalization factor g discussed before). Using adjustable variance terms we can determine how smooth the mapping takes place. The smaller the variance terms the smaller the response area will be which results in a less smooth mapping. In Figure(3.5) we see how different values of the variance terms change the form of the response area in a 2-D CMAC with 20 weights in each dimension. Comparing both figures we see that the larger variance terms enlarge the response area and at the same time reduce the amount of share for each weight.

Now lets define the update method used for the weights of the CMAC. We use the same CMAC used in equation (3.44). The method of TD(0) learning explained in Section 3.4.1 is used to calculate the error in order to update the weights of the CMAC. And we use the gradient

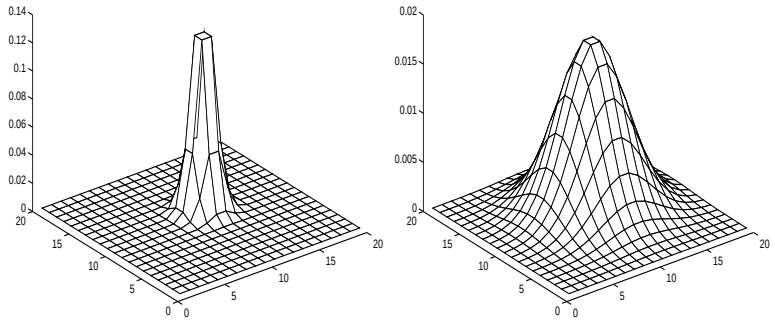

Figure 3.5: The Gaussian sharing response centered at $(10, 10)$. On the left variance terms are both 1 and on the right 3.

method explained in Section 3.5.2 for adjusting the weights of the CMAC when used as the critic. Suppose that the temporal difference at time t is represented by $e(t)$ then we could adjust the weights of the CMAC critic by:

$$w_{ij}(t+1) = w_{ij}(t) + \alpha e(t) \frac{R_{ij}(s_1(t), s_2(t))}{\sum_{i,j=1}^{N} R_{ij}(s_1(t), s_2(t))} \tag{3.46}$$

Notice that

$$\frac{\partial \hat{f}(s_1(t), s_2(t))}{\partial w_{ij}(t)} = \frac{R_{ij}(s_1(t), s_2(t))}{\sum_{i,j=1}^{N} R_{ij}(s_1(t), s_2(t))} \tag{3.47}$$

where α is the learning rate. So the amount of correction in a weight is dependent on the amount of its R value. This way the weights with more significant effect on the output would undergo more change.

3.7.4 CMAC and On-Line Control

When global learning methods such as backpropagation are used for learning a model, the order of presentation of the training data to the network could often influence the convergence of the network, learned mapping and whether it gets trapped in a local minima or not.

In on-line model learning the data is most likely nonrepeating and it may not always be possible to control the ordering of inputs. Further, data may be unequally distributed in the input space with some parts being represented with a higher density than others. For real-time control purposes it is often required that the model is quickly built without having to wait for hundreds or thousands of training samples to be presented to the network and therefore learning methods that can learn fast with a relatively smaller training set are needed.

CMAC networks seem to be very suitable for on-line model learning. Unlike global learning methods CMACs don't cause global degradation of the network because the weight update takes place locally. In global learning the repetitive adaptation for a localized part of the input space causes global and mostly undesired changes to weights just to minimize the error for that region of the input. Further CMACs avoid the local minima. The high complexity of the networks using global learning methods usually results in error surfaces which contain a lot of ups and downs, and in the process of minimizing the error the network could get trapped in a local minima. This is not the case with CMAC since the output is simply in the linear form of $\sum_{ij} w_{ij}$. When we choose for a gradient method of adjusting the weights of the CMAC in order to minimize the squared error, we will have an error surface of the second order which does not result in a local minimum. Other advantages of CMACs is that they are not sensitive to the ordering and the distribution of training data.

All these characteristics make the local learning method used in CMAC networks ideal for real-time learning purposes.

Chapter 4

Experiments with the Linear Controller

4.1 Introduction

This chapter presents the setup and the results of the experiments with the single water vessel using different control methods. First, some aspects of the modeling of the single vessel are discussed. Then the setup for each of the controllers used is explained in Section 4.3. Sections 4.5 and further will present the results of the experiments done with P-, PI- and RL-Controllers. Finally in Section 4.8 the results will be discussed and compared.

4.2 A Model of The Process

Figure 4.1 shows the dimensions of the real system as they have been used in the simulations with d_i and d_o denoting the diameters of the vessel at its top and exit.

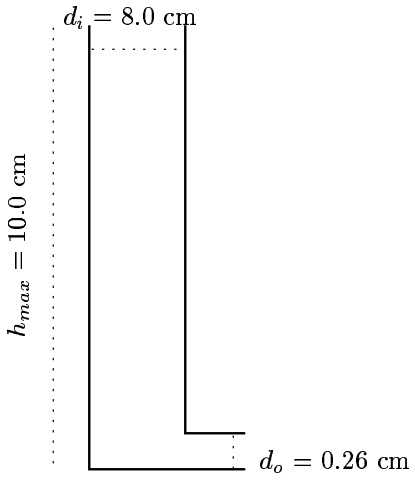

Figure 4.1: The dimensions of the real vessel.

The following differential equation is used for the purpose of modeling the system as described in Boom [7]:

$$\frac{dh}{dt} = \frac{1}{A_i}(f_i - f_o) \tag{4.1}$$

where h represents the height of the water in the vessel at time t and A_i the area of the vessel at the top and f_i and f_o resp. the inflow and the outflow of the vessel. For calculating the outflow we use:

$$f_o = A_o\sqrt{2hg} \tag{4.2}$$

with A_o being the area of the vessel at the exit and $g = 9.8$ the gravity constant. A_i and A_o are calculated by:

$$A_i = (\frac{d_i}{2})^2 * \pi \qquad A_o = (\frac{d_o}{2})^2 * \pi \tag{4.3}$$

Further, we have the followings:

- The maximum value of the flow of the pump is $38\frac{ml}{sec}$.
- A_i and A_o are kept constant throughout the process.
- There is a variable delay involved in the process. That is, the time needed for the water to run down the vessel's wall in order to actually influence the height. This delay changes depending on how much water is already in the vessel and is between 1 to 2 seconds at its maximum. During the simulations a constant 2 second delay is used meaning:

  $$f_i(t) = f_p(t-2) \tag{4.4}$$

  where $f_i(t)$ represents the effective inflow at time t and f_p the flow of the pump.
- In the simulation we consider the presence of low-frequent disturbances on the outflow. This disturbances represent the changes in the height of the water in an unexpected manner. Suppose that something gets stuck in the exit and reduces the amount of outflow or that there is a leak in the vessel such that it increases the outflow. The former is simulated by making the exit smaller and the latter by making the exit larger. For this purpose a step signal is used which changes the size of the exit as in Figure 4.2 during 500 seconds. First, it is simulated that something is stuck in the exit and is gradually washed away. Afterwards, a hole in the vessel is simulated which after sometimes is found and repaired! Notice that in reality these intervals and types of disturbances might be different but we need somehow to simulate them for these experiments.

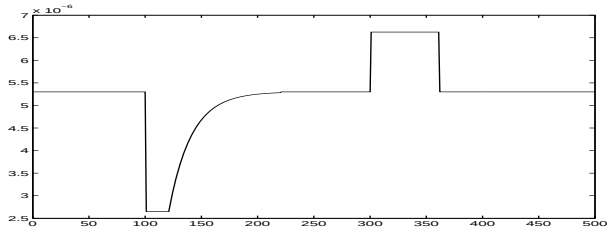

Figure 4.2: The area of the exit A_o changing during 500 seconds.

The system will be tested under the following conditions:

1. No disturbance and no delay: in Section 4.5.
2. Delay: in Section 4.6.
3. Delay and disturbance: in Section 4.7.

4.3 Setup of Controllers

In the following setup descriptions we use h_d and h respectively to refer to the desired height and the current height, $e = h_d - h$ is the error in height. e could therefore range from $-100cm$ to $100cm$ which in the negative region would cause the controllers to produce a negative output meaning that some water should be pumped out. Since this is not an option for our pump we limit the controller output to only positive values acceptable by the pump. Considering that the pump also has a maximum flow, the control signal should always be set between 0 and $38\frac{ml}{sec}$ as shown in Figure 4.3.

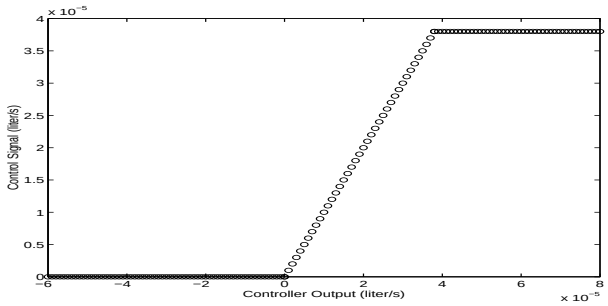

Figure 4.3: Actual control signal as a function of controller output.

We will use u and f_p to resp. represent the controller output and the control signal sent to the pump.

4.3.1 P-Controller

This type of controller was discussed in Section 2.7.1. The complete vessel control system is shown in Figure 4.4. The controller output is calculated by $u = K_p * e$.

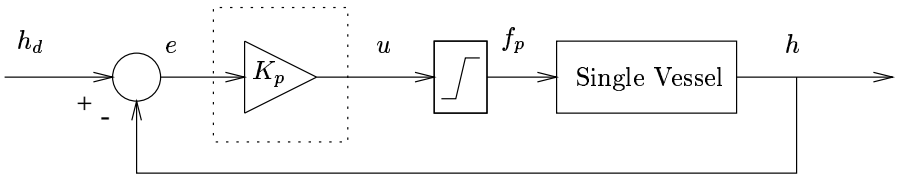

Figure 4.4: Single vessel controlled by a P-Controller.

4.3.2 PI-Controller

We discussed this class of controllers shortly in Section 2.7.3. Figure 4.5 shows the system when PI-controller is used. The control output is calculated by $u = K_p * e + K_i * e_s$.

4.3.3 RL-Controller

Finally we come down to the experiments with the RL-Controller (see Section(3.3)). Figure 4.6 shows the resulting control system.

The controller output is calculated by $u = w * e$, with w being a gain value which is adapted using the reinforcement signal r sent from the critic. The critic (see Section 3.7) used in the simulations is implemented as a 2-dimensional CMAC with 10 weights in each dimension. The inputs to the critic are the control signal sent to the pump f_p and the error in height e. Critic

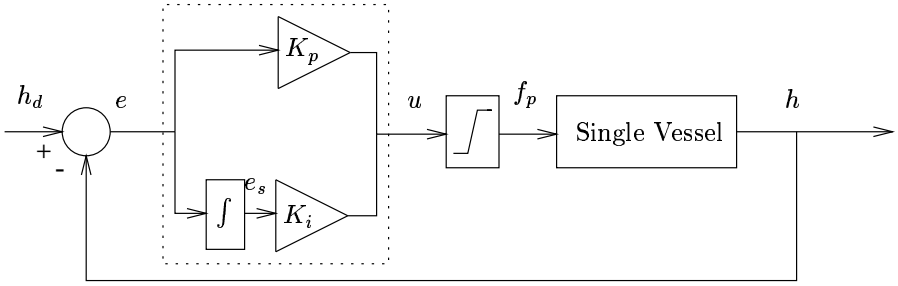

Figure 4.5: Single vessel controlled by a PI-Controller.

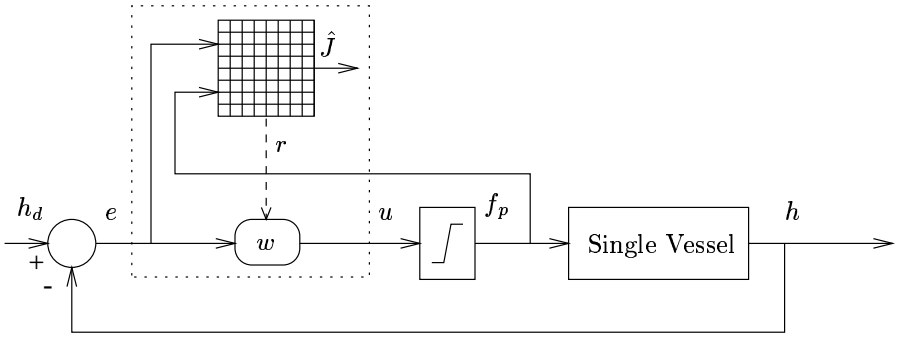

Figure 4.6: Single vessel controlled by an RL-Controller.

produces a reinforcement signal which is an approximation of the expected value of the discounted sum of future costs.

$$\hat{J}(e(t), u(t), t) = \sum_{i=t}^{\infty} \gamma^{i-t} q(i) \tag{4.5}$$

where the immediate cost of the system q is defined as the squared error in height.

$$q(t) = e(t)^2 \tag{4.6}$$

It should be mentioned that critic is designed to work with positive errors only. To deal with negative errors we define the controller output u as shown in Figure 4.7. This means that the RL-Controller is not adapted for $e \leq 0$.

$$u = \begin{cases} e * w & \text{where} \quad 0 \leq e \leq 100cm \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

Just to be able to do some general comparisons later, let us mention that the controller could be also seen as a P-Controller with a variable gain which is its weight. The reinforcement signal from the critic provides the controller with the information regarding how to adapt its weight. The parameter set shown in Table 4.1 has been used for all simulations.

4.4 Simulation Notes

In this section some general notes will be given on the setup of the simulations.

Matlab has been used as the simulation environment for all the experiments. In all the experiments a step reference signal is used with 500 seconds duration for each step. This signal is shown in Figure 4.7.

In Section 4.7.4 an improved method of learning is introduced for the RL-Controller using a normal noise on the height which we call adaptation noise.

Parameter	Value
Critic learning rate	100.00
Critic Discount factor	0.95
Critic variance for e dimension	1.00
Critic variance for u dimension	6.00
Controller learning rate	300.00

Table 4.1: Linear RL-Controller parameter set.

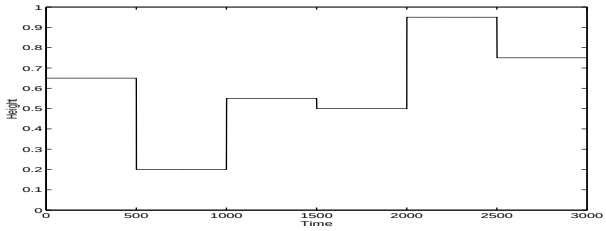

Figure 4.7: Reference signal used in all the experiments.

Since there will be a lot of plots to look at and compare, the following abbreviations will be printed under each figure to show which case they belong to. I blame the rest of the readability problems on LaTeX.

D^+,D^- For resp. system with and without delay.

LF^+,LF^- For resp. system with and without low-frequent disturbances on the outflow.

AN^+,AN^- For resp. system with and without adaptation noise.

4.4.1 Conventional Controllers

All the values for the proportional and integral gains have been found by trial and error. In case of the proportional gain the chosen values are the maximum values for which the height does not oscillate. For the integral gain a compromise is made between the amount of overshoot and the faster decrease in the error.

4.4.2 RL-Controller

- The parameters as shown in Table 4.1 are set based on the behaviour of the system. The current values seem to work fine with the single vessel process.
- When the training starts both controller and critic's weights are initialized to zero. The weights of both elements will be adapted every $1s$ till the reference signal ends. If the training needs to be repeated the controller's weight is initialized to zero again but critic will use its already trained weights. Basically, the idea is to get the critic well-trained such that it can produce good reinforcement for the adaptation of the controller.
- At the end of each Result-section an example is given in which the trained RL-Controller finds the gain for a setpoint not previously present in the training.
- In the final case where both delay and the low-frequent disturbances are present in the process an extra case is simulated in which the learning is improved by adding a normal noise with mean zero and variance 2 cm to the destination height. It resulted in faster learning and a more stable controller weight.

4.5 No Disturbance and No Delay

4.5.1 Result of P-Controller

The value of the proportional gain was chosen as $K_p = 6.0e-3$. The result is shown in Figure 4.8.

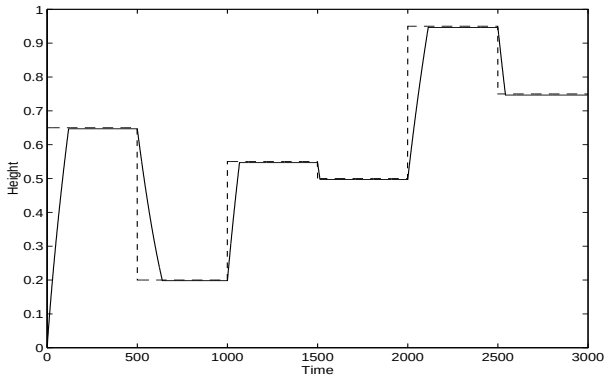

Figure 4.8: D^-LF^-: P-Controller.

Notice that the system becomes stable but the error is resting at a value unequal to zero. As we discussed it before in Section 2.7.1 the steady state error is a known problem of P-Controllers.

4.5.2 Result of PI-Controller

In this case $K_p = 6.0e-3$ and $K_i = 4.0e-5$ were chosen. Figure 4.9 shows the result for this case.

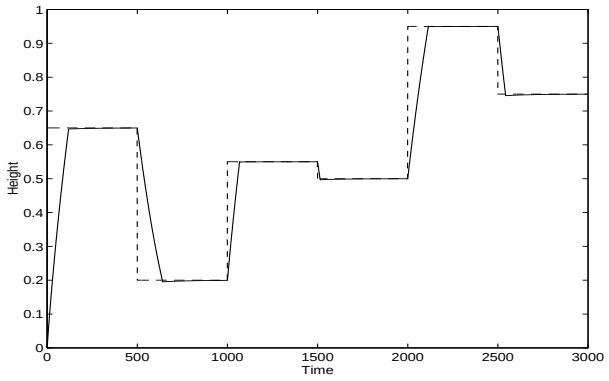

Figure 4.9: D^-LF^-: PI-Controller.

As we explained before the accumulation of the error at steady state drives the controller towards the zero error state.

4.5.3 Result of RL-Controller

Figure 4.10 shows the result obtained by this controller.

As you notice there are large stationary errors for some of the setpoint. By training the critic once more with the same reference signal we see that the controller improves its performance by resulting in a smaller final error for each setpoint as in Figure 4.11.

The changes of the controller's weight during the training is as in Figure 4.12.

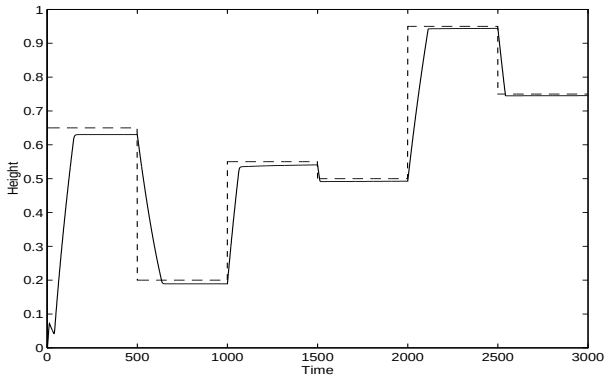

Figure 4.10: D^-LF^-: RL-Controller first training.

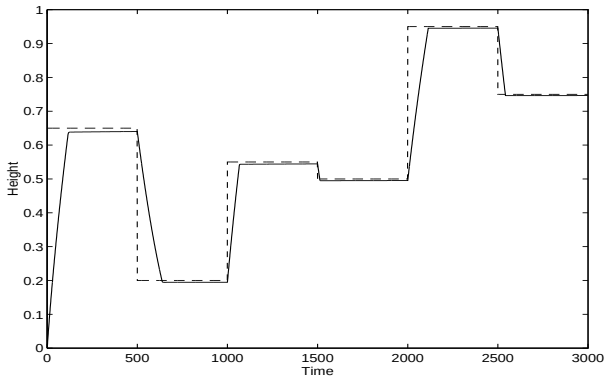

Figure 4.11: D^-LF^-: RL-Controller second training.

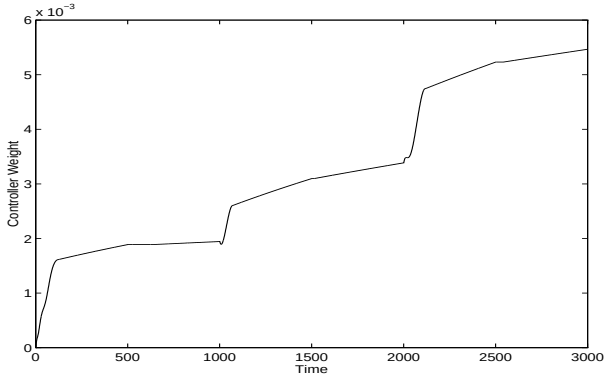

Figure 4.12: D^-LF^-: The controller weight during the training.

The surface of the CMAC critic after the second training is shown in Figure 4.13.

Example of a new setpoint

Now using the trained critic of Figure 4.13 we will give an example of how the RL-Controller will adapt its weight to bring the height to a specific setpoint. We will assume that the vessel is filled till the height of 10 cm and we want to increase the level of water to 80 cm. Figures(4.14(a) and (b)) show how the changes in the vessel and the RL-Controller take place such that the desired height is reached.

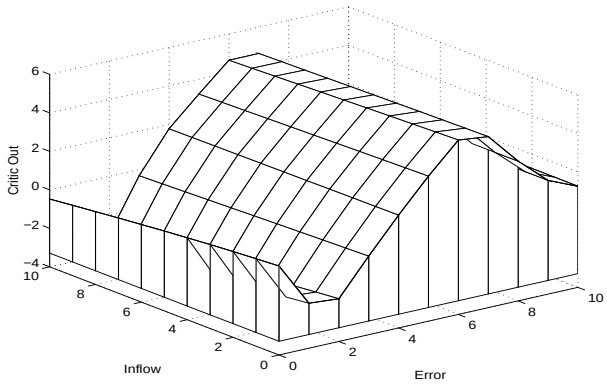

Figure 4.13: D^-LF^-: The surface of the critic after the reference signal.

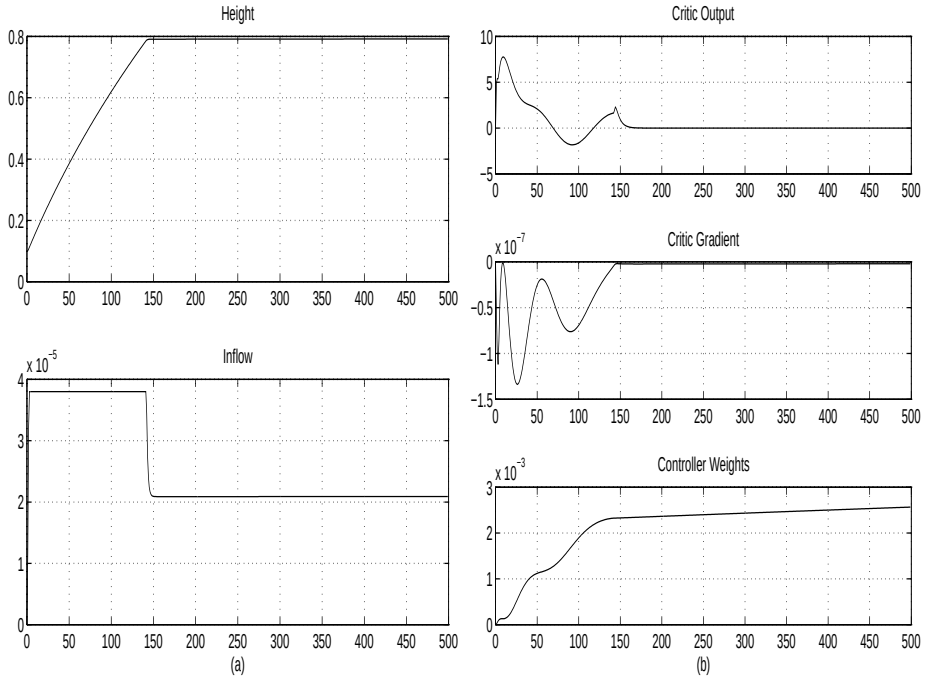

Figure 4.14: D^-LF^-:(a) The change of height and inflow in the vessel. (b) The change of critic and controller.

The RL-Controller results in a stable state with the error unequal to zero. The inflow is almost constant and the weight of the controller is converging very slowly. This is more obvious if we look at the gradient value which is very close to zero. The reason for this is similar to what caused the P-Controller to produce the stationary error. No matter how well our RL-Controller is adapted it could never produce a zero error since that causes the control signal to become zero too.

To give you an idea how the surface of the critic changes, a plot of this surface is given in Figure 4.15. Notice how the region corresponding to $error = 7cm$ increased its height. This increase will help the critic to produce better gradients next time if a reference signal contains an error around this value.

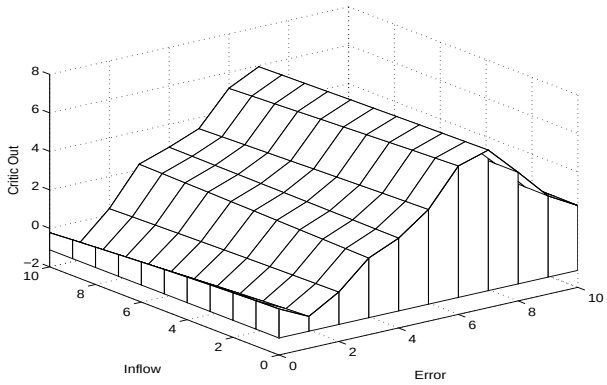

Figure 4.15: D^-LF^-: The surface of the critic after the example.

4.6 Delay

4.6.1 Result of P-Controller

First, the same parameters were used as in Section 4.5.1. But as the result in Figure 4.16 shows the controller started oscillating.

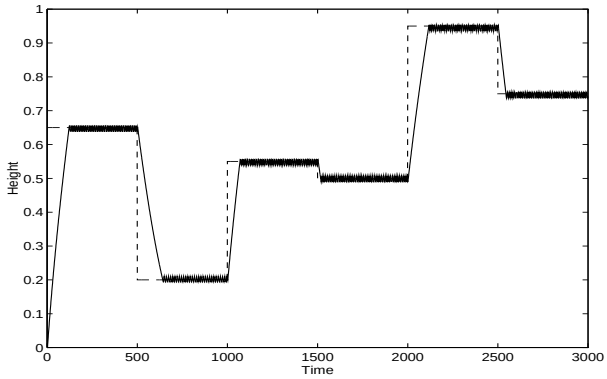

Figure 4.16: D^+LF^-: P-Controller oscillating.

To make the system less sensitive to the delay and produce a constant inflow we need to reduce the proportional gain. The $K_p = 2.2e-3$ was used which resulted in Figure 4.17.

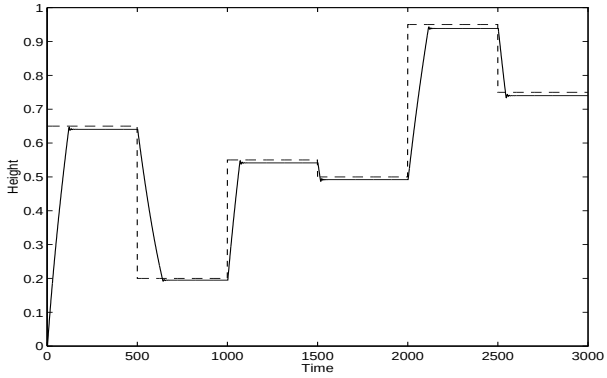

Figure 4.17: D^+LF^-: P-Controller.

4.6.2 Result of PI-Controller

Now the P-Controller of the previous section was used ($K_p = 2.2e-3$) but to achieve the zero error without too much overshoot the integral gain had to be reduced to adjust to the delay. The $K_i = 1.0e-5$ was used. Figure 4.18 shows the result.

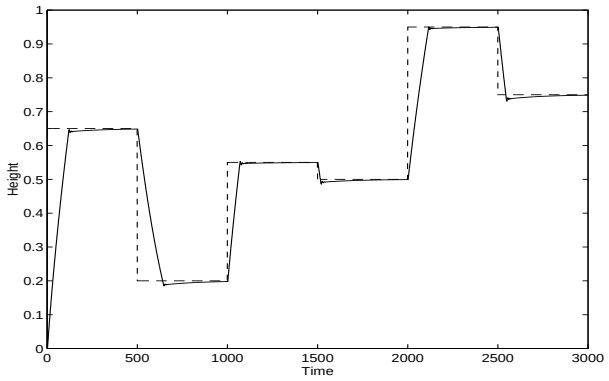

Figure 4.18: D^+LF^-: PI-Controller.

4.6.3 Result of RL-Controller

By training the critic twice with the same signal we get the result shown in Figure 4.19.

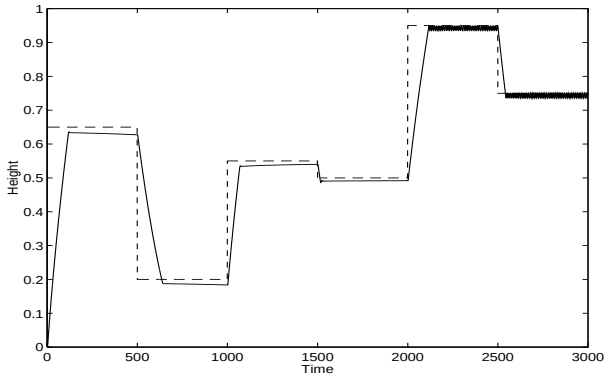

Figure 4.19: D^+LF^-: RL-Controller.

Let's see how the controller's weight changes during the training. As shown in Figure 4.20 the value of the weight reaches a maximum of $3.3e-3$ near the end of the training signal. This is less than the maximum reached in Figure 4.12 when there was no delay in the system. In a way, this tells us that the RL-Controller is adjusting itself to the delay and keeps the value of the controller's weight relatively small in order to avoid the oscillation of the height at the setpoint. There is also another interesting point related to this weight change. As you see near to the end of the training some oscillation occurs which is due to the increase of controller's weight. The reason for that is because the objective of the critic is to decrease the error and it does that by increasing the inflow, in other words, the controller's weight. However, this increase of the control signal could cause the oscillation of height when the inflow is delayed. Of course, the oscillation does not always have to increase the error, but if after sometime the critic finds out that the performance is worsening then it produces a corresponding reinforcement to the controller such that it adjust its weight value to improve performance (look at the weight reduction at the end).

The surface of the CMAC critic after the second training is shown in Figure 4.21.

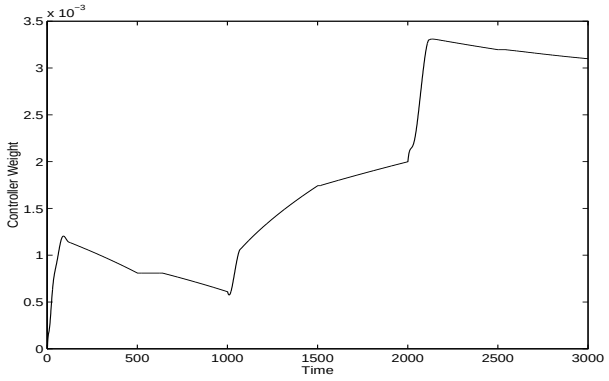

Figure 4.20: D^+LF^-: The controller weight during the training.

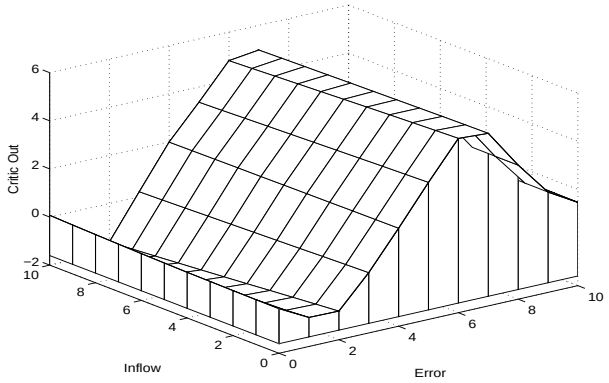

Figure 4.21: D^+LF^-: The surface of the critic after the reference signal.

Example of a new setpoint

Now using the trained critic of Figure 4.21, we will test the same example from the previous section. Figures(4.22(a) and (b)) show how the changes in the vessel and the RL-Controller take place such that the desired height is reached.

A plot of the critic's surface after this example is given in Figure 4.23.

4.7 Low-Frequent Disturbances

4.7.1 Result of P-Controller

The same parameters as the case with delay were used. Figure 4.24 is the result of using this controller under disturbances on the outflow and delay.

4.7.2 Result of PI-Controller

The same parameters as the case with delay were used. Figure 4.25 has the result for this case.

4.7.3 Result of RL-Controller

This time we trained the critic three times in order to get the result shown in Figure 4.26. The RL-Controller does seem to deal well with these disturbances on the outflow and the height stays reasonably close to the setpoint despite the changes in the outflow. There is however the oscillation of the height at the set point that we already discussed in Section4.6.3.

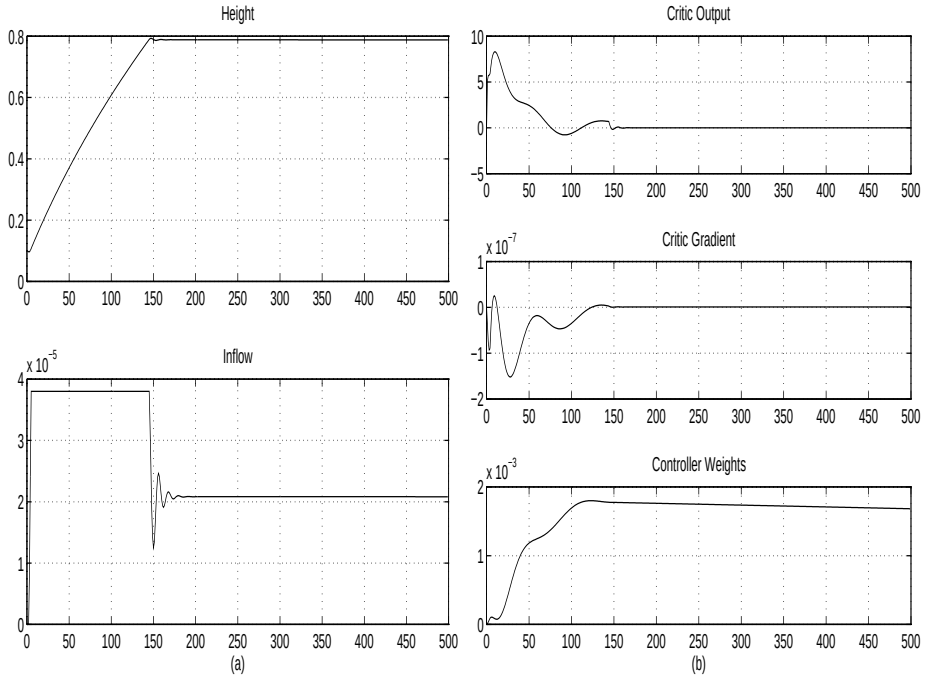

Figure 4.22: D^+LF^-:(a) The change of height and inflow in the vessel. (b) The change of critic and controller.

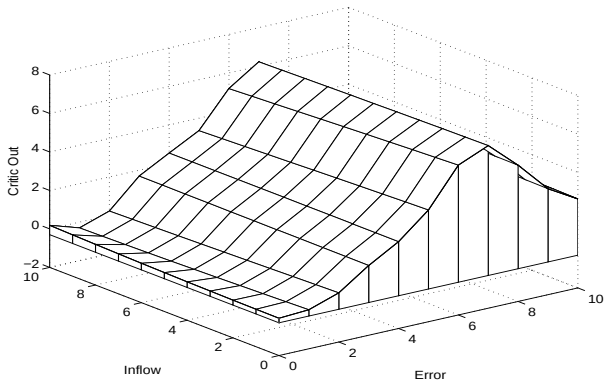

Figure 4.23: D^+LF^-: The surface of the critic after the example.

The changes of the controller's weight during the training is as in Figure 4.27. The weight seems to have become more stable due to the added noise. This stabilization is a result of the better exploration of the critic's surface caused by the noise. We will discuss this more in Section 4.8.

The surface of the CMAC critic after the third training is shown in Figure 4.28.

Example of a new setpoint

Now using the trained critic of Figure 4.28 we will test the same example from the previous section. Figures(4.29(a) and (b)) show how the changes in the vessel and the RL-Controller take place such that the desired height is reached.

A plot of the critic's surface after this example is given in Figure 4.30.

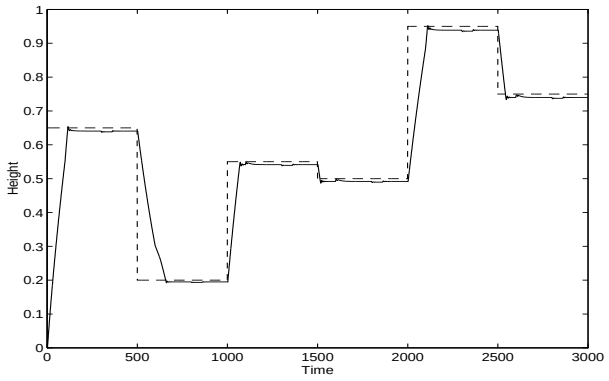

Figure 4.24: D^+LF^+: P-Controller.

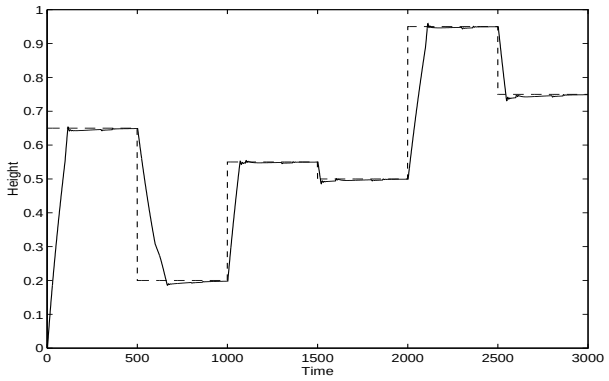

Figure 4.25: D^+LF^+: PI-Controller.

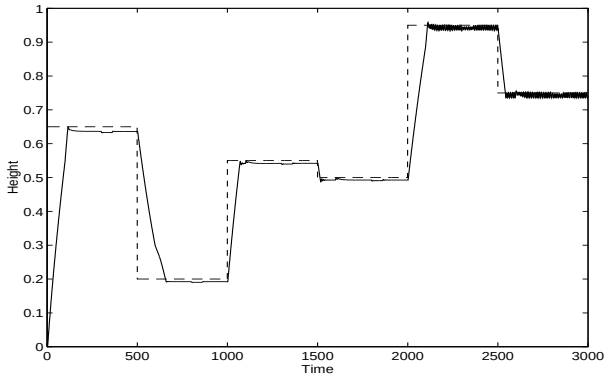

Figure 4.26: $D^+LF^+AN^-$: RL-Controller.

4.7.4 Result of RL-Controller With Improved Learning

In this section we will see how the convergence of the controller's weight could be improved by adding a normal noise with mean zero and variance 2 cm to the destination height. In Section 4.8 we will discuss this issue a bit more. This time the critic was trained two times, to produce the results shown in Figure 4.31, so it is faster than the case in Section 4.7.3.

The changes of the controller's weight during the training is as in Figure 4.32. Comparing this changes to Figure 4.27 we notice an improvement in the stabilization of the controller's weight due to the adaptation noise. This will become more clear when we consider the example of a new

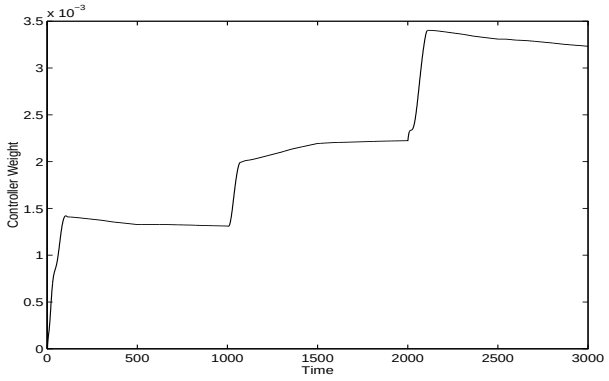

Figure 4.27: $D^+LF^+AN^-$: The controller weight during the training.

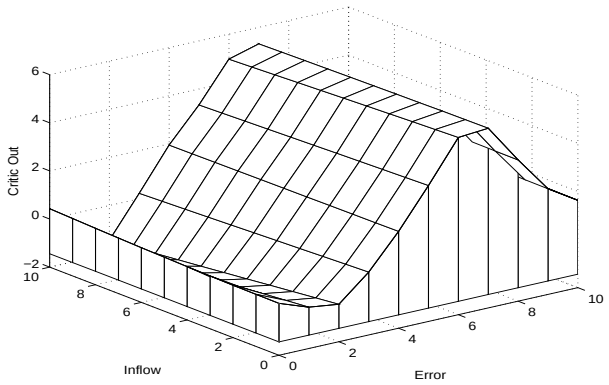

Figure 4.28: $D^+LF^+AN^-$: The surface of the critic after the reference signal.

setpoint next.

The surface of the CMAC critic after the second training is shown in Figure 4.33.

Example of a new setpoint

Now using the trained critic of Figure 4.28 we will test the same example from the previous section. Figures(4.34(a) and (b)) show how the changes in the vessel and the RL-Controller take place such that the desired height is reached. Notice that the weight of the controller stays stable comparing to the previous case, and further it is set to a higher value which could reduce the final error even more.

A plot of the critic surface after this example is given in Figure 4.35. Comparing this to the critic of Figure 4.30 shows us that the region around the zero error line is almost flat which in other words means that the gradient of the critic with respect to u is zero in that region. This is also why the controller weight stays stable in this case.

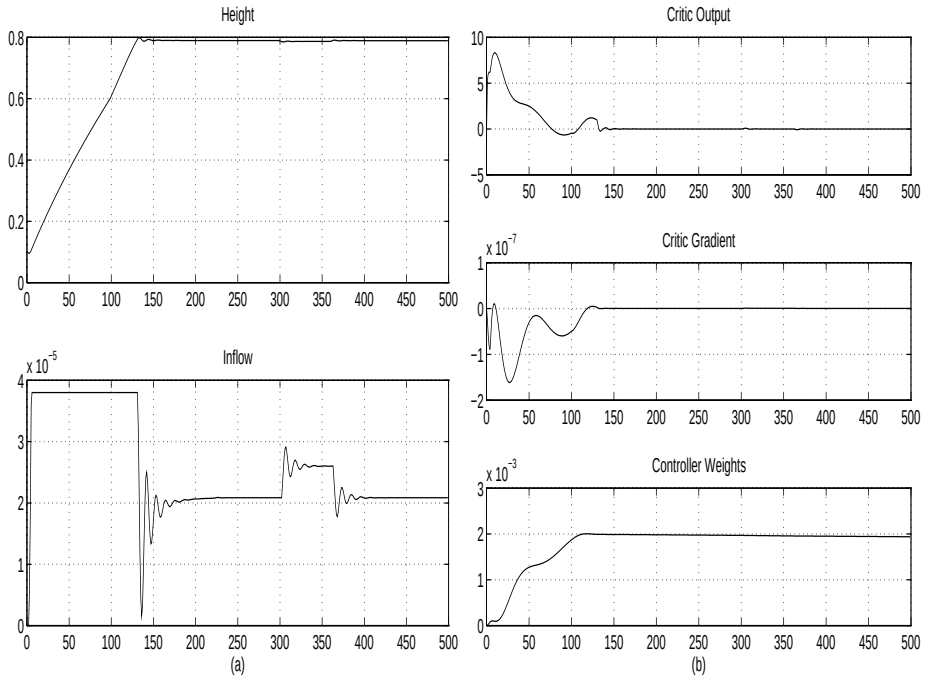

Figure 4.29: $D^+LF^+AN^-$:(a) The change of height and inflow in the vessel. (b) The change of critic and controller.

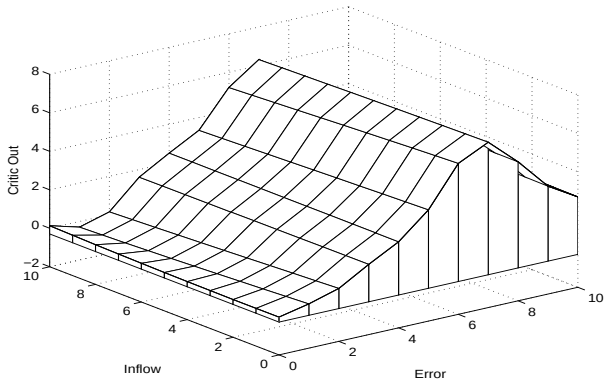

Figure 4.30: $D^+LF^+AN^-$: The surface of the critic after the example.

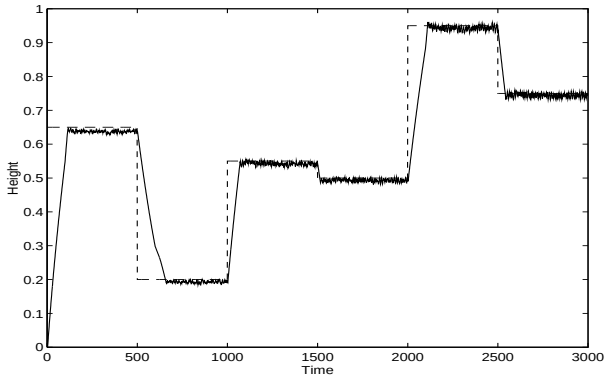

Figure 4.31: $D^+LF^+AN^+$: RL-Controller.

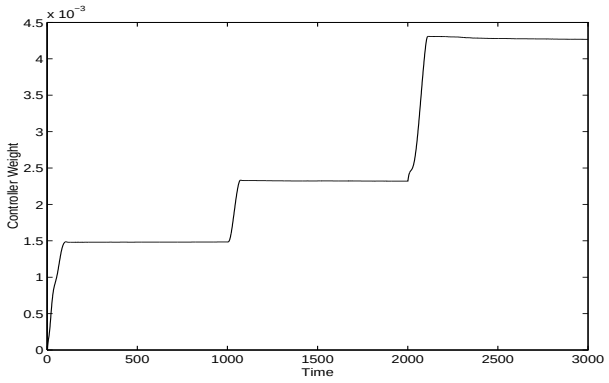

Figure 4.32: $D^+LF^+AN^+$: The controller weight during the training.

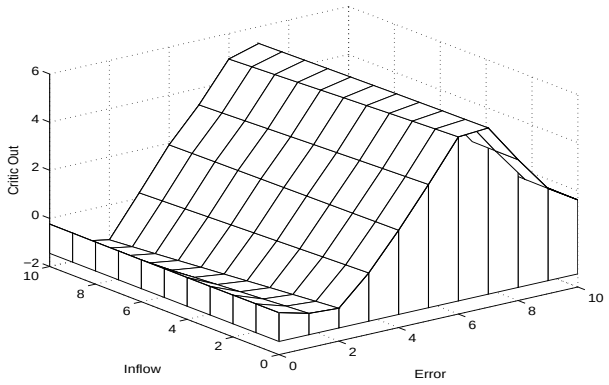

Figure 4.33: $D^+LF^+AN^+$: The surface of the critic after the reference signal.

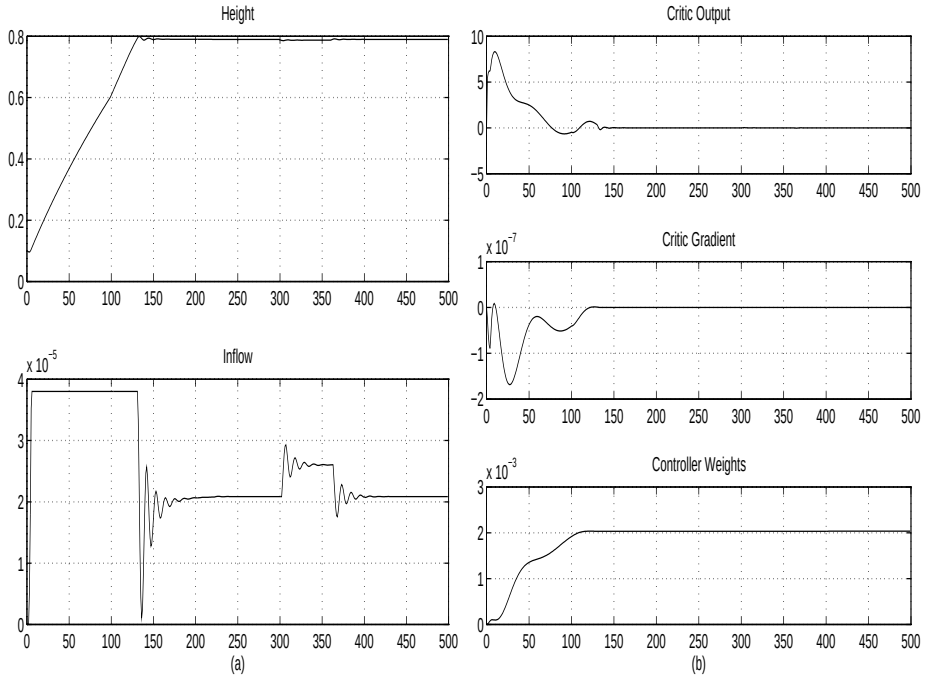

Figure 4.34: $D^+LF^+AN^+$:(a) The change of height and inflow in the vessel. (b) The change of critic and controller.

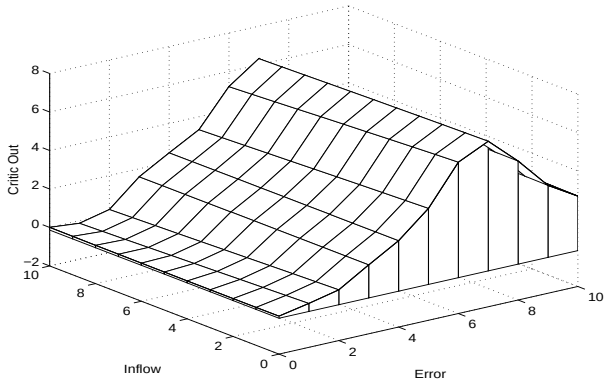

Figure 4.35: $D^+LF^+AN^+$: The surface of the critic after the example.

4.8 Conclusion

Tables 4.2, 4.3 and 4.4 contain the RMS (root of the mean squared) height error for all three different cases under which the simulations took place. Looking at the overall performances we

Controller	RMS
P	0.1052
PI	0.1051
RL	0.1053

Table 4.2: D^-LF^-: RMS height errors.

Controller	RMS
P	0.1085
PI	0.1084
RL	0.1084

Table 4.3: D^+LF^-: RMS height errors.

Controller	RMS
P	0.1086
PI	0.1085
RL(LF)	0.1088
RL(LF,AN)	0.1087

Table 4.4: D^+LF^+: RMS height errors.

can conclude that the variable gained RL-Controller is performing more or less at the same level of it's equivalent P-Controller. PI-Controller shows the best performance since it can reduce the final error to zero.

The adaptive RL-Controller could adjust its weights by itself for all cases to achieve almost the same results as the conventional controllers. On the other hand, the gain values for the conventional controllers needed to be adjusted manually (by an expert!) in order to get reasonable results under delay and disturbance on the outflow.

Naturally the performance of all three controllers is affected by adding the disturbance, but we can notice a small improvement in the performance of RL-Controller by doing so. Comparing Figures(4.22) and (4.29) shows us that in the latter case the controller's weight is more stable than the former caused by the presence of the disturbance. In a while we explain why that is. Of course we could not expect that the RMS error would become smaller too, since adding the system noise would cause some increase in the RMS error anyway.

Also some improvement was achieved when we added the extra adaptation noise to the height. We need less training of the critic, also the controller's weight seems to converge faster. It might be more clear if we compare the Figures(4.29) and (4.34). The weight of the controller is more stable in the latter case where the adaptation noise was added to train the critic.

The reason for the improved performance by adding disturbance on the outflow and noise could be explained as follows. The adaptation of the controller weight in the RL-Controller depends on the reinforcement signal produced by the critic. The critic's learning on the other hand depends on how well its surface is being explored. Adding the outflow-disturbance and noise to the system utilizes the exploration process of the critic and as a result the produced reinforcement signal will be better.

To draw the final conclusion it should be added that the important issue here is not whether the RL-Controller is producing a smaller RMS error than the conventional P-Controller or not. You could actually see that the conventional controller is performing better than the RL one in two out of three cases. This results are based on a limited set of tests and are by no means the proof for better performance.

However, an important result is that the RL-Controller is adaptive (needing a bit of training) and can find a reasonable value for the controller gain by itself under each of the simulated cases. The conventional controllers required adjustments of their parameters by a supervisor to perform well.

Chapter 5

Experiments with the Nonlinear Controller

5.1 Introduction

In the previous chapter we showed how a RL-Controller with one variable weight could produce control signals in order for the system to follow a certain reference signal. The output of the linear RL-Controller was calculated by $u = e * w$. However, one major problem was that this controller would never achieve zero error since that would also mean that the control action would be zero which is not desired in the water vessel process. To overcome the problem two other solutions were considered. We could either add a bias term to the linear controller $u = e * w + bias$ or design a nonlinear controller where for example the control action will be a function of the error and the destination height $u = f(e, h_d)$. Looking at the vessel's differential equation and the nonlinear behaviour of the outflow, the latter design was chosen for the implementation. In this chapter we are going to see how such a controller performs when used to control the inflow.

It should be added that this controller was designed as the final part of my project and due to shortage of time is not yet fully studied. But still it does show that such a controller could be trained to control the single vessel process. Hopefully the results could shed some light on the further research of this problem.

5.2 RL-Controller

A schematic representation of the control system when the nonlinear RL-Controller is used is given in Figure 5.1.

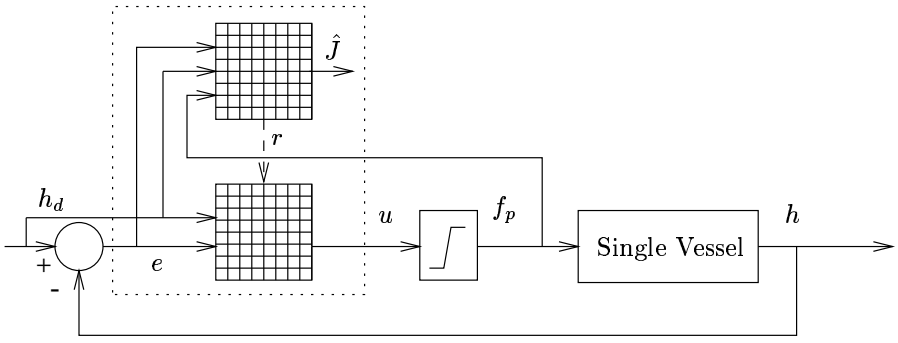

Figure 5.1: Single vessel controlled by a nonlinear RL-Controller.

The critic was chosen to be a 3-dimensional 6x6x6 CMAC. Its inputs are system error e, destination height h_d and the control action f_p. The controller is a 2-dimensional 7x7 CMAC. The controller output is a function of e and h_d. Using h_d as an input to the controller, we expect to improve the performance of the system since the outflow increases with height. An increase in height causes an increase in the outflow, according to $f_o = A_o\sqrt{2hg}$ (previous chapter). This would simply mean that the inflow should increase too if we are to keep the height constant at its level since $h' = \frac{1}{A_i}(f_i - f_o)$.

In the linear approach we reduced the range of e to only its positive values to avoid problems with the critic that was designed only for positive errors. But, now we use a different range for e and we define the controller output u as shown in Figure 5.1. This means that the RL-Controller is not trained for $e \leq -10cm$.

$$u = \begin{cases} f(e, h_d) & \text{where} \quad -10cm \leq e \leq 100cm \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

Increasing the error range to include some negative values is expected to improve the performance of the controller when the error is negative. The controller could actually start producing inflow when approaching h_d from a higher height. A well-trained controller would then be able to find the correct inflow before the water level has dropped below the destination.

5.3 Training

In the previous chapter we used the reference signal shown in Figure 4.7 to train and test the linear RL-Controller. This signal was presented to the control system a number of times till we finally had a trained critic which could find the optimal value for the controller's weight when initialized to zero.

The nonlinear controller CMAC can memorize the control signals for different system states. For this reason a different training method was used that makes use of this ability. The controller is only initialized once to zero and then trained together with the critic for all times. The training procedure is explained next.

We train both critic and controller by letting the system track a set of reference points. This differs a bit from the training of the linear RL-Controller of the previous chapter. As Figure 4.7 shows, in that approach when a new reference point was presented to the system, the initial height was set to the height of the water at that time. However, in the new approach the initial height is determined by the training points and might be different from the height of the water at the start of the training. The training points are pairs of (e, h_d). When the point $(3, 8)$ is presented to the system, the initial height should be $5cm$ and the destination height $8cm$. When a real water vessel is involved and not a simulation, it could mean that a supervisor has to change the height of the water manually such that the initial height is $5cm$, and then start the training for this point. Point $(-4, 2)$, would then require the initial height to be $6cm$ and the destination height $2cm$ and etc.

Although this method might look inefficient at first, but it speeds up the learning of the controller's surface a great deal since the training points could be selected by the supervisor where necessary. And besides, this happens only for the training and once the controller is trained the system will be left untouched to track the reference points as it receives them. It should be mentioned that using the signal in Figure 4.7 for the training is also possible, but then the controller would be learnt partially. To avoid this we would have to include more reference points in the signal to let the controller explore as many different situations as possible. That would take relatively more time than the method just explained.

Figure 5.2 shows which points of the controller surface are presented to the system as samples. Each sample was represented to the system for a maximum of $800s$. The controller and critic were adapted every $1s$. The system status was monitored every $50s$ and the training for each sample was stopped when the controller either achieved the zero error or the changes in height became very small while still not at destination. Notice that the latter could happen since the RL-Controller

is not yet trained to reach the setpoint as fast as possible. It should also be mentioned that by zero error we mean a very small error which is continuously decreasing.

After the system was trained once with all the samples, it was presented for the second time with the ones for which it had failed to find the correct control signal the first time.

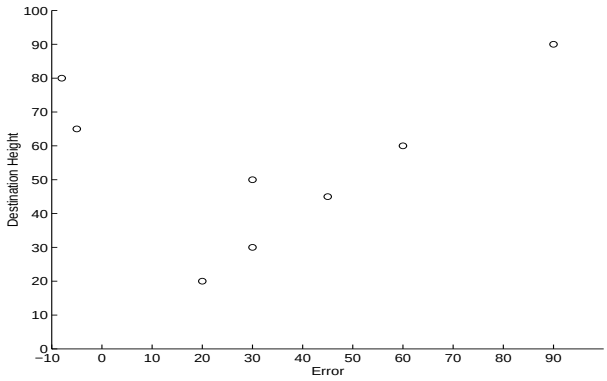

Figure 5.2: Samples used to train the nonlinear RL-Controller.

5.4 Simulation Notes

The parameter set shown in Table 5.1 has been used for all simulations.

Parameter	**Value**
Critic learning rate	0.9
Critic variance for e dimension	0.2
Critic variance for u dimension	0.4
Critic variance for h_d dimension	0.2
Critic discount factor	0.9
Controller learning rate (training)	0.8
Controller learning rate (testing)	0.01
Controller variance for e dimension	0.3
Controller variance for h_d dimension	0.2

Table 5.1: Nonlinear RL-Controller parameter set.

Further we have the following:

- We decrease the learning rate of the controller after the the training is finished. Although keeping the same learning rate worked fine for most of the consecutive tests, still for some cases the inflow became instable around the zero error which resulted in a final stationary error. This problem is caused by the critic's gradient information which the controller uses to adapt its weights. In the stable system this gradient is zero (or very close to zero). The high learning rate could move the critic out of its flat surface to a neighbouring region. As a result the gradient changes which causes the unstable inflow. Now if this new region is also flat then the inflow would become stable with the stationary error in the system. Choosing a smaller learning rate after the training resulted in good performance since the changes were not large enough to result into this unwanted gradient values. The results showed that the controller however, will continue adapting its weights, where needed, using a small learning rate.

- Remember that in the previous chapter we introduced a normal adaptation noise which when used during the training utilized the learning and caused the controller's weight to become stable. However, the nonlinear RL-Controller seems to be able to produce a stable inflow and there was no need to use the extra adaptation noise. But, we could also consider this noise as a simulation of an error caused by the measurement of height. In that case the results of the tests with such noise are given in Section 5.7.1.

5.5 No Disturbance and No Delay

The surface of the controller after the training is shown in Figure 5.3.

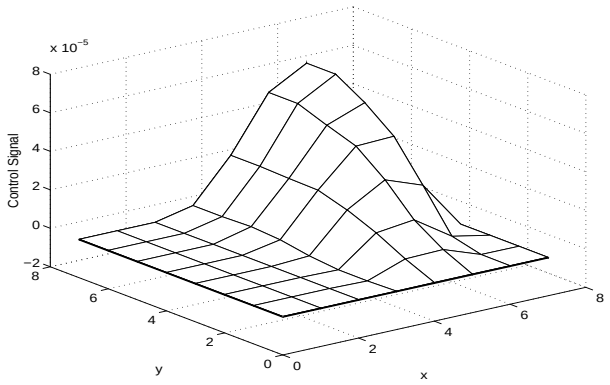

Figure 5.3: D^-LF^-: The surface of the controller after the training. The x axis denotes the error with $0cm$ at $x = 4$. The y axis denotes the destination height with $100cm$ at $y = 7$

Notice that the surface of the controller is clearly divided into two parts, one for positive and one for negative errors. Table 5.2 shows how the controller parameters are scaled to fit in each dimension of the cmac. The scaling is as you notice different for positive and negative errors. The reason for this is discussed in Section 5.8.

e(cm)	x
−10	0
0	4
100	7

h_d(cm)	y
0	0
50	3.5
100	7

Table 5.2: The scaling of the controller parameters.

Now let's see what this surface exactly means. The part corresponding to negative error is almost flat and close to zero showing that no water should be pumped in when the water level is high above the desired height, letting the height decrease fast, as long as possible. On the other hand the part for which the error is positive, results in large inflows when the water level is far below the desired height. On the positive side there is also a region that could never be trained. These are the set of impossible states where e is greater than h_d. Further, looking at the zero error-line ($x = 4$) we see that the controller produces, as expected, larger desired inflows as the desired height increases.

The result of the test with the reference signal is shown in Figure 5.4.

5.6 Delay

By training the controller as explained in Section 5.3 we get the surface in Figure 5.5.

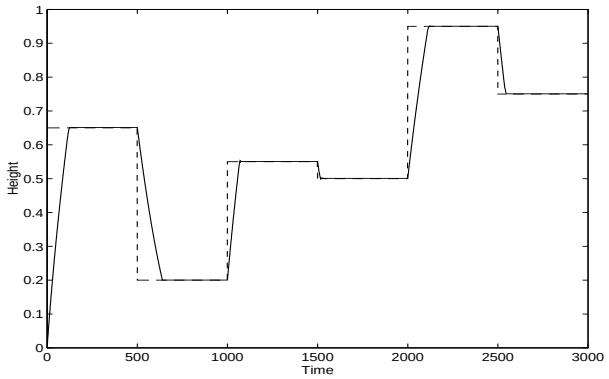

Figure 5.4: $D^{-}LF^{-}$: RL-Controller following the reference signal.

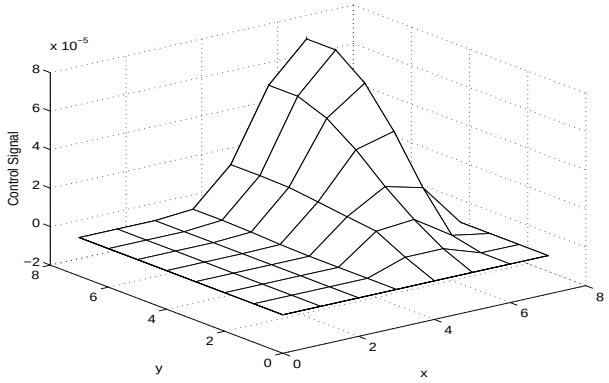

Figure 5.5: $D^{+}LF^{-}$: The surface of the controller after the training.

We already explained in the previous section what each part of the surface meant. But by looking closely at this surface we could see that it is slightly different from the trained controller with no delay, shown in Figure 5.3. First, the controller produces relatively larger inflows when the error is positive. The controller receives the delayed information about the change in the level of water. As a result it increases the inflow to increase the height, which due to delay it assumes to be low. Second, as the error gets closer to zero we see that relatively smaller inflows are produced comparing to the case without delay. This could be interpreted as the controller being aware of the delay and producing relatively smaller inflows as a result. This avoids the oscillation of the water level around the setpoint.

The result of the test with the reference signal is shown in Figure 5.6.

The results showed that although the controller might avoid the oscillation for some time by keeping the inflow low, on the other hand it will gradually increase the inflow to decrease the error which at some point would result in oscillation. In turns, oscillation increases the error and causes the inflow to decrease and etc. ...

5.7 Low-Frequent Disturbances

Now we add the disturbances on the outflow to the system of the trained controller with delay to see how it performs. The result of the test with the reference signal is shown in Figure 5.7. The system seems to be responding fast to the changes on the inflow. This shows that the surface of the controller is smooth enough to be able to react well to such variations on outflow. As the learning goes on the system is expected to improve its performance even more.

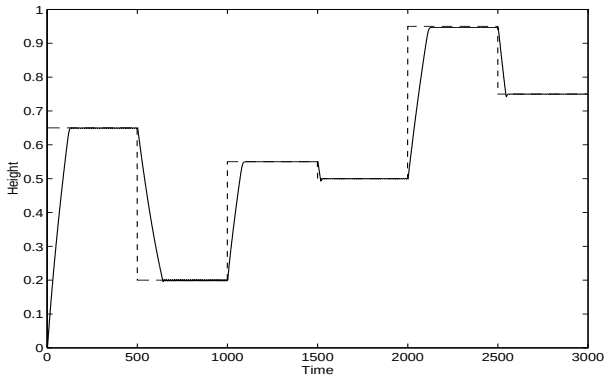

Figure 5.6: D^+LF^-: RL-Controller following the reference signal.

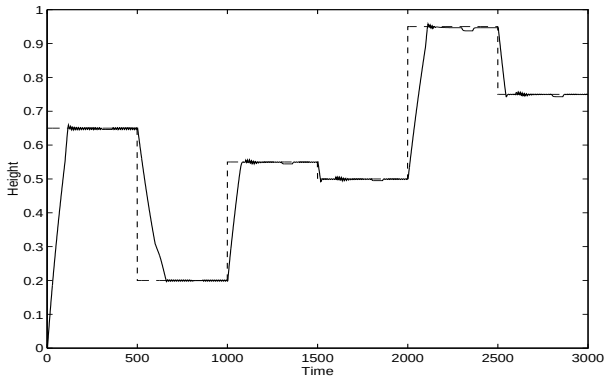

Figure 5.7: D^+LF^+: RL-Controller following the reference signal.

5.7.1 Adaptation Noise

As mentioned before in Section 5.4 we assume that a normal noise is presented on the destination height. The result of the test with the reference signal is shown in Figure 5.8. The height is kept close to the setpoint and the system stays stable when the noise is added to the height, resulting in a good tracking performance.

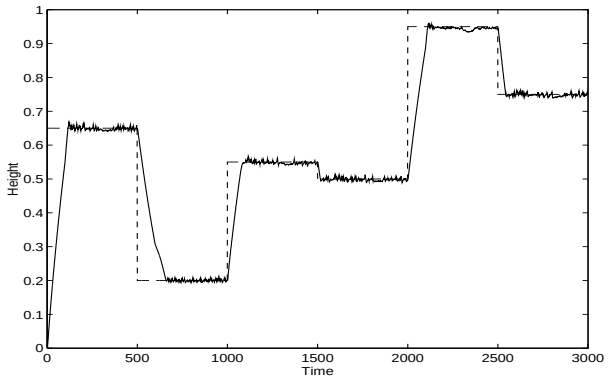

Figure 5.8: $D^+LF^+AN^+$: RL-Controller following the reference signal.

5.8 Scaling Problem

During the experiments it became clear that the local update of the weights of the controller is causing problems with generating optimal inflow signals. As a result of the local update of the weights on the positive error region also the weights on the negative error region close to zero error-line are changed, which is not always desirable. If these weights are increased too much, they would force the pump to start pumping water into the vessel when the error is negative and the height of the water is still *high* above the setpoint. Notice that optimally the inflow should stay zero as long as possible in a situation like this, allowing the water level to decrease in a faster speed. This early and high inflow was at some points so much that the height became stable somewhere on top of the desired setpoint.

Two natural solution to this problem could be:

1. Increasing the resolution of the CMACs to reduce this local update effect. An increase in the resolution of the CMACs enough to solve the problem would result in a large network taking too much time for the simulations.

2. Reducing the variance terms of the Gaussian share function (Section 3.7.3) along the error dimension to reduce this local update effect. This variance reduction would also result in slow learning since then a lot of training is needed. Besides, the local learning ability of the CMAC is then almost lost.

However, the limited time that was available to end the research of the nonlinear controller forced a different solution to the problem. Scaling positive and negative errors differently, giving the negative errors more weights than the positive ones. This did solve the problem of local update noise. Next the result of a sample experiment is presented with the equal scaling of errors of both signs.

Just to satisfy the curiosity, near the end of the project a small test was done to explore the possibilities of the first solution mentioned above. It was assumed that no delay or disturbances were present. The resolutions of critic and controller were both increased to 11 weights in each dimension. Now the error could be scaled such that each $10cm$ gets one weight and the range $-10cm \leq e \leq 100cm$ was used for the input error. We used the same parameter set as in Table 5.1 except for the learning rate of the controller which was increased to 5.0. After a long and slow training the controller surface was as shown in Figure 5.9.

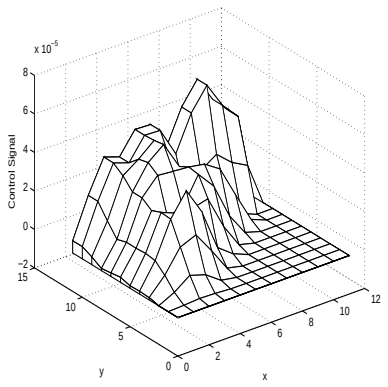

Figure 5.9: The surface of the controller after the training. The x axis denotes the error with $0cm$ at $x = 1$. The y axis denotes the destination height with $100cm$ at $y = 11$

Since the variance terms of the Gaussian share function are relatively small for this resolution, the surface is not very smooth. This also means that generalization would be poor in this case. Based on the previous experiments this problem could probably be solved by adjusting these terms.

5.9 Conclusion

Tables 5.3, 5.4 and 5.5 contain the RMS (root of the mean squared) height error for all three different cases under which the simulations took place. A simple comparison of these results

Controller	RMS
P	0.1052
PI	0.1051
RL (linear)	0.1053
RL (nonlinear)	0.1052

Table 5.3: $D^{-}LF^{-}$: RMS height errors.

Controller	RMS
P	0.1085
PI	0.1084
RL (linear)	0.1084
RL (nonlinear)	0.1088

Table 5.4: $D^{+}LF^{-}$: RMS height errors.

Controller	RMS
P	0.1086
PI	0.1085
RL(LF) (linear)	0.1088
RL(LF,AN) (linear)	0.1087
RL(LF) (nonlinear)	0.1086
RL(LF,AN) (nonlinear)	0.1087

Table 5.5: $D^{+}LF^{+}$: RMS height errors.

shows us that the nonlinear RL-Controller is able to perform at least as good as the linear RL-Controller and the P-Controller. Of course, as it was the case with the linear controller, these RMS errors could not be used as good basis for drawing conclusions about the performance of each type of controller. These just represent the performance over a limited testset.

What is probably safe to conclude is that a nonlinear controller could be designed that using an initial set of parameters as in Table 5.1 was able to adapt itself to different conditions such as delay and noise by adjusting those parameters by itself.

By looking at the controller surfaces in Figures(5.3,5.5) we could also see that local learning ability of the CMAC has made it possible for the controller to be generalizing well for the system states that it has never been in before resulting in a smooth surface.

Further, the simulations showed that the nonlinear RL-Controller improves its performance each time a setpoint is presented to it. This means a more optimal tracking of the reference signal under different conditions.

Chapter 6

Conclusion

6.1 Conclusion

We studied how a single water vessel system could be controlled using different RL-Controllers based on adaptive critic method of learning. The function approximations were all done using CMAC neural networks. We used RMS error in height to compare the results of different controllers.

First, we experienced with a simple RL-Controller in Section 4 in which the controller had only one weight. We saw that a trained critic could adjust the weight of the controller when initialized to zero. It was also shown that the RL-Controller is able to generalize to produce control signals for the parts of the state space not presented to it during training. Comparisons with the conventional controllers showed that the RL-Controller performed as well as P-Controller, but worse than PI-Controller. PI-Controller had the best results of all three controllers because it was able to achieve the zero error by summing up the error near the setpoint. Another important result was that the RL-Controller could adjust its parameters to adapt to conditions like delay, unlike the conventional controllers which had to be manually adjusted for each case.

Secondly, a nonlinear RL-Controller was designed, since the relationship between the control parameters and the water vessel state is also a nonlinear one. The critic and the controller were both trained together and then tested with a reference signal. The RL-Controller was able to adjust its weights to adapt to delay in order to avoid the oscillation of the water level around the setpoint. It was also shown that the zero error could be achieved by this nonlinear controller, assuming that the zero error is considered as a very small error which is continuously decreasing. A smooth surface of the trained controller also made generalization in the state space possible.

These results do indicate that reinforcement learning is a promising method of controlling the water vessel process. However, the research is still in its early stages and to prove the superiority of the RL-methods over the conventional controllers a lot more research and tests must be done. Some of the possible future work based on the results of these research will be mentioned next.

6.2 Future Work

Of course the final goal of the water vessel project is to study the design of a neurocontroller for the dual cascaded water vessel system shown in Figure 1.1 using reinforcement learning. In order to do this, future work could be put into three different categories.

1. *Single Vessel*: Looking back at the experiments there are some things that could be tried out for the single vessel process.

    - In Section 5.8 we already discussed how the unequal scaling of the error in the nonlinear RL-Controller could be avoided by changing the resolution of the CMAC or selecting other values for the variance terms of the Gaussian share function.

- A totally different approach would be to exclude the negative errors from the controller as we did in the linear controller of Section 4. Then we follow the common sense solution of setting the pump activity to 0 when the error is negative. This idea is easier to implement and does not have the scaling problem.

  However, this approach might cause problems when small overshoots result in negative errors around the setpoint. The small increases of the inflow around the setpoint could cause the error to become negative, setting inflow to zero. The system then waits till the error becomes positive. But, then it might produce the same inflow that caused the overshoot before and etc. If the controller does not decrease the inflow, it will result in the oscillation of the water level around the setpoint,

2. *Intermediate System*: Once the control problem with the single vessel is done, it will be a good idea to make the problem only a bit harder and not make a big step to the dual cascaded system. In this case we could experiment on a system consisting of two vessels, one on top of another, such that the outflow of the top vessel is the inflow of the vessel under it. The goal will then be to control the level of water in the lower vessel. The functions critic and controller could then take the following forms:

$$\text{critic} = J(e, h_{desired}, h_{top}, u), \quad \text{controller} = u(e, h_{desired}, h_{top})$$

   h_{top} is the height of water in the top vessel and is used since it is an indication of the amount of inflow to the lower vessel.

3. *Dual Cascaded System*: When the dual system of vessels is also controlled optimally we could get to the real business of dual cascaded system. Solutions could be grouped into two categories:

   Parallel Control in which two controllers could be designed for each pump. These controllers will be then sharing one critic which views the whole system and provides both controllers with the information they need to improve their performances. This is more like a case in which a teacher is looking at the whole system and coordinating the behaviour of each pump accordingly.

   Distributed Control in which each pump has its own critic and controller. The controllers will be then trying to become experts in their tasks and at the same time cooperate with each other through the reinforcement signals they receive.

   The former is more of a *coordinated control* and the latter *cooperated control*. In a recent work, Crites & Barto [6] showed that the parallel control results in better performance when applied to an elevator dispatching problem.

   In many real-world tasks involving real people, coordination is usually regarded more important than cooperation, but is it also the case with the autonomous agents?

Bibliography

[1] J.S. Albus. A new approach to manipulator control: The cerebellar model articulation controller (cmac). *Transactions of the ASME, Journal of Dynamics Systems Measurement and Control*, 97:220–227, 1975.

[2] A.G. Barto. Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology*, 4:229–256, 1985.

[3] A.G. Barto. Reinforcement learning and adaptive critic method. In *Handbook of Intelligence control, Neural, Fuzzy and Adaptive approaches*, pages 469–491. Van Nostrand Reinhold, New York, 1992.

[4] A.G. Barto, R.S. Sutton, and C.J.C.H. Watkins. Neuronlike adaptive elements that can solve difficult learning problems. *IEEE Transactions*, 13:834–846, 1983.

[5] D. Burghes and A. Graham. *Introduction to control theory including optimal control.* Ellis Horwood Ltd., Publishers, West Sussex, 1980.

[6] R.H. Crites and A.G. Barto. Improving elavator performance using reinforcement learning. University of Massachusetts, Computer Science Department, 1995.

[7] T.J.J. Van den Boom. *MIMO system identification for H_∞ robust control*, pages 161–179. Delft University of Technology, Department of Electrical Engineering, 1992.

[8] L. Dorst. *An introduction to robotics for the computer science.* University of Amsterdam, Department of Computer Science and Mathematics, 1992.

[9] R.F. Drenick and R.A. Shahbender. Adaptive servomechanisms. *AIEE Transactions*, 76:286–292, 1957.

[10] J. Van Amerongen et al. *Regeltechniek (dutch).* Open Universiteit, 1992. ISBN 90-358-0705-7.

[11] V.W. Eveleigh. *Adaptive control and optimization techniques.* McGraw-Hill Book Company, New York, 1967.

[12] S. Haykin. *Neural Networks- A Comprehensive Foundation.* Macmillan Publishing Company, 1994.

[13] J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the theory of neural computation.* Addison Wesley, 1991.

[14] S.S. Keerthi and B. Ravindran. A tutorial survey of reinforcement learning. Indian Institute of Science, Department of Computer Science and Automation, Bangalore, 1994.

[15] J.R. Leigh. *Essentials of nonlinear control theory.* Peter Peregrinus Ltd., London, 1983.

[16] W.T. Miller, R.P. Hewes, F.H. Glanz, and L.G. Kraft. Real time control of an industrial manipulator using a neural network based learning controller. *IEEE Journal of Robotics Research*, 6.2:84–98, 1987.

[17] M. Minsky and S. Papert. *Perceptrons: An introduction to Computational Geometry.* The M.I.T. Press, 1969.

[18] M.L. Minsky. Steps toward artificial intelligence. In *Proceedings of Institute of Radio Engineers*, pages 8–30, 1961.

[19] K.S. Narendra and A.M. Annaswamy. *Stable Adaptive Systems.* Prentice-Hall Inc., Englewood Cliffs, N. J. 07632, 1989.

[20] F. Rosenblatt. *Principles of Neurodynamics.* Spartan Books, New York, 1959.

[21] D.E. Rumelhart and J.L. McClelland. *Parallel Distributed Processing: Exploration in the Microstructure of Cognition.* The M.I.T. Press, 1986.

[22] G. Schram, L. Karsten, B.J.A. Kröse, and F.C.A. Groen. Optimal attitude control of satellites by artificial neural networks: a pilot study. Technical Report CS-94-05, Department of Computer Systems, Faculty of Mathematics and Computer Science, University of Amsterdam, 1994.

[23] R.S. Sutton. *Temporal credit assignment in reinforcement learning.* PhD thesis, University of Massachusetts, Amherst, 1984.

[24] R.S. Sutton. Learning to predict by the method of temporal difference. *Machine Learning*, 3:9–44, 1988.

[25] E.L. Thorndike. *Animal Intelligence.* Hafner and Darien, CT, 1911.

[26] M.D. Waltz and K.S. Fu. A heuristic approach to reinforcement learning control systems. *IEEE Transactions on Automatic Control*, 10:390–398, 1965.

[27] C.J.C.H. Watkins. *Learning from delayed rewards.* PhD thesis, Cambridge University, Cambridge, 1984.

[28] C.J.C.H. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8:279–292, 1992.

[29] P. Werbos. Approximate dynamic programming for real-time control and neural modelling. In *Handbook of Intelligence control, Neural, Fuzzy and Adaptive approaches.* Van Nostrand Reinhold, New York, 1992.

[30] D.A. White and D.A. Sofge. Applied learning: Optimal control for manufacturing. In D.A. White and D.A. Sofge, editors, *Handbook of Intelligence control, Neural, Fuzzy and Adaptive approaches.* Van Nostrand Reinhold, New York, 1992.

[31] D.A. White and D.A. Sofge. Cmac learning method. In D.A. White and D.A. Sofge, editors, *Handbook of Intelligence control, Neural, Fuzzy and Adaptive approaches*, pages 269–273. Van Nostrand Reinhold, New York, 1992.

[32] D.A. White and D.A. Sofge, editors. *Handbook of Intelligent control, Neural, Fuzzy and Adaptive approaches.* Van Nostrand Reinhold, New York, 1992.

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