THESIS

CONTINUOUS REINFORCEMENT LEARNING FOR FEEDBACK CONTROL SYSTEMS

Submitted by
Jilin Tu
Computer Science Department

In partial fulfillment of the requirements
for the Degree of Master of Science
Colorado State University
Fort Collins, Colorado
Summer 2001

ABSTRACT OF THESIS

CONTINUOUS REINFORCEMENT LEARNING FOR FEEDBACK CONTROL SYSTEMS

To achieve the best control performance, feedback control techniques require a very accurate mathematical model for the plant, which is not possible in many real problems. Reinforcement learning algorithms, on the other hand, are able to explore and learn to improve control performance without knowing the plant model in advance. Therefore, applying reinforcement learning techniques to feedback control systems could redeem the loss of performance of feedback control caused by the imprecision of plant models.

This thesis studies how to integrate statespace models of control systems with reinforcement learning and analyzes why one common reinforcement learning architecture does not work for control systems with Proportional-Integral (PI) controllers. As many control problems are best solved with continuous state and control signals, a continuous reinforcement learning algorithm is then developed and applied to a simulated control problem involving the refinement of a PI controller for the control of a simple plant. The results show that a learning architecture based on a statespace model of the control system outperforms the previous reinforcement learning architecture, and that the continuous reinforcement learning algorithm outperforms discrete reinforcement learning algorithms.

Jilin Tu
Computer Science Department
Colorado State University
Fort Collins, Colorado 80523
Summer 2001

ACKNOWLEDGEMENTS

This research was supported by Grant number CMS-9804747 from the National Science Foundation.

I would like to thank my academic advisor Prof. Charles W. Anderson, who spent a lot of time helping me when I met difficulties in research and gave me a lot of directions on the thesis. Without his help, I can not imagine how to complete this research. I also thank the other Advisory Committee members, Prof. Peter Young who helped me a lot to understand the control theories and gave me insight in the feedback control, and Prof. Ross Beveridge who helped me a lot on how to write a thesis. I also thank Prof. Doug Hittle for his generous help on my research during the group meetings.

I am grateful to Prof R. Matthew Kretchmar, whose previous work on reinforcement learning was my starting point. I am also grateful to Mr. Michael Anderson for being an enormous help when I was starting this research, and for taking the time to explain the control theories to me and giving me tips on how to further my research. I also want to thank my friends Lingzhi Yang, JiangJiang Yang. Discussions with them really deepened my understanding of control and Markov theory, respectively.

At last, I want to express my gratitudes to my family, my father Shiyong Tu, my mother Shuixiang Chen who died of cancer in 1993, my stepmother Yiping Hong, my elder brother Jihong Chen, and my elder sister Jimei Tu. without their support I would not have the opportunity to study in Colorado State University. I owe them a lot.

Thanks to all for everything!

DEDICATION

This thesis is dedicated to my family, they have always been there to support and encourage me.

谨以此文献给我的家人，

没有你们的支持就没有我的今天。

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

1.1 Motivation

A feedback control system is composed of two parts: a plant and a controller. The controller takes the difference between the reference signal and the plant output as input and provides control signals to the plant in an attempt to force the plant output to track the reference signal. A nominal feedback control system is shown in Figure 1.1. In this thesis, we only consider this tracking problem for a feedback control system, where the value of a reference signal is to be matched by the measured plant output as closely as possible.

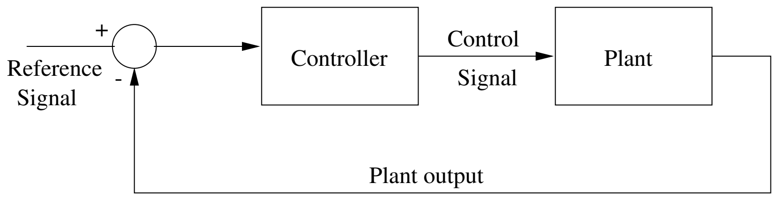

Figure 1.1: A nominal feedback control system

When designing control systems, it is important to acquire accurate a priori knowledge of the plant, otherwise the controller could take improper actions that could destabilize the system, and possibly result in damage to equipment or injury to people. In such situations, stability is equally important, or sometimes more important, than performance.

However, most of the time, it is very difficult to acquire a precise model for the physical plant, because the plant could be time-varying, non-linear, or have high-order dynamics. As computation resources are limited and current control theories usually can only handle linear and time-invariant models, control engineers have to build the model simple enough to meet the limitations but accurate enough to meet the performance requirements.

To ensure the stability of a control system, uncertainties caused by model simplifications have to be analyzed and considered during the controller design. In robust control, bounds of uncertainties are estimated. A controller design based on the knowledge of these bounds can maintain stable control of the system in all cases. Such a design approach limits the aggressiveness of the controller, and thus could result in suboptimal control performance (Doyle et al., 1991).

It is well known that reinforcement learning (RL) agents can learn optimal behaviors in an unknown environment through on-line, trial and error exploration when certain conditions are satisfied(Sutton & Barto, 1998). Therefore, using reinforcement learning techniques as part of a controller could be an approach to regaining the loss of performance caused by conservative design.

One approach is to replace the traditional controller completely with a reinforcement learner. Kavehercy (1996) applied this approach to a water vessel process control problem. The experiments show that the RL-controller can do as well as a proportional controller, but worse than a proportional-integral controller. However, it is apparent that such an approach cannot prevent the RL-controller from taking some improper actions that could damage the hardware, as exploring every state in the system is a theoretical requirement of learning. Therefore, this approach is not for industrial application.

Morimoto and Doya (2000) further proposed “robust reinforcement learning”. Similar to the robust design of a traditional controller, the uncertainty of the plant model is taken into account as the norm of the output deviation and the norm of the disturbance while learning the best control performance. Theoretically, this approach can guarantee the robustness of the control system after the successful training of RL-controller. Their results show that this approach achieved robust control performance even when plant parameters varied from their assumed values. Standard RL controllers, however, cannot handle such situations. Though an approach along this direction will learn robust control generally, the disadvantage of this approach is that the stability of the system during training is still not guaranteed, which would limit its wide application in industry.

Another approach is to use reinforcement learning to tune the parameters of a traditional controller to the best performance. Howell (2000) successfully applied this approach to a vehicle engine idle speed control problem. However, the best performance that the learning can reach is limited by the dynamics of the controller.

In a paper by Anderson et al. (1996) and the Ph. D. dissertation of Matthew Kretchmar (Kretchmar, 2000), a new reinforcement learning architecture was proposed and studied. As shown in Figure 1.2, the learning agent takes the tracking error as input and outputs an action signal which is added to the output of the controller. The summed control signal is then applied to the plant. In addition, stability analysis is applied to the actor of the learning agent so that the output of the learning agent is guaranteed not to disturb stability. This control system works fairly well in the experiments: the controller defines the conservative framework of stable control, and the learning agent refines the control and improves the control performance. Therefore, the advantages of this approach are that the exploration by the learning agent is guaranteed not to disturb the stability, and that the best learning performance is not limited by the dynamics of controller.

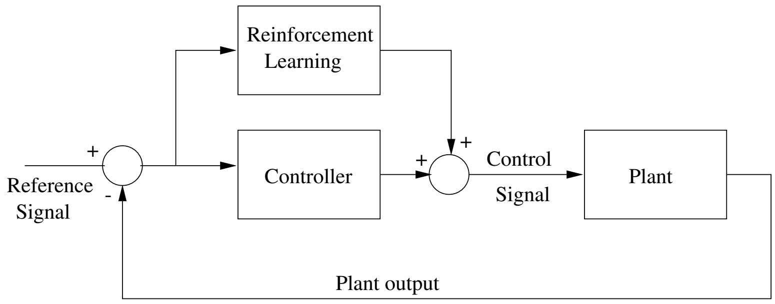

Figure 1.2: Reinforcement learning for feedback control system

However, the controller employed in Kretchmar’s thesis was a simple proportional controller. When we applied this learning architecture to a control system with a proportional-integral (PI) controller, we found that the learning agent did not learn the expected optimal actions. One objective of this thesis is to find the cause of this problem.

The state transition in a control system is continuous, and the control output of the RL-controller should also be continuous. However, the fundamental theories of traditional reinforcement learning, such as the temporal-difference (TD) theory, are based on discrete-time, discrete-state assumptions. When applying these theories to control problems, the time, state space, and action have to be discretized appropriately. Pareigis (1997) proposed an algorithm to choose the grids for discretization of time and space adaptively. However, no matter how the discretization of a discrete learning algorithm is refined, it is always cumbersome to apply discrete theories to continuous systems, either because of the curse of dimensionality or because of the inaccuracy of the discretized model of learning. The curse of dimensionality can be somewhat reduced by using functional approximation techniques. However, there is no theoretical guarantee of convergence of learning with function approximation in the RL community (Boyan & Moore, 1995; Sutton, 1996). The learning convergence problem with global function approximators, such as artificial neural network (ANN) with sigmoid actuation functions, is especially severe. Thrun (1993) summarized the cause of the problem as over-estimation due to functional approximation noise and proposed several possible solutions. Weaver et al. (1998) identified interference, the phenomenon that learning in one area of the input space causes unlearning in another area, as the cause of the problem and proposed a localizing training algorithm to penalize such unlearning with a measure of interference. However, all the improvements can only decrease the severity of this problem, they cannot eliminate it. There is still no theory that can guarantee the convergence of learning with function approximation in general.

To cope with this continuous learning problem theoretically, many approaches were proposed to extend the reinforcement learning theory. Gullapalli (1990) proposed a Stochastic Real-Valued Units that can learn continuous outputs using a connectionist network. Baird (1993) extended the Q-learning algorithm to continuous learning problems and proposed the advantage updating algorithm. Continuous temporal-difference theory was developed by formulating the continuous value surface updating process using a Hamilton-Jacobi-Bellman equation in (Bradtke & Duff, 1994; Munos, 1997b; Munos, 1997a; Doya, 2000).

In Kretchmar's dissertation, a discrete learning algorithm based on a CMAC functional approximator was employed. To enhance the learning capability of his learning architecture, we need to develop continuous reinforcement learning algorithms.

Therefore, the first motivation of this thesis is to explain why the above learning architecture does not work for a control system with a PI controller, and if possible to develop some rules for the design of reinforcement learning architecture in more complex control system. The second motivation is to develop a continuous reinforcement learning algorithm that can outperform a discrete reinforcement learning algorithm in a continuous control system.

1.2 Problem Details

Figure 1.3 shows the Actor/Critic learning architecture for a simple control system utilized in Kretchmar's dissertation. The controller is a P controller, and the plant can be represented by a discrete statespace model,

$$\begin{bmatrix} x(t+1) \\ y(t) \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x(t) \\ u(t) \end{bmatrix},$$

where $x(t)$ is the state, $u(t)$ is control signal and $y(t)$ is output. The difference between the plant output and the reference output is sent to both the proportional controller and the Actor/Critic learning agent. The sum of the learning agent's output and the controller's output is sent to the plant as the control signal. Within the Actor/Critic learning agent, the Critic learns a prediction of a sum of future error magnitude, given the current tracking error and action as input. Based on its prediction, it estimates a best action. The Actor takes the current error signal and the estimation of current best action and outputs an action that approximates the action estimation, which is regarded as the output of Actor/Critic learning agent.

Suppose the plant is an first-st order air-conditioning system with $A = B = C = 1$ and $D = 0$, Figure 1.4 shows the result after the learning agent has been trained. The plant output Tao is plotted both with and without the learning agent. The tracking error between the plant output and the reference signal is largely reduced by the learning agent after learning.

In the above simple control system, the proportional controller is good enough to define the framework of control. When we apply a proportional controller to a more complex plant, steady state error cannot be eliminated. Such steady state error could be so large that a proportional-integral (PI) controller becomes a necessity.

However, we noticed a learning problem when a PI control is utilized. The learning agent never stabilizes to a set of optimal control actions even when the input to the learning agent is almost the same. The output of the learning agent keeps drifting, however this achieves almost the same control performance for the same input. A typical set of control sequences that the Actor/Critic learns to apply to the system is shown in Figure 1.5. From the figure, we notice that a constant value added

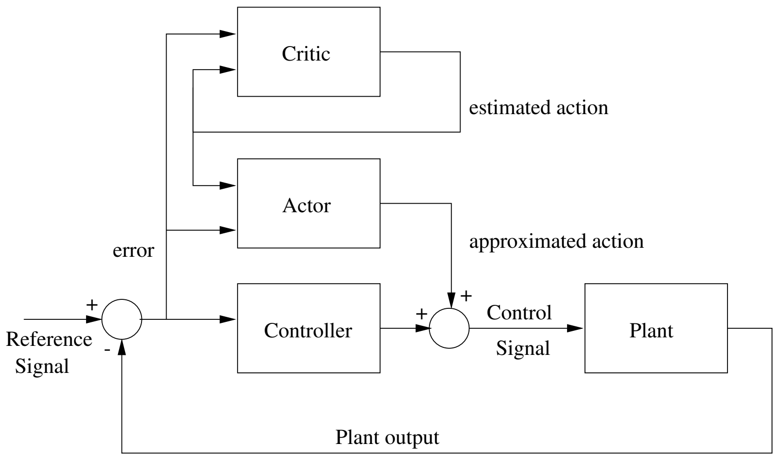

Figure 1.3: Actor/Critic architecture in (Kretchmar, 2000)

to the control sequence of the learning agent does not make a big difference to control performance. The negative effect of this constant input is eliminated by the integral term of PI controller. This makes the optimal control sequence learned by the learning agent non-stationary. Each time noise is injected into the system, the learning agent settles down to a slightly different control sequence. In other words, the learning does not converge.

Besides, we notice that there are high frequency oscillations in the agent control sequence which are caused by the discrete nature of the reinforcement learning algorithm. Such oscillation is not acceptable in industrial applications, therefore should be eliminated.

1.3 Objectives

The primary objective of this work is to identify the cause of the learning problem of the current learning architecture based on theories of reinforcement learning and control and to develop a reinforcement learning strategy that can avoid this problem. As a second objective, we will develop continuous reinforcement learning algorithms and apply them to the control system. Reinforcement learning algorithms based on continuous state and action should outperform the discrete reinforcement learning in situations involving continuous control systems and state transition. To verify this, we will do experiments to compare their performance.

Outcomes of this thesis are

- an explanation of the learning problem of the previous learning architecture,
- a new learning architecture that can avoid the learning problem,
- continuous learning algorithms applied to the control system with an analysis of the results, and
- experimental results that verify that the new learning architecture outperforms the previous one, and that the continuous learning algorithm has an advantage over current discrete learning algorithm.

1.4 Overview of the Thesis

This thesis is a synthesis of research on reinforcement learning and feedback control. Chapter 2 reviews the theories related to feedback control and reinforcement learning. For the control theories, we explain the concepts on feedback control, PI control and statespace models. For reinforcement

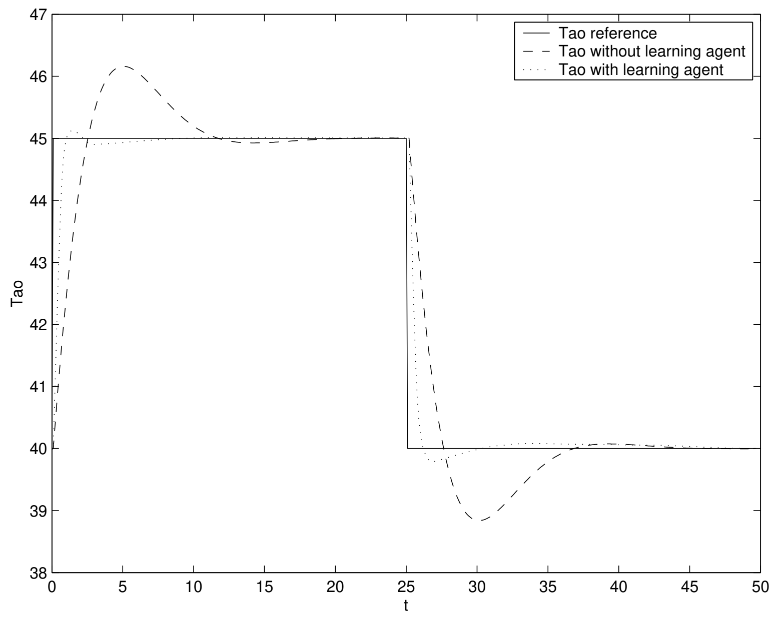

Figure 1.4: Comparison of reference signal and plant outputs (Tao) with and without learning

learning, we review the Markov properties, basic ideas of discrete reinforcement learning, and continuous reinforcement learning approaches.

Chapter 3 first analyzes the learning problem based on reinforcement learning theories and experimental findings. An analysis of the statespace model of the control system leads us toward a new learning architecture that can avoid the known learning problem. Then, the details on how to apply the continuous reinforcement learning algorithm are presented.

In Chapter 4, we present three sets of experimental results. The purpose of the first experiment is to verify the advantage of the new learning architecture over the previous one. The second experiment shows how the continuous reinforcement learning algorithms are applied to a simple control system with only a proportional control. The third experiment will apply the continuous learning algorithm to the new learning architecture, and compare the performance of the discrete learning algorithm and the continuous ones in this new learning architecture for the same control system.

Chapter 5 analyzes and summarizes the findings in this thesis, discusses the successes and difficulties of our approach, and finally suggests future work based on current work.

Appendix A provides a brief introduction of the object-oriented design of our learning system software.

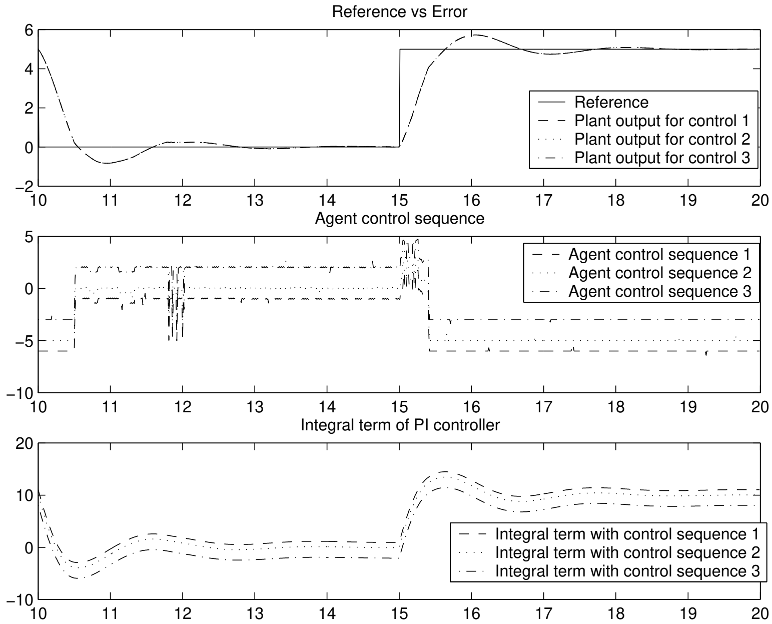

Figure 1.5: A set of control sequences of learning agent

Chapter 2

Theoretical Background

This chapter reviews the theoretical background related to our research. We first review the nominal control theories, then PI control and statespace models are analyzed. These are the control theories we need to understand this thesis. The traditional reinforcement learning theories are then discussed, and the foundation of continuous reinforcement learning theories are presented at the end.

2.1 Control theory

2.1.1 Feedback Control

A control system is composed of at least two parts, a controller and a plant. The purpose of the controller is to drive the plant output to a desired reference signal. The simplest version of a control system is shown in Figure 2.1. The controller takes the reference signal as input, and produces a control signal that drives the plant to output the desired reference value.

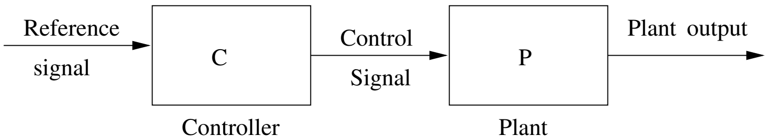

Figure 2.1: A feedforward control system

It is convenient to design a control system in this way mathematically. If we have an accurate model of the plant, we can design a controller with a transfer function that is the inverse of the plant transfer function, $C = P^{-1}$. This technique is called *driving*.

The driving technique works well if the plant model is accurate and there is no noise in the system. This requirement, however, in reality is impractical. To overcome the negative influence of noise and uncertainty of the physical control system, we need to introduce a feedback control mechanism.

In a feedback control system, the output of the plant is compared to the reference signal. The error is fed into a controller which then sends a control signal to the plant. The whole process is illustrated in Figure 2.2, where $r(t)$ is the reference signal, $o(t)$ is the plant output, $e(t)$ is the tracking error, and $u(t)$ is the control signal.

Suppose the transfer function of the controller and the plant are C and P, respectively. We can define the *loop transfer function* as $L = PC$. The *sensitivity function*, which is the transfer function from reference input r to tracking error e is

$$S = \frac{1}{1+L} .$$

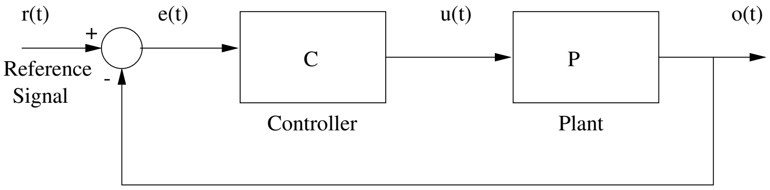

Figure 2.2: A typical feedback control system

The transfer function from reference input r to plant output o is

$$T = \frac{L}{1+L} .$$

A controller C that maintains stable control of such a nominal single-input-single-output control system must meet the *Nyquist Stability Criterion* (Miron, 1989). To maintain robustness of the control system and overcome the negative effects of uncertainty in the plant model, robust design techniques should also be applied, such as loopshaping, etc., which, however, may worsen the control performance (Doyle et al., 1991; Skogestad, 1996). In this thesis, we suppose we already have a robust controller, and the objective is to enhance the controller performance using reinforcement learning techniques without disturbing system stability. Therefore, we will not further discuss these robust design topics as they are not strongly related to our objectives.

2.1.2 PI Control

The control law for a Proportional-Integral (PI) controller is as follows:

$$u = K_p e(t) + K_i \int_0^t e(t)dt ,$$

where $e(t) = r(t) - o(t)$, K_p is the weight for the proportional term, and K_i is the weight for the integral term. In the Laplace Domain, the transfer function of a PI controller is

$$C = K_p + K_i \frac{1}{s} .$$

For an arbitrary plant P, we can derive the transfer function of the feedback control system as

$$\begin{aligned} T &= \frac{L}{1+L} \\ &= \frac{PC}{1+PC} \\ &= \frac{P(K_p s + K_i)}{s + P(K_p s + K_i)} . \end{aligned}$$

It is apparent that when the system reference input is constant ($s \to 0$), we have $T = 1$, which means that the plant output will eventually approximate the reference input of the control system as long as the system is stable and there is no disturbance. Therefore, a PI controller is a generally applicable controller as long as the control performance is not an important issue.

If $K_i = 0$, the PI controller is turned into a P controller. A P controller, however, cannot eliminate the steady state error, though an infinitely large K_p could make the steady state error approximate zero theoretically, which is explained by the sensitivity function of feedback system:

$$\begin{aligned} S &= \frac{e(s)}{r(s)} \\ &= \frac{1}{1+PC} \\ &= \frac{1}{1+K_p P} . \end{aligned}$$

In Kretchmar's dissertation (Kretchmar, 2000), reinforcement learning is successfully utilized to improve the performance of a P controller. However, we observed that its application to a PI controller was a failure. In Chapter 3, the cause of this failure is explained and a new learning strategy is proposed.

2.1.3 Statespace Models

The statespace model for a system is defined as follows (Jobling, 1999). Suppose a system has n state variables $(x_1, x_2, ..., x_n)$, q inputs $(u_1, u_2, ..., u_q)$, and r outputs $(y_1, y_2, ..., y_r)$. We have

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{2.1}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \tag{2.2}$$

where

$\mathbf{x} = [x_1, x_2, ..., x_n]^T$,
$\mathbf{u} = [u_1, u_2, ..., u_q]^T$,
$\mathbf{y} = [y_1, y_2, ..., y_r]^T$,
$\mathbf{A}_{n\times n}$ is *system matrix* ,
$\mathbf{B}_{n\times q}$ is *input matrix* ,
$\mathbf{C}_{n\times r}$ is *output matrix* , and
$\mathbf{D}_{r\times q}$ is *feedforward matrix* .

Although a dynamic system represented by a statespace model is unique, its system transfer function can be transformed into many different statespace models. If a statespace model is developed from an analysis of the elementary dynamics of the physical system, each internal state of the model will have physical meaning. However, often it is more convenient to transform the models into some canonical forms, such as Companion Form, Controller Canonical Form, Observer Canonical Form and the Normal form (Jobling, 1999).

Matlab (from Mathworks, Inc) transforms system transfer functions into the Controller Canonical Form, which is described as follows. For a system transfer function

$$G(s) = \frac{b_{n-1}s^{n-1} + b_{n-2}s^{n-2} + ... + b_0}{s^n + a_{n-1}s^{n-1} + a_{n-2}s^{n-2} + ... + a_0} ,$$

the Controller Canonical Form of the statespace model is

$$\begin{aligned}
\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_{n-1} \\ \dot{x}_n \end{bmatrix} &= \begin{bmatrix} -a_{n-1} & -a_{n-2} & \cdots & -a_1 & -a_0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} u , \\
y &= \begin{bmatrix} b_{n-1} & b_{n-2} & \cdots & b_1 & b_0 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & \cdots & x_{n-1} & x_n \end{bmatrix}^T .
\end{aligned} \tag{2.3}$$

In this thesis, we will use Matlab to create the Controller Canonical Form statespace model for feedback control systems. We will also create a statespace model whose states have physical meanings according to system dynamics analysis.

2.2 Reinforcement Learning

2.2.1 Introduction

Depending on what information of the environment is provided to the learning agent, learning methods can be grouped into three categories: supervised learning, unsupervised learning and reinforcement learning. In supervised learning, the desired input/output pairs for the learning agent in the environment are provided as successful examples explicitly. Therefore supervised learning can be reduced into a function approximation problem: how to approximate the functional mapping between the input and output pairs that can be generalized for estimating the best output given an input that is not in the training set. This approach has been studied by many research fields including machine learning, statistical pattern recognition, artificial neural network, etc. Its application to real world learning problems has been a success as long as the desired input/output training set can be provided. Unsupervised learning, on the other hand, is based on no information from the environment except the data itself. A model is discovered by analyzing the data according to some rules and is utilized in future application. For example, the data could be classified into several classes that are not predefined, based on the criterion of maximal between-class difference and minimal within-class difference. This technique is applicable to some situations where the training data is difficult to generate. However, its advantage is also its shortcoming: it is difficult to utilize the available prior knowledge of the environment to improve the learning performance.

Reinforcement learning is a compromise between supervised learning and unsupervised learning. It learns to map situations to actions via online trial-and-error exploration. Instead of providing desired input/output pairs for training, or no information other than the data, the environment provides a feedback to the learning agent on how good the agent's outputs are after the learning agent interacts with the environment. The feedback is called a *reward*, or *reinforcement*. The evaluation of actions might be done after some delay. In a chess game, the moves could be evaluated only at the end of the game. A '1' reinforcement signal could be fed back to the learning agent for a victory, or a '0' for the loss. It is up to the learning agent to figure out which moves account for this ending (Russell & Norvig, 1995). Therefore, the philosophy behind this learning strategy is similar to models of animal learning from reinforcement stimuli.

By taking random moves and evaluating the rewards, a reinforcement learning agent can eventually create a predictive model of the environment and learn to take the wise moves that lead to victory. The process of taking random moves is called *exploration*. To refine the predictive model that can generate the best actions, the learning agent should also prefer those actions that were tried and produced the most rewards. This process is called *exploitation*. A successful reinforcement learning algorithm must find a way to balance these two processes. Ideally, we would prefer a learning agent that explores the state space heavily at the beginning, but relies on exploitation more and more progressively (Sutton & Barto, 1998).

In order to model the environment based on reinforcement information, an estimate of a sum of reinforcement signals is updated inside the learning agent after each time an action is taken. One of the most popular summations is the so-called *infinite-horizon discounted model*

$$E(\sum_{t=0}^{t=\infty} \gamma^t r_t) \ ,$$

where E is an expectation operator and $\gamma < 1$. This accumulation model sums up the future rewards with future rewards discounted geometrically.

By discretizing the state space, this reward evaluation is carried out for the discrete state that the agent is currently in after each step of taking actions. The current strategy of taking actions is called the *policy*, denoted as π, which maps the states $\mathbf{s}$ to actions $\mathbf{a}$ with probability $\pi(\mathbf{s}, \mathbf{a})$.

The expectation of future rewards at each state forms a surface in state space, which is called the *value surface*. A more accurate formulation of the calculation of the value surface is as follows (Sutton & Barto, 1998).
For the current policy π, the value $V^\pi(\mathbf{s})$ at state $\mathbf{s}$ is defined as

$$V^\pi(\mathbf{s}) = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | \mathbf{s}_t = \mathbf{s}\}$$

After the value surface is updated, it can better estimate the reward for each state, based on which a better policy π_{t+1} can be produced. A better policy could then help to obtain a better value surface. Under assumptions described later, iterations of this updating loop will lead to optimal actions.

Assuming the learning process is a Markov Decision Process and assuming there is an accurate mathematical model of the environment, the process of updating $V^\pi(\mathbf{s})$ can be formulated as a dynamic programming recursive model:

$$\begin{aligned} V^{\pi}(\mathbf{s}) &= E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | \mathbf{s}_t = \mathbf{s}\} \\ &= E_{\pi}\{r_{r+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | \mathbf{s}_t = \mathbf{s}\} \\ &= E_{\pi}\{r_{t+1} + \gamma V^{\pi}(\mathbf{s}_{t+1}) | \mathbf{s}_t = \mathbf{s}\} , \end{aligned} \tag{2.4}$$

where the E_{π} is an expectation operator based on the stochastic policy π and the state transition probabilities that are assumed known. The learning problem can then be solved using dynamic programming methods.

Unfortunately, it is impractical to have an accurate model of state-transition probabilities in advance. The agents have to learn the value surface by interacting with the environment directly, instead of using a known model. To accommodate this need, Monte Carlo methods were proposed. The sequence of state, action and reward are recorded while the agents are interacting with the environment. Monte Carlo methods can learn from these recorded sequences after each episode of execution.

The shortcoming of Monte Carlo methods is that they are not step-by-step incremental learning. The executions of the first few episodes of a task could take forever as no information of the environment is fed back to the agent to guide the execution.

By combining the ideas of dynamic programming and Monte Carlo methods, the *Temporal-Difference Learning (TD)* algorithms described below were proposed to avoid the above shortcomings. In other words, the TD methods can learn directly from raw experience incrementally without a model of the environment. This method has become one of the central ideas in reinforcement learning.

2.2.2 Markov Decision Processes

The goal of reinforcement learning is to learn a set of optimal policies that map the current states to optimal actions. To make sure such a mapping exists, it is important that the states contain all relevant information for making decisions. This property of a state is called the *Markov property.*

The Markov property can be defined mathematically as follows. Denote the state, action and reward of a decision process at time t as (s_t, a_t, r_t), for $t = 0, 1, 2, \ldots$. The state has the Markov property if and only if the probability that the state is s' and the corresponding reward is r' at time *t+1* satisfies

$$P\{s_{t+1} = s', r_{t+1} = r' | s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0, a_0\} = P\{s_{t+1} = s', r_{t+1} = r' | s_t, a_t\} \tag{2.5}$$

for all possible s' and r'.

A decision process whose state satisfies the Markov property is called a *Markov decision process* (MDP). That a learning task is a Markov decision process is one of the most important assumptions in reinforcement learning. Violation of this assumption will cause suboptimal learning.

2.2.3 Discrete Reinforcement Learning

2.2.3.1 Temporal-Difference Theory

Suppose you have an inaccurate estimation of reward $V(s_t)$ for state s_t, and you now obtain a more accurate estimation $R(s_t)$. It is reasonable to update the inaccurate estimation as follows

$$V(s_t) \to V(s_t) + \alpha(R(s_t) - V(s_t))$$

where α is the learning rate.

In a step-by-step incremental training process, the learning agent can observe reward r_{t+1} and estimation $V(s_{t+1})$ after the operations at time t. According to equation 2.4, if the current estimate of reward $V(s_t)$ is inaccurate, the more accurate estimate is $R(s_t) = r_{t+1} + \gamma V(s_{t+1})$, because it contains the new experience of reinforcement r_{t+1}. Therefore, the value surface updating rule is

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) \ . \tag{2.6}$$

In equation 2.6, $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ is called the *Temporal-Difference Error*, based on which a set of online reinforcement learning algorithms were developed, such as SARSA and Q-learning. It was proved that these algorithms approximate the accurate estimation of reward after an infinite time of exploration (Sutton & Barto, 1998).

2.2.3.2 Q-learning

Q-learning was proposed by Watkins (1989). The basic idea behind this learning algorithm is very similar to those based on value surface updating. However, comparing it to the other algorithms, Q-learning is easier to understand and implement. The biggest difference in Q-learning is that it calculates $Q_\pi(s, a)$, the estimation of future reward of taking action a in state s according to policy π, instead of just $V_\pi(s)$. $Q_\pi(s, a)$ at time t is defined as

$$Q_\pi(s_t, a_t) = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k})\} \ ,$$

where $r(s_t, a_t)$ is the observed reward of taking action a_t in state s_t.

As $V_\pi(s)$ is the expectation of future reward of state s assuming the current action policy π is applied, we can have $V_\pi(s) = \max_a\{Q_\pi(s, a)\}$.

Similar to the value surface updating rule, the Q-learning updating rule is

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_a Q(s', a) - Q(s, a)) \ . \tag{2.7}$$

Given $Q(s, a)$, we can obtain the corresponding action policy as $\pi(s) = \text{argmax}_a Q(s, a)$.

The SARSA (State-Action-Reward-State-Action) algorithm (Sutton & Barto, 1998)is almost the same as Q-learning, except it employs an on-policy temporal-difference error and updates the Q function as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a)) \ , \tag{2.8}$$

where s' and a' are the actual state and action for the next time step.

2.2.3.3 Actor/Critic learning

It is convenient to separate the learning into two parts: the critic and the actor. The critic updates the estimate of future reward and produces critique information to the actor. The actor updates the action policy based on the critique information and produces an action. The architecture is shown in Figure 2.3. The critique signal could be the temporal-difference error, the derivatives of the value surface or an expected action depending on how the actor is implemented.

Kretchmar (2000) used a Q-learning algorithm with a CMAC function approximator for the critic. The critic outputs an expected action to the actor as critique information. The actor was implemented with a neural network whose output approximates the expected action. The benefits of this approach are twofold. First, the Q-learning algorithm produces an expected action using a time-consuming search technique. Using an actor to duplicate the policy structure implicit in the critic speeds up the agent's response after the learning is accomplished. Second, the neural net structure of the actor satisfies the assumptions of robust stability theory as shown by Kretchmar.

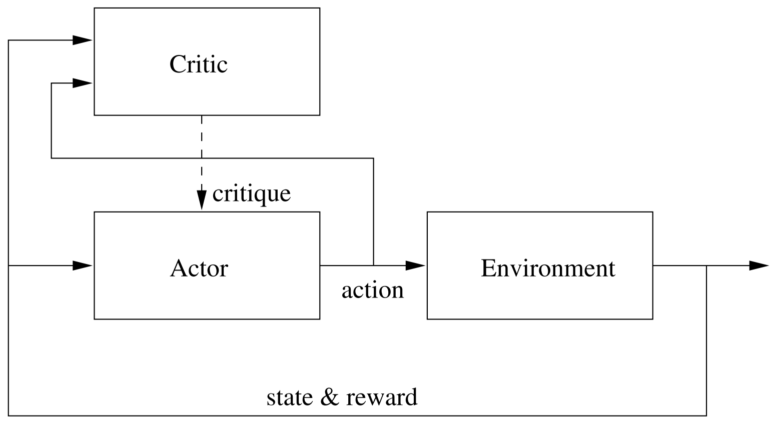

Figure 2.3: The Critic/Actor learning architecture

2.2.4 Continuous Reinforcement Learning

2.2.4.1 Continuous Temporal-Difference Theory

The mathematical formulation of a reinforcement learning problem in a continuous-time control system is as follows. Let a continuous control system be represented by

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \ , \tag{2.9}$$

where $\mathbf{x} \in \mathbf{X} \subset \mathbf{R}^n$ is the state and $\mathbf{u} \in \mathbf{U} \subset \mathbf{R}^m$ is the control input. The immediate reward for the current state and action is

$$r(t) = r(\mathbf{x}(t), \mathbf{u}(t)) \ . \tag{2.10}$$

The goal is to find a continuous control policy

$$u(t) = \pi(\mathbf{x}(t)) \ , \tag{2.11}$$

that maximizes the future rewards

$$V^{\pi}(\mathbf{x}(t)) = \int_t^{\infty} e^{-\frac{s-t}{\tau}} r(\mathbf{x}(s), \mathbf{u}(s)) ds \ , \tag{2.12}$$

where τ is the time constant for discounting future rewards.

The *continuous temporal-difference error* can be derived by differentiating definition 2.12 by t:

$$\dot{V}^{\pi}(\mathbf{x}(t)) = \frac{1}{\tau} V^{\pi}(\mathbf{x}(t)) - r(t) \tag{2.13}$$

As equation 2.13 should hold for an accurate estimation V, an inaccurate estimate V should be adjusted to reduce the inconsistency:

$$\begin{aligned} V(\mathbf{x}(t)) &\leftarrow V(\mathbf{x}(t)) + \delta(t) \ , && (2.14) \\ \delta(t) &= r(t) - \frac{1}{\tau} V^{\mu}(\mathbf{x}(t)) + \dot{V}^{\mu}(\mathbf{x}(t)) \ . && (2.15) \end{aligned}$$

This is the definition of *continuous temporal-difference error.*

If an optimal policy π^* is already learned, the value function V^* is

$$V^*(\mathbf{x}(t)) = \max_{\mathbf{u}[t,\infty)} \left[\int_t^{\infty} e^{-\frac{s-t}{\tau}} r(\mathbf{x}(s), \mathbf{u}(s)) ds \right] \ , \tag{2.16}$$

where $\mathbf{u}[t,\infty)$ denotes an optimal control sequence $\mathbf{u}(s)$, $t \le s < \infty$. Equation 2.16 can be rewritten in the form of a Hamilton-Jacobi-Bellman equation (Doya, 2000):

$$\frac{1}{\tau}V^*(\mathbf{x}(t)) = \max_{\mathbf{u}(t)}[r(\mathbf{x}(s),\mathbf{u}(s)) + \frac{\partial V^*(\mathbf{x})}{\partial \mathbf{x}} f(\mathbf{x}(s),\mathbf{u}(s))] \tag{2.17}$$

Therefore the optimal control policy π^* can be determined by

$$\mathbf{u}(t) = \pi^*(\mathbf{x}(t)) = \mathrm{argmax}_{\mathbf{u}(t)\in\mathbf{U}}[r(\mathbf{x}(s),\mathbf{u}(s)) + \frac{\partial V^*(\mathbf{x})}{\partial \mathbf{x}} f(\mathbf{x}(s),\mathbf{u}(s))] \tag{2.18}$$

While implementing the continuous temporal-difference error updating rule, we can utilize the discretized version of the time derivative of V(t), $\dot{V}(\mathbf{x}(t)) = \frac{V(\mathbf{x}(t))-V(\mathbf{x}(t-\Delta t))}{\Delta t}$, and obtain

$$\delta(t) = r(t) + \frac{1}{\Delta t}[(1-\frac{\Delta t}{\tau})V(\mathbf{x}(t)) - V(\mathbf{x}(t-\Delta t))] \,. \tag{2.19}$$

The $V(\mathbf{x})$ is represented by some function approximator $V(\mathbf{x};\mathbf{w})$ in implementation, where $\mathbf{w}$ is the parameter vector for the approximation. To update $V(\mathbf{x})$ by the amount of continuous TD error, the parameter $\mathbf{w}$ in the function approximator at time $t - \Delta t$ should be adjusted according to gradient descent algorithm

$$\dot{w}_i = \eta\,\delta(t)\,\frac{\partial V(\mathbf{x}(t-\Delta t);\mathbf{w})}{\partial w_i} \,. \tag{2.20}$$

Consider *exponential eligibility tracing*, which can improve the learning speed by allowing the learning agent to update the value function of the states in the past simultaneously according to current instantaneous reward. A more complex but more effective updating rule is as follows (Doya, 2000)

$$\begin{aligned} \dot{w}_i &= \eta\delta(t)e_i(t) \,, && (2.21) \\ \dot{e}_i(t) &= -\frac{1}{\kappa}e_i(t) + \frac{\partial V(\mathbf{x}(t-\Delta t);\mathbf{w})}{\partial w_i} \,, && (2.22) \end{aligned}$$

where $\kappa < \tau$ is a time constant for the discount of future reward estimate.

2.2.4.2 Continuous Actor/Critic Learning

The architecture of continuous Critic/Actor learning is exactly the same as that of discrete Critic/Actor learning. What distinguishes the continuous version from the discrete one are the continuous learning rules.

In a continuous critic, the value function is represented by a function approximator. For an instantaneous input of state $\mathbf{x}(t)$ and $\mathbf{r}(t)$ at time t, the continuous TD error is calculated and the function approximator is updated using equations 2.15 and 2.22, respectively. The TD error is also output to the actor as a critique signal.

There are many ways to implement the actor. Doya (2000) implemented the actor based on the Stochastic Real-Valued (SRV) unit algorithm of Gullapalli (1990). It employs a function approximator to approximate the continuous action policy $\pi : \mathbf{x} \to \mathbf{a}$, and the action is calculated as follows:

$$\mathbf{u}(t) = s(A(\mathbf{x}(t);\mathbf{w}^A) + \sigma\mathbf{n}(t)) \tag{2.23}$$

where $A(\mathbf{x}(t);\mathbf{w}^A) \in R^m$ is a function approximator, $\mathbf{n}(t) \in R^m$ is noise that allows the Actor to do exploration, σ defines the rate of exploration, and $s(\cdot)$ is a monotonically increasing output function. According to the criterion proposed by Gullapalli (1990) that the parameters of the policy approximator should be adjusted in favor of positive temporal difference error, the parameters $\mathbf{w}^A$ are adjusted as

$$\dot{w}_i^A = \eta^A\delta(t)\mathbf{n}(t)\frac{\partial A(\mathbf{x}(t);\mathbf{w}^A)}{\partial w_i^A} \,. \tag{2.24}$$

For a continuous learning problem, it is convenient to employ a low-pass filtered noise defined as follows

$$\tau_n \dot{\mathbf{n}}(t) = -\mathbf{n}(t) + \mathbf{N}(t) , \tag{2.25}$$

where $\mathbf{N}(t)$ is normal Gaussian noise.

The variance of noise σ should be reduced while performance is improving. To achieve this goal, we define $\sigma = \sigma_0 \min[1, \max[0, \frac{V_1 - V(t)}{V_1 - V_0}]]$ where V_0 and V_1 are the minimal and maximal values of the expected reward.

2.2.4.3 Continuous Value Gradient Learning

In a discrete learning algorithm, the best action for a state has to be found using a search technique, which is time-consuming. This problem would be even more severe when the action set is continuous.

However, given equation 2.18 as the formulation of an optimal policy, the best action could be calculated based on the fact that the derivative at the maximal point of a function is equal to zero. The solution is unique when the function with respect to the input is convex. Specific for this optimization problem, the best action can be calculated by solving the equation

$$\frac{\partial r(\mathbf{x}(t), \mathbf{u}(t))}{\partial \mathbf{u}(t)} + \frac{\partial V(\mathbf{x})}{\partial \mathbf{x}} \frac{\partial f(\mathbf{x}(t), \mathbf{u}(t))}{\partial \mathbf{u}(t)} = 0 \tag{2.26}$$

for the value of $u(t)$.

According to the above equation, if the system dynamics $f(\mathbf{x}, \mathbf{u})$ with respect to $\mathbf{u}$ are linear, we can make the right hand side of equation 2.18 convex by defining

$$r(\mathbf{x}, \mathbf{u}) = R(\mathbf{x}) - \sum_{j=1}^{m} S_j(u_j) \tag{2.27}$$

where $R(\mathbf{x})$ is the instantaneous reward for the current state $\mathbf{x}$ and $S_j(\cdot)$ is a convex cost function for action variable u_j.

Accordingly, we have

$$-\frac{\partial S_j(u_j)}{\partial u_j} + \frac{\partial V(\mathbf{x})}{\partial \mathbf{x}} \frac{\partial f(\mathbf{x}(t), \mathbf{u}(t))}{\partial u_j(t)} = 0 \tag{2.28}$$

for $j = 1, ..., m$.

The best action can be calculated as follows

$$\mathbf{u} = S'^{-1}\left(\frac{\partial V(\mathbf{x})}{\partial \mathbf{x}} \frac{\partial f(\mathbf{x}(t), \mathbf{u}(t))}{\partial \mathbf{u}(t)}\right) . \tag{2.29}$$

Doya (2000) also proposed a convex cost function that can ensure that the calculated action is within the range of the predefined action amplitude. Suppose the bound of action is $|u_j| \leq u_j^{max}$, for $j = 1, ..., m$. The action cost function is defined as

$$S_j(u_j) = c_j \int_0^{u_j} s^{-1}\left(\frac{u}{u_j^{max}}\right) du , \tag{2.30}$$

where $s(\cdot)$ is a sigmoid function ($s(x) = \frac{2}{\pi} \arctan(\frac{\pi}{2} x)$) and c_j is the weight on the cost function, as shown in Figure 2.4. The best action therefore is

$$u_j = u_j^{max} s\left(\frac{1}{c_j} \frac{\partial V(\mathbf{x})}{\partial \mathbf{x}} \frac{\partial f(\mathbf{x}(t), \mathbf{u}(t))}{\partial \mathbf{u}(t)}\right) . \tag{2.31}$$

When the weight $c_j \to 0$, we have a "bang-bang" optimal control policy

$$u_j = u_j^{max} sgn\left(\frac{\partial V(\mathbf{x})}{\partial \mathbf{x}} \frac{\partial f(\mathbf{x}(t), \mathbf{u}(t))}{\partial \mathbf{u}(t)}\right) . \tag{2.32}$$

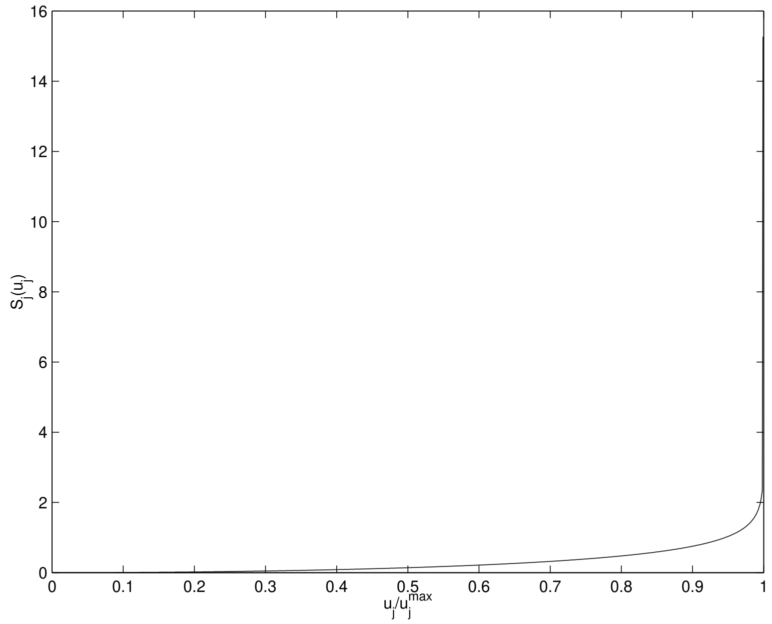

Figure 2.4: The cost function for normalized action

Chapter 3

A Continuous Reinforcement Learning Architecture for Feedback Control

In this chapter, the cause of the learning problem for a feedback system with a PI controller is analyzed. Based on this analysis, a new learning architecture with the statespace model of a feedback control system is proposed. The details on how we applied the continuous reinforcement learning algorithms are described.

3.1 Analysis of the Learning Problem with PI controller

To perform the analysis, we will convert the closed-loop feedback system to an open-loop system. Consider the output of the learning agent $a(s)$ as the input to a black box system, and the input to the learning agent $e(s)$ as the output of this black box system, as shown in Figure 3.1. We have the following equations in the Laplace domain,

$$\begin{aligned} e(s) &= r(s) - o(s) \\ &= r(s) - (e(s)C + a(s))P \ . \end{aligned}$$

If $r(s) = 0$, we have

$$\frac{e(s)}{a(s)} = \frac{-P}{1 + CP} \ ,$$

and if $a(s) = 0$, we have

$$\frac{e(s)}{r(s)} = \frac{1}{1 + CP} \ .$$

Therefore, as shown in Figure 3.2, the feedback control system can be converted into the 2-input-1-output open-loop system

$$\left[\ e(s) \ \right] = \left[\ \tfrac{-P}{1+CP} \quad \tfrac{1}{1+CP} \ \right] \left[\begin{array}{c} a(s) \\ r(s) \end{array} \right] \ .$$

One critical assumption of reinforcement learning is that the learning task should satisfy the Markov property, otherwise the learning result would be suboptimal (see Section 2.2.2). After we convert the feedback system into an open-loop system, we have the opportunity to analyze whether a learning problem with this system architecture possesses the Markov property.

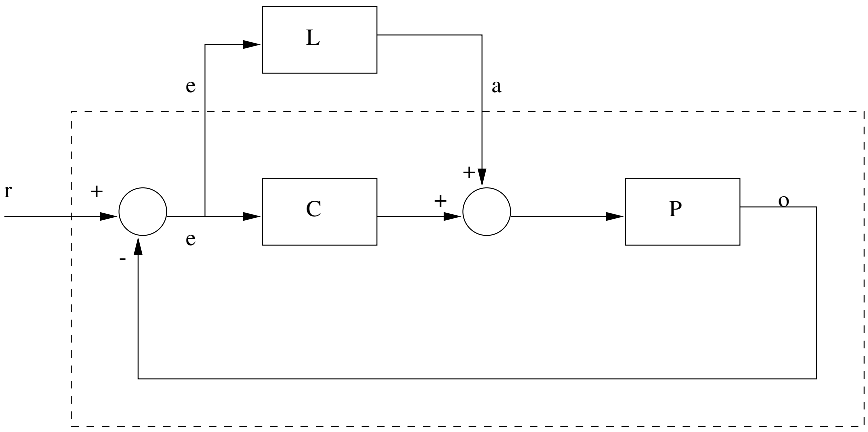

Figure 3.1: The feedback system with learning agent. L is the learning agent, C is a controller, and P is a plant.

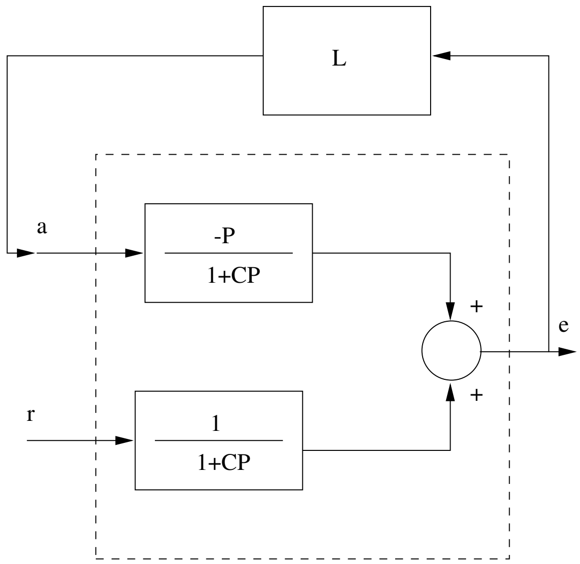

Figure 3.2: Open-loop view of the feedback system with learning agent

In the Laplace Domain, a system of transfer function F(s) with input X(s) has output Y(s)=X(s)F(s). The corresponding output in the time domain is $y(t) = \int_{-\infty}^{\infty} f(\tau)x(t-\tau)d\tau$ where $y(t) = L^{-1}(Y(s))$, $f(t) = L^{-1}(F(s))$ and $x(t) = L^{-1}(X(s))$. Similarly, for a discrete system, we have

$$y(k) = \sum_i f(i)x(k-i) \ ,$$

where

$$f(i) = \begin{cases} f(i), & i >= 0 \\ 0, & i < 0 \end{cases}$$

and

$$x(i) = \begin{cases} x(i), & i >= 0 \\ 0, & i < 0 \end{cases} \quad .$$

As shown in Figure 3.3, the output $y(k)$ is the sum of the product of $f(i)$ and $x(k-i)$ as $x(-i)$ moves forward while k increases. For time $t = k$, we have

Figure 3.3: Convolution in discrete time

$$y(k) \quad = \quad f(0)x(k) + f(1)x(k-1) + ... + f(k)x(0) \; .$$

As shown in the open-loop view of the system in Figure 3.2, the learning agent has the discretized measurement of tracking error $e(k)$ as $y(k)$, and generates the next action $a(k+1)$ as the input $x(k+1)$. If the system is Markov, the following equation should hold for the next system output $y(k+1) = e(k+1)$:

$$\begin{aligned} y(k+1) &= f(0)x(k+1) + f(1)x(k) + ... + f(k+1)x(0) \\ &= f(0)x(k+1) + c(f(0)x(k) + ... + f(k)x(0)) \\ &= f(0)x(k+1) + cy(k) \; , \end{aligned}$$

where c is an unknown constant.
So whether the learning process is a Markov Decision Process or not depends on whether $f(k) = cf(k-1)$ holds for the system function $f(k)$.

This explanation can be verified with a small example. Suppose $P = \frac{1}{1+s}$, and $C = K_p + K_i \frac{1}{s}$. We have

$$\begin{aligned} \frac{-P}{1+PC} &= \frac{-\frac{1}{s+1}}{1 + (K_p + K_i \frac{1}{s})\frac{1}{s+1}} \\ &= \frac{-s}{s^2 + (K_p+1)s + K_i} \; . \end{aligned}$$

Let $K_p = 1$, and $K_i = 0, 0.1, 0.5, 1, 5$, or 10, we can plot the impulse response of $f(k) = \mathcal{L}^{-1}\frac{-P}{1+PC}$. As shown in Figure 3.4, when K_i is zero, the impulse response is $f(k) = -e^{-2k}$, which exactly satisfies the equation $f(k) = cf(k-1)$ if $c = e^{-2}$. Therefore the system is Markov, and there is no learning convergence problem. However, if K_i is not zero, the equation $f(k) = cf(k-1)$ no longer holds for the system response, therefore the learning process becomes non-Markov and the learning has a convergence problem.

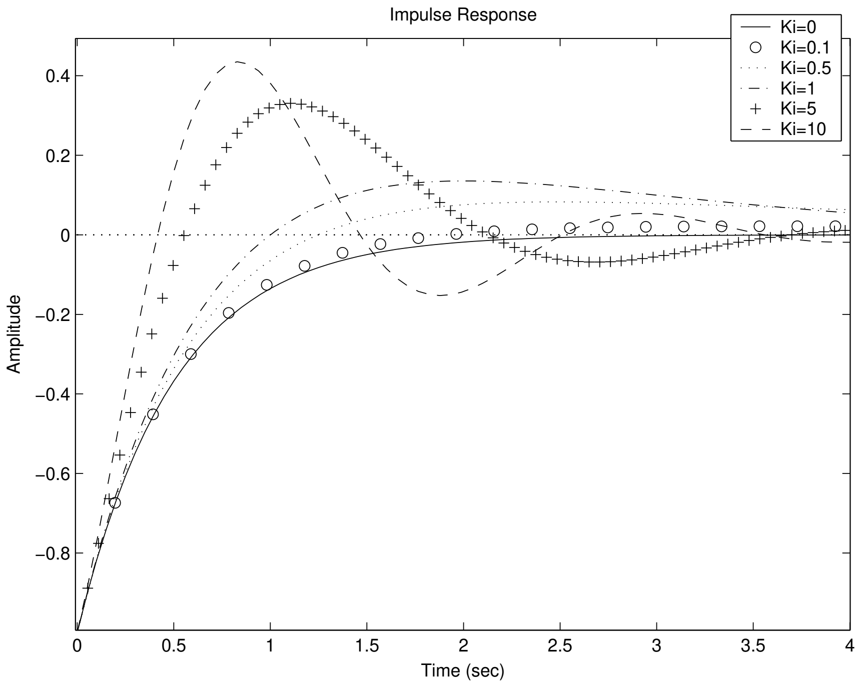

Figure 3.4: Impulse response of system transfer function $\frac{-P}{1+PC}$ with different K_i

3.2 Converting the System into a Statespace Model

3.2.1 Direct Conversion

A reasonable solution is to convert the feedback system into a statespace model. According to the definition of a statespace model, if the learning agent has the internal states of the model as input, the learning task is converted into a Markov Decision Process (MDP) problem. Though this idea is simple, it was often ignored because of the intricacy of converting the whole feedback system into a statespace model.

According to Figure 3.2, we can easily convert the system into a 2-input-1-output statespace model:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{3.1}$$
$$y = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}\ , \tag{3.2}$$

where

$$\mathbf{u} = \begin{bmatrix} a \\ r \end{bmatrix}, \text{ and } \ y = e.$$

For a plant $\frac{1}{s+1}$ with a PI controller, we have

$$\frac{-P}{1+CP} = \frac{-s}{s^2 + (K_p + 1)s + K_i}$$

and

$$\frac{1}{1+CP} = \frac{s^2+s}{s^2+(K_p+1)s+K_i} \; .$$

From the above equations, we can tell the system is a second-order system, therefore the state is

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} .$$

For $K_p = 1$ and $K_i = 10$, the matrices **A**, **B**, **C**, **D** obtained by using the $[A, B, C, D] = ss(sys)$ command in Matlab control toolbox are

$$\mathbf{A} = \begin{bmatrix} 0 & -2.5 \\ 4 & -2 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0 & -1.25 \\ -0.5 & -0.5 \end{bmatrix}, \mathbf{C} = \begin{bmatrix} 0 & 2 \end{bmatrix}, \mathbf{D} = \begin{bmatrix} 0 & 1 \end{bmatrix} .$$

The new architecture of the learning agent is shown in Figure 3.5. The learning agent takes 5 inputs: action, reference signal, the two states of the statespace model, and the tracking error signal as performance reward.

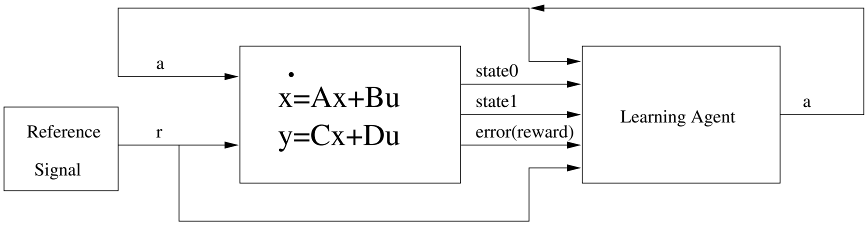

Figure 3.5: The learning with state space model of the plants

3.2.2 A Statespace Model With Physical Meaning

As eventually the learning agent has to cooperate with the physical system, it is important to study how to convert a feedback system into a statespace model with physical meaning. This section will show the possibility with a small example.

As we know, the state space model of a PI controller, $K_p + K_i/s$, is

$$\dot{\bar{x}}_0 = 0\bar{x}_0 + K_i u_c \; , \tag{3.3}$$

$$y_c = \bar{x}_0 + K_p u_c \; . \tag{3.4}$$

Suppose the state space model of the plant is

$$\dot{\bar{x}}_1 = A\bar{x}_1 + Bu_p \; , \tag{3.5}$$

$$y_p = C\bar{x}_1 \; . \tag{3.6}$$

Here we have $D = 0$ for the physical plant because it is *strictly proper* ($y_p \to 0$ when the frequency of input u_p goes to infinity).

The states of the two first-order statespace models have their own physical meanings. According to the equations, $\bar{x}_0$ can be obtained by measuring the controller output and input as $\bar{x}_0 = y_c - K_p u_c$, and $\bar{x}_1$ can be obtained by measuring the plant output as $\bar{x}_1 = y_p/C$.

According to Figure 3.6, we also have:

$$u_c = r - y_p \tag{3.7}$$

$$u_p = a + y_c \tag{3.8}$$

Based on these statespace models and the above equations, we can derive for the learning agent a statespace model whose states have physical meanings:

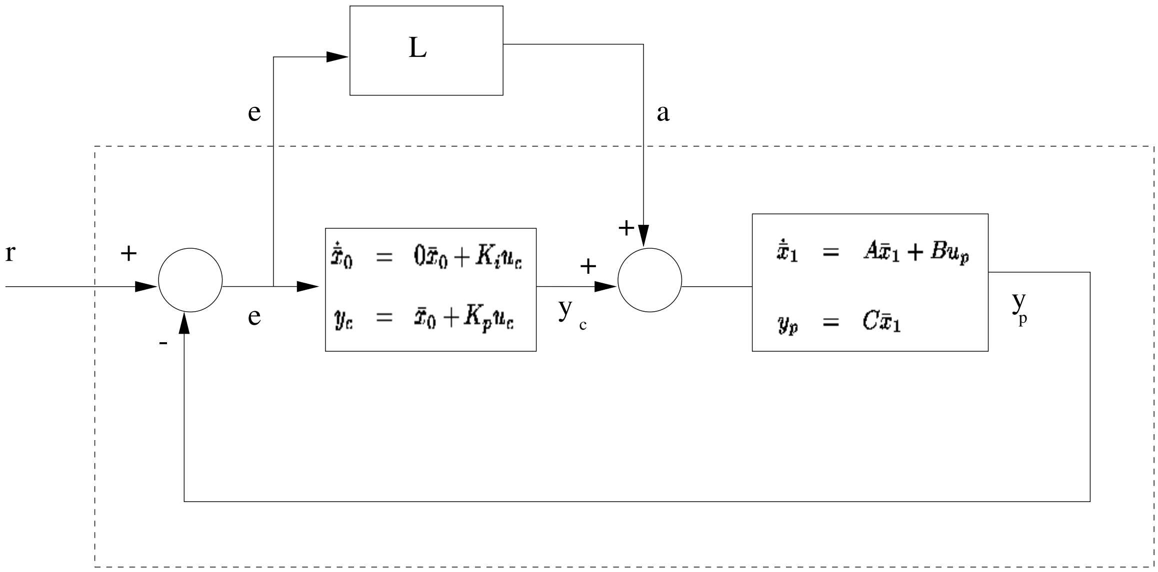

Figure 3.6: The state space model view of the feedback system. L is the learning Agent.

$$\begin{bmatrix} \dot{\bar{x}}_0 \\ \dot{\bar{x}}_1 \end{bmatrix} = \begin{bmatrix} 0 & -K_iC \\ B & A-K_pBC \end{bmatrix} \begin{bmatrix} \bar{x}_0 \\ \bar{x}_1 \end{bmatrix} + \begin{bmatrix} 0 & K_i \\ B & BK_p \end{bmatrix} \begin{bmatrix} a \\ r \end{bmatrix}, \tag{3.9}$$

$$e = \begin{bmatrix} 0 & -C \end{bmatrix} \begin{bmatrix} \bar{x}_0 \\ \bar{x}_1 \end{bmatrix} + \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ r \end{bmatrix}. \tag{3.10}$$

Letting $K_p = 1$, $K_i = 10$ for the controller, and $A = -1$, $B = 1$, $C = 1$ for the plant $\frac{1}{s+1}$, we have

$$\begin{bmatrix} \dot{\bar{x}}_0 \\ \dot{\bar{x}}_1 \end{bmatrix} = \begin{bmatrix} 0 & -10 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} \bar{x}_0 \\ \bar{x}_1 \end{bmatrix} + \begin{bmatrix} 0 & 10 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} a \\ r \end{bmatrix}, \tag{3.11}$$

$$e = \begin{bmatrix} 0 & -1 \end{bmatrix} \begin{bmatrix} \bar{x}_0 \\ \bar{x}_1 \end{bmatrix} + \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ r \end{bmatrix}. \tag{3.12}$$

A comparison of this statespace model and the statespace model produced by Matlab will give us the mapping between the two state spaces:

$$\bar{x}_0 = -8x_0 , \tag{3.13}$$

$$\bar{x}_1 = -2x_1 . \tag{3.14}$$

3.3 Continuous Reinforcement Learning for Feedback Control System

To avoid the learning problem described in the previous section, we employ the new learning architecture with a statespace model of the system. To exemplify how to apply the continuous learning algorithms to feedback control systems, we will utilize the statespace model in equations 3.9 and 3.10 during the learning. The reference signal is also considered as an input to the learning. Therefore the learning space is three-dimensional (two system states, and one reference signal). As the purpose of the control is to reduce the tracking error, the reward is defined as

$$R(\mathbf{x}) = R(\begin{bmatrix} x_0 \\ x_1 \\ r \end{bmatrix}) = -|r - x_1|$$

where x_0 is the state of the PI controller (which is ignored if it is a P controller), x_1 is the plant output, and r is the reference signal.

The learning algorithms are described in Chapter 2. The rest of this chapter will describe how they are implemented within the above learning architecture.

3.3.1 Continuous Critic/Actor learning

Under the guidance of the statespace model of the feedback control system, we can construct a more detailed learning architecture as shown in Figure 3.7. Comparing it to the learning architecture

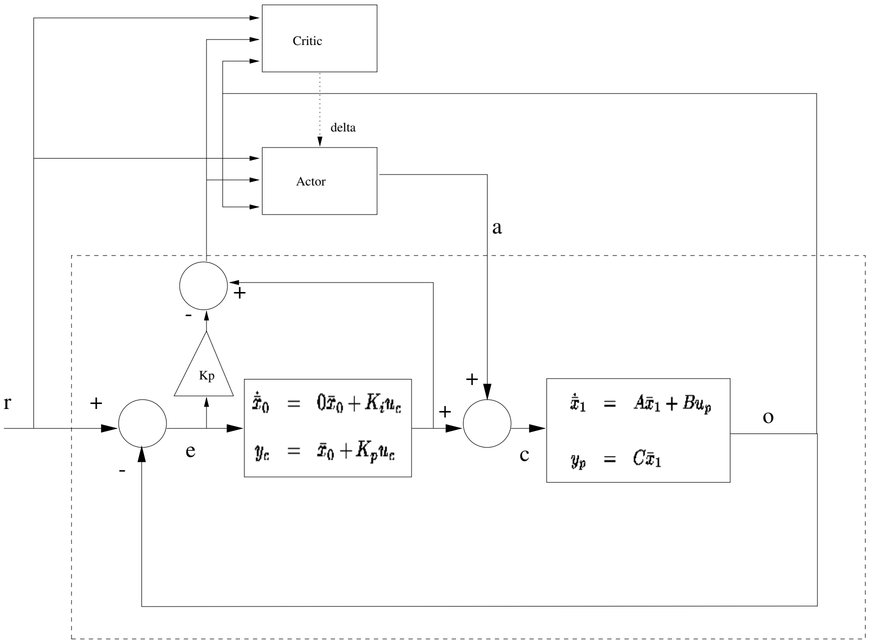

Figure 3.7: New Critic/Actor learning architecture with the statespace model

shown in Figure 1.3, the biggest difference of the new architecture is that it extracts the physical state information from the feedback control system and turns the learning problem into a Markov one.

As in (Doya, 2000), both the Critic and Actor utilize a Normalized Gaussian Network $G(\mathbf{x};\mathbf{w})$ function approximator defined by

$$G(\mathbf{x};\mathbf{w}) = \sum_{k=1}^{K} w_k b_k(\mathbf{x}) \ , \tag{3.15}$$

where

$$b_k(\mathbf{x}) = \frac{a_k(\mathbf{x})}{\sum_{l=1}^{K} a_l(\mathbf{x})}, \quad a_k(\mathbf{x}) = e^{-\|\mathbf{s}_k^T(\mathbf{x}-\mathbf{c}_k)\|^2} \ . \tag{3.16}$$

The vectors $\mathbf{c}_k$ define the centers of the basis functions, and the vectors $\mathbf{s}_k$ define the size of the basis functions. To simplify the learning algorithm, we fixed the centers in a grid, and set the size to a constant value.

It is convenient to note the following facts:

$$\frac{\partial G(\mathbf{x};\mathbf{w})}{\partial w_k} = b_k(\mathbf{x}) \tag{3.17}$$

$$\frac{\partial G(\mathbf{x};\mathbf{w})}{\partial x_i} = \sum_k w_k \frac{\frac{\partial a_k(\mathbf{x})}{\partial x_i} Sa - a_k(\mathbf{x})\frac{\partial Sa}{\partial x_i}}{Sa^2} \tag{3.18}$$

where

$Sa = \sum_{l=1}^{K} a_l(\mathbf{x})$,

$\frac{\partial Sa}{\partial x_i} = \sum_{l=1}^{K} \frac{\partial a_l(\mathbf{x})}{\partial x_i}$, and

$\frac{\partial a_l(\mathbf{x})}{\partial x_i} = 2s_{lj}(x_j - c_{lj})$.

Equation 3.17 is important for updating the weights of the network when adjusting the value function, and equation 3.18 is important for the value gradient learning, which will be further discussed in the next section.

3.3.2 Value Gradient Learning

With this algorithm, the best action can be calculated directly based on the value function. However when we were trying to implement it exactly as described in Chapter 2, we noticed a strange phenomenon: the learning agent cannot learn to output an optimal action that is close to zero.

The cause of this problem is the implementation of the action cost function. As shown in Figure 2.4, the derivative of the cost function is very close to zero when the action is close to zero. In order to implement this integral function, we have to discretize it and store the integral values in a table. The discretization would set the derivatives to zero especially when the action value is very small. This derivative information is crucial for calculating the optimal action. Therefore, the learned optimal action close to zero actually is not optimal given the current implementation.

There is no better implementation for an integral function. Therefore, we proposed to use a different cost function,

$$S_j(u_j) = c_j(u_j/u_j^{max})^2 \ , \tag{3.19}$$

for which the best action is calculated by

$$u_j = u_j^{max}\frac{1}{c_j}\frac{\partial V(\mathbf{x})}{\partial \mathbf{x}}\frac{\partial f(\mathbf{x}(t),\mathbf{u}(t))}{\partial \mathbf{u}(t)} \ . \tag{3.20}$$

$\frac{\partial V(\mathbf{x})}{\partial \mathbf{x}}$ can be calculated according to equation 3.18, and from equation 3.9, $\frac{\partial f(\mathbf{x}(t),\mathbf{u}(t))}{\partial \mathbf{u}(t)}$ can be obtained by

$$\frac{\partial f(\mathbf{x}(t),\mathbf{u}(t))}{\partial \mathbf{u}(t)} = \begin{bmatrix} 0 \\ B \\ 0 \end{bmatrix} \ . \tag{3.21}$$

Though it can ensure that the learned optimal action close to zero is correct, this cost function cannot ensure the calculated optimal action is within the range of u_{max}. However, the user can adjust the weight c_j to a value large enough to make this happen.

Chapter 4

Experiment Results and Analysis

In this chapter, we present three sets of experimental results. The first experiment shows the performance improvement due to the learning architecture with the statespace model of the feedback system over the one utilized by Kretchmar (2000). The second experiment shows how the continuous reinforcement learning algorithms work with a simple feedback control system with a P controller. The third experiment verifies our conjecture: a learning architecture employing continuous reinforcement learning algorithms based on a statespace model of the feedback system can achieve the desired control performance, but the previous learning architecture employing a discrete learning algorithm cannot achieve the desired performance.

4.1 Performance Comparison Between the Learning Architectures With and Without the Statespace Model

Theoretically, the learning architecture with the statespace model of the system is superior to that without the statespace model. This experiment will verify this hypothesis. In both architectures, we employ the same discrete Q-learning algorithm with CMAC approximation described in (Kretchmar, 2000). The only difference between the two architectures is that one does not employ the states of the statespace model of the feedback system (Figure 1.3) while the other one does (Figure 3.5).

The feedback control system is composed of a PI controller with $K_p = 1$ and $K_i = 10$ and a plant $\frac{1}{s+1}$. The learning rate for the learning algorithm is 0.01. The reference signal changes within the range of [-5, 5] with uniform probability every 2.5 seconds. The simulation lasts 4,000,000 seconds with a sampling rate of 0.01 second. Figure 4.1 shows a typical system response with no RL controller. The average tracking error is 1.66.

Figure 4.2 shows the changes of the average tracking error of every 4000 seconds during the training. The average tracking error when learning without the statespace model drops much faster at the beginning, but stabilizes at a higher value on average after 2 million seconds, compared to learning with the statespace model. This is reasonable because the input to the learning agent when training without the statespace model is two dimensional (the reference signal and the error input), while the input when training with the system statespace model is four dimensional (reference signal, error input, the state of PI controller, and the state of the plant), which takes a longer time to explore for the same resolution of grid, but benefits the learning in the long run.

Figures 4.3 and 4.4 show the plant tracking error versus the reference signal and the corresponding action signal of the RL-controllers with the two learning architectures of 30 seconds after being trained for 4 million seconds. Both learning architectures can reduce the tracking error. However, the performance of the architecture with the statespace model of the system is better than that of the one without the statespace model. For the same reference signal input sequence, learning with the statespace model can reduce the average tracking error to 1.37 while learning without the statespace model can only reach 1.49. The improvement of control performance is illustrated by Figure 4.5. Visual inspection of the two figures can also lead us to the same conclusion: more

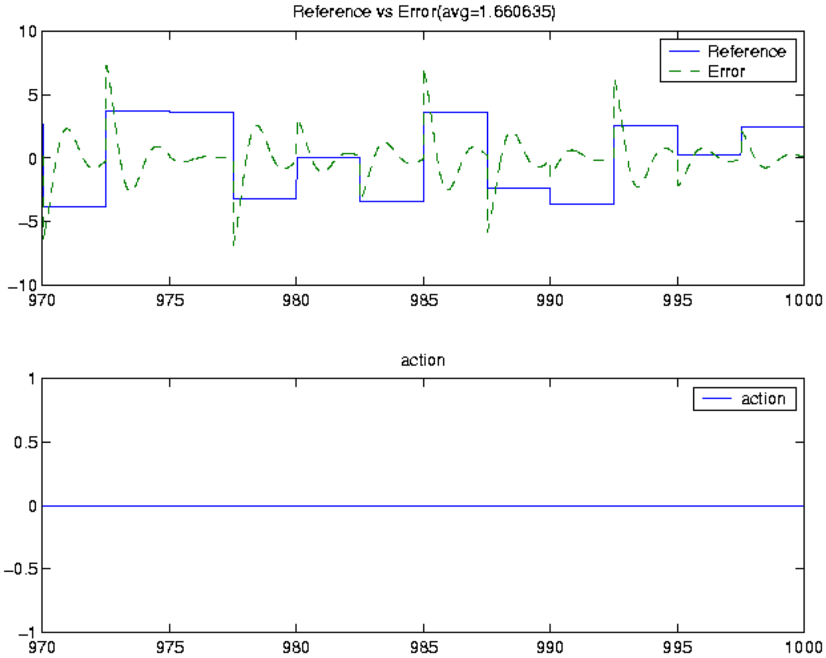

Figure 4.1: System response with no learning

overshooting is eliminated when learning with the statespace model. However, it is also worth noting that the learned optimal action sequence contains high frequency oscillations which are generated by the discretized state space searching. Such oscillation in the control signal is not desirable in a physical system, and therefore it should be eliminated. To solve the problem theoretically, we need to develop learning algorithms in continuous time and state space.

4.2 The Continuous Learning Algorithms Applied To Feedback System With P Controller

The purpose of this experiment is to analyze the performance of two continuous learning algorithms applied to feedback control system. For the purpose of visualizing the learning, we employed a feedback control system composed of a plant $\frac{1}{s+1}$ and P controller. As this control system is actually a first-order system, it contains only one internal state. Together with the reference signal, the input to the learning agent is (x_1, r) where x_1 is the plant output, and r is reference signal. We can visualize the learning by plotting the value surface and the action policy surface so that the mechanism of the learning algorithm applied to feedback control system can be easily understood.

For the P controller, we let $K_p = 20$. The reference signal changes between -5 and 5 every 0.5 seconds. Therefore it is reasonable to set the range of the states as $r \in [-5, 5]$ and $x_1 \in [-7.5, 7.5]$. We also simulated a smooth transition of the reference signal because such smooth signal transitions are typical in physical systems. In our simulation, the smooth transition takes about 0.1 second. The sampling rate of simulation for the feedback control system is 0.001 second. Figure 4.6 shows a typical system response with no RL controller. The average tracking error is 1.79, and there is a visible steady state error resulting from the lack of an integral term in the controller.

4.2.1 Continuous Actor/Critic Learning

We employed a Normalized Gaussian Network for the Actor and Critic with 25×25 bases, base size $\sigma = 10$. The time constants for learning are $\tau = 1$ and $\kappa = 0.1$. The sampling rate for learning

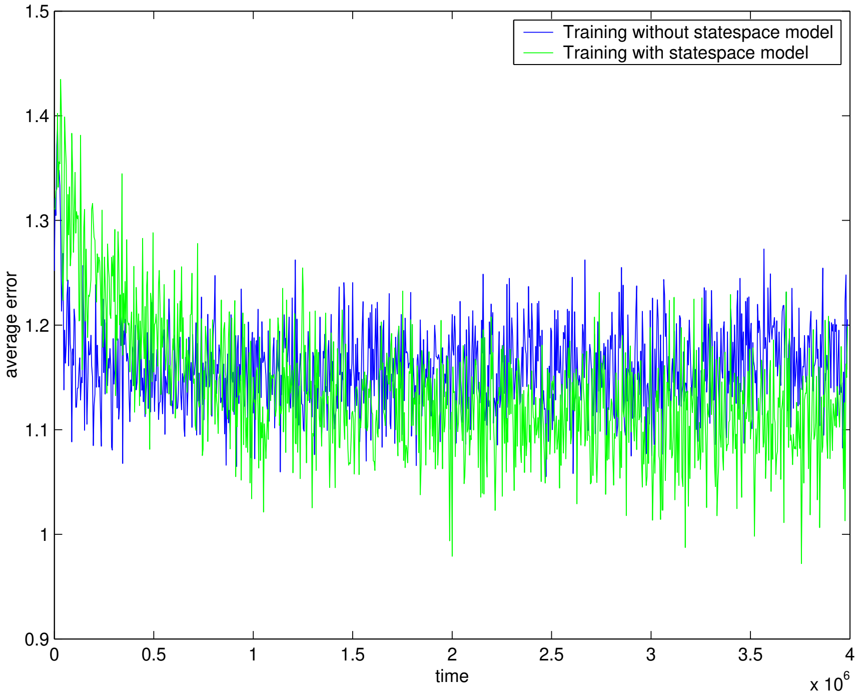

Figure 4.2: The comparison of performance of training with and without the statespace model

is $\Delta t = 0.01$. A low-pass filtered noise is injected to the Actor for the purpose of exploration with $\sigma_0 = 50$, $\tau_n = 1$, $V_0 = -10$ and $V_1 = 0$. The learning rate for the Critic is $\eta = 1$, and the learning rate for the Actor is $\eta^A = 5$. The range of action is $u_{max} = 100$ and the weight of penalty on action is $c = 0.1$.

Figure 4.7 shows the system response with the learning agent for 5 seconds after it has been trained for 1,000 seconds. The average tracking error is reduced to 1.51 and the steady state error is eliminated.

As the training space is two dimensional, we can plot the value surface and the policy surface with the x-axis as the plant output and the y-axis as the reference signal. Figure 4.8 shows the value surface, and Figure 4.9 shows the policy surface. The value surface is formed by the Critic, with which the estimate of future rewards from the current state is obtained and the TD error between state transitions is calculated and sent to the Actor as the critique signal. The policy surface is maintained by the Actor, with which the current state is mapped to a best action. After training, a trajectory of the state transitions on the policy surface is shown in Figure 4.10. Comparing it to Figure 4.7 will help to understand how the best policy is stored and retrieved in the Actor. The positions $(-5, -5)$ and $(5, 5)$ are the two steady states where the tracking error is zero. When the reference signal moves from -5 to 5, the plant output also moves to 5 along the upper-left trajectory and the peak of the action surface along this trajectory forces the plant output to move to 5 quickly to reduce the tracking error, and vice versa for the transition of the reference signal from 5 to -5.

4.2.2 Value Gradient Learning

For the Value Gradient learning algorithm, we employed a Normalized Gaussian Network with 20×20 bases, base size $\sigma = 10$. The time constants are $\tau = 1$ and $\kappa = 0.1$. The sampling rate for learning is 0.01. The parameters of low-pass filtered noise for exploration are $\sigma_0 = 250$, $\tau_n = 2$, $V_0 = -10$ and $V_1 = 0$. The learning rate for the Critic is $\eta = 1$. The range of action is $u_{max} = 100$ and the weight of penalty on action is $c = 0.05$.

Figure 4.11 shows the system response with the learning agent for 5 seconds after being trained for

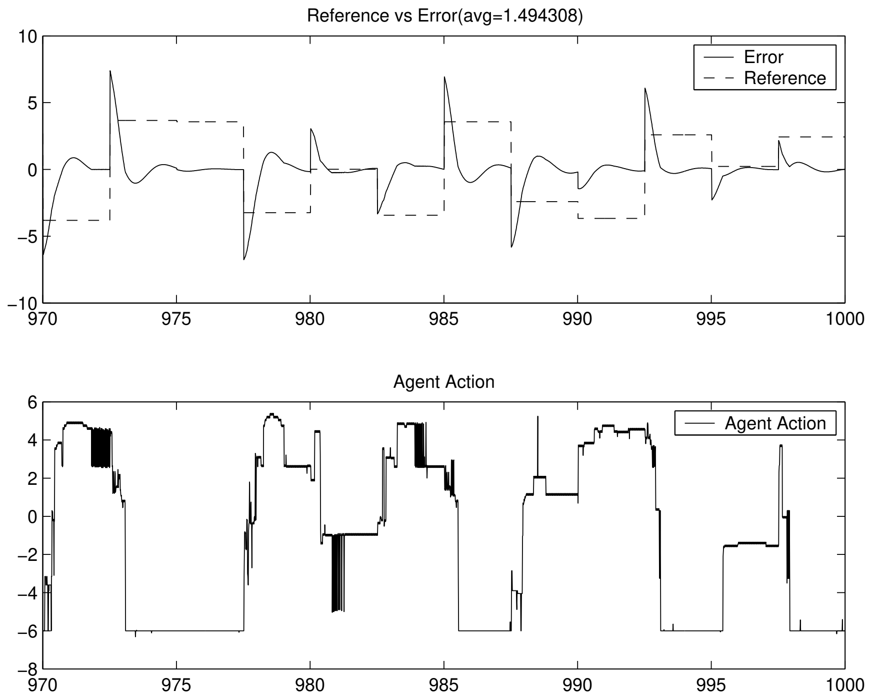

Figure 4.3: The control performance after training of 4M seconds for architecture without statespace model

5,000 seconds. The average tracking error is reduced to 1.55 and the steady state error is eliminated.

Figure 4.12 shows the value surface and Figure 4.13 shows the policy surface. The trajectory of the state transition on the policy surface is shown in Figure 4.14. It is interesting to notice that the value surface and policy surface are different from that of the Actor/Critic learning shown in Figures 4.8, 4.9 and 4.10. They are more complex. The Value Gradient algorithm learned to output a spike of action at an earlier stage on the trajectory of state transition and learned to output more oscillations in the action signal. These oscillations, however, are acceptable as they contain no high frequencies. The differences might be caused by the different learning strategies and the different setting of training parameters. However, we consider these two different control policies as equivalent here as they reached almost the same control performance and they agreed on the best actions to eliminate the steady state error. The improvement of control performance with continuous learning agents is shown in Figure 4.15.

4.3 The Continuous Learning Algorithms Applied To Feedback System With PI Controller

The purpose of this experiment is to verify the advantages of the continuous learning algorithm applied to feedback control system with PI controller. The plant is still $\frac{1}{s+1}$ and the parameters for PI controller are $K_p = 1$ and $K_i = 10$. The input to the learning agent is (x_0, x_1, r) where x_0 is the internal state of PI controller, x_1 is the plant output, and r is reference signal. Since we now have three inputs, surface plots cannot be shown for the following experiments.

The range of state x_0 is $[-25, 25]$, the range of state x_1 is $[-9, 9]$, and the reference signal is set to a random value between -5 and 5 every 2.5 seconds. The smooth transition of the reference signal takes about 0.5 second. The sampling rate of simulation for the feedback control system is 0.01 second.

Figure 4.16 shows a typical system response with no RL controller. The average tracking error is 1.06, and the tracking error is mostly caused by overshooting. Figure 4.17 shows the system response

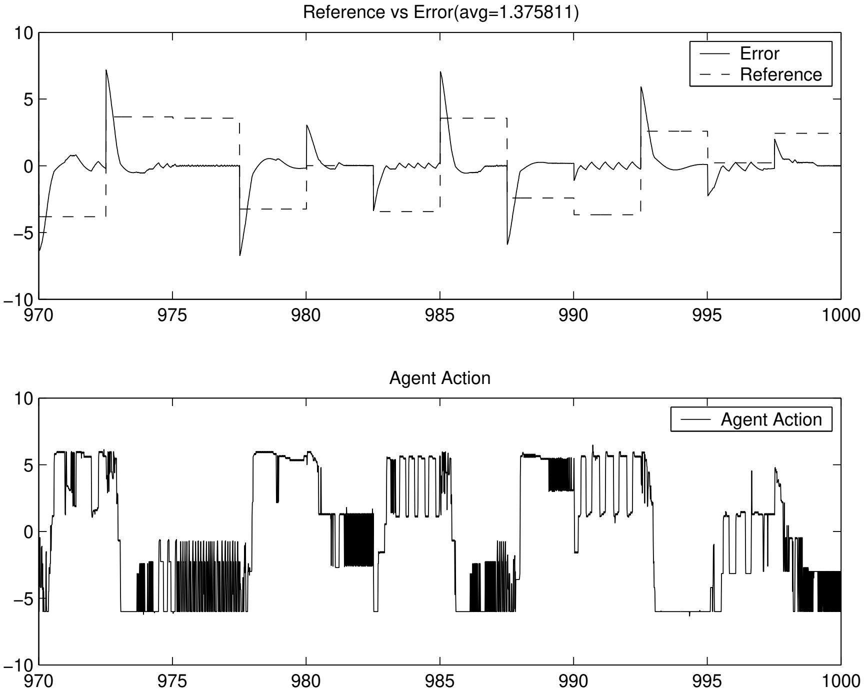

Figure 4.4: The control performance after training of 4M seconds for architecture with statespace model

with discrete RL controller trained for 10000 seconds (with learning rate $\alpha = 1$). While the average error is reduced to 0.67, the oscillation in the control sequence is severe. The following two sections will illustrate how the two continuous learning algorithms outperform the discrete learning on this point.

4.3.1 The Continuous Actor/Critic Learning

As the space for training is now three-dimensional, the grid of the bases of Normalized Gaussian Network for the Actor and Critic is $20 \times 20 \times 20$, and the base size is $\sigma = 10$. The time constants for training are $\tau = 1$ and $\kappa = 0.1$. The sampling rate for learning is $\Delta t = 0.1$. The parameters for the low-pass filtered noise are $\sigma_0 = 30$, $\tau_n = 1$, $V_0 = -10$ and $V_1 = 0$. The learning rate for the Critic is $\eta = 1$, and the learning rate for Actor is $\eta^A = 5$. The range of action is $u_{max} = 20$ and the weight of penalty on action is $c = 0.01$.

Figure 4.18 shows the system response with the learning agent for 30 seconds after being trained for 10,000 seconds. The average tracking error is reduced to 0.68 and the overshooting is largely reduced. Comparing it to the result of discrete learning algorithm, the control sequence is more continuous and has no high frequency oscillations.

4.3.2 The Value Gradient Learning

We employed a Normalized Gaussian Network for the approximation of the value surface with $20 \times 20 \times 20$ bases, base size $\sigma = 10$. The time constants are $\tau = 1$ and $\kappa = 0.1$. The sampling rate for learning is 0.01. The parameters of low-pass filtered noise for exploration are $\sigma_0 = 50$, $\tau_n = 2$, $V_0 = -10$ and $V_1 = 0$. The learning rate for the Critic is $\eta = 1$. The range of action is $u_{max} = 20$ and the weight of penalty on action is $c = 0.05$.

Figure 4.19 shows the system response with the learning agent for 30 seconds after being trained for 10000 seconds. The average tracking error is reduced to 0.66. Comparing it to that of the continuous Actor/Critic algorithm, we can tell that the learned optimal control policies of these

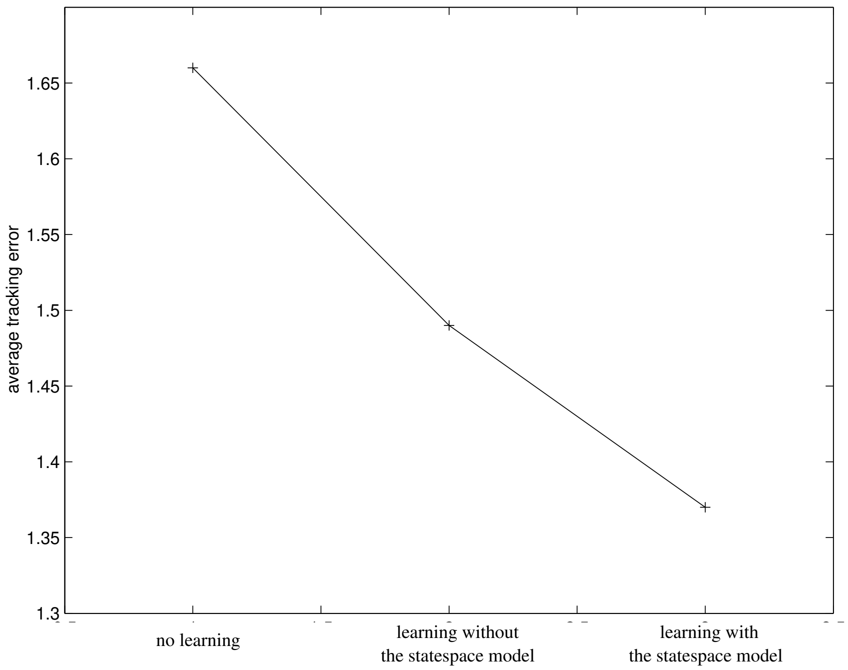

Figure 4.5: The average tracking error of no learning, of learning without the statespace model, and of learning with the state space model

two learning agents are slightly different as the control sequences and system responses are slightly different. However, again they reached similar control performances and thus should be considered as equally optimal. Similar to the control sequence of continuous Actor/Critic, there is no hazardous high frequency oscillation, which is the advantage of the continuous learning algorithms. The improvement of control performance with continuous learning agents is shown in Figure 4.20. The performance of discrete learning is marked with a black dot because its control sequence contains high frequency oscillations.

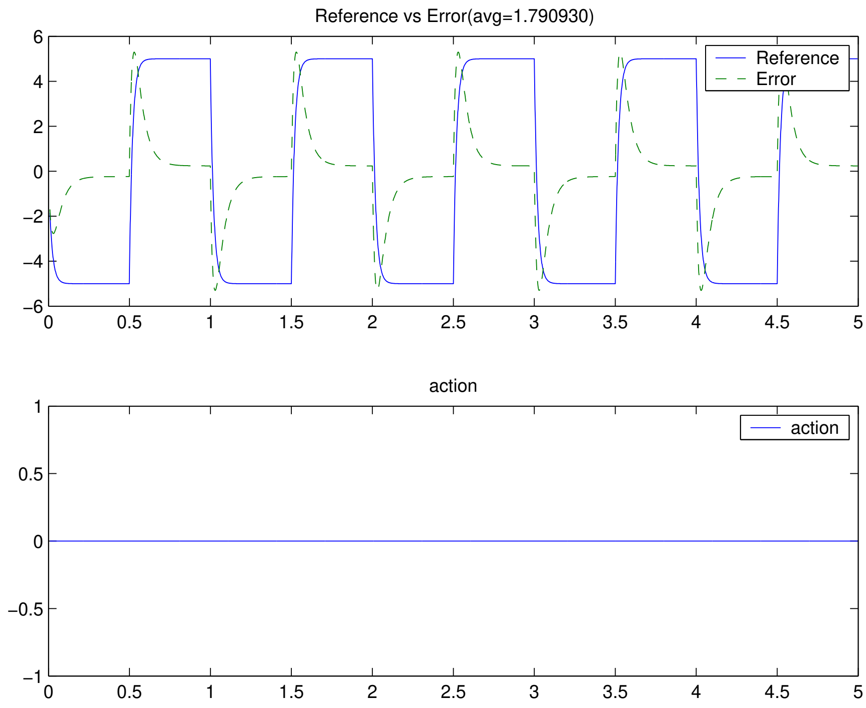

Figure 4.6: The system response without RL controller in experiment 2

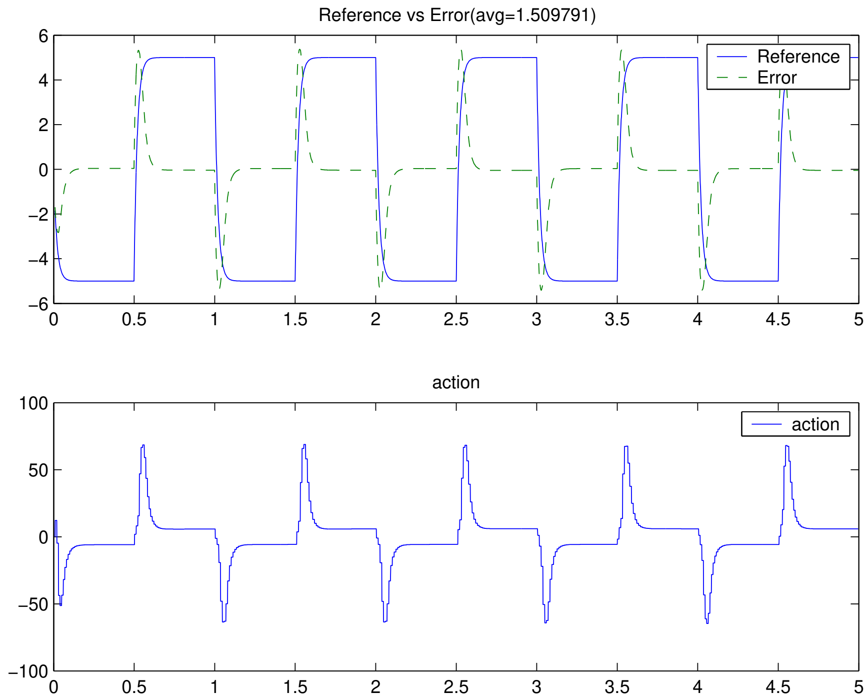

Figure 4.7: The system response with Actor/Critic trained for 1000 seconds in Experiment 2

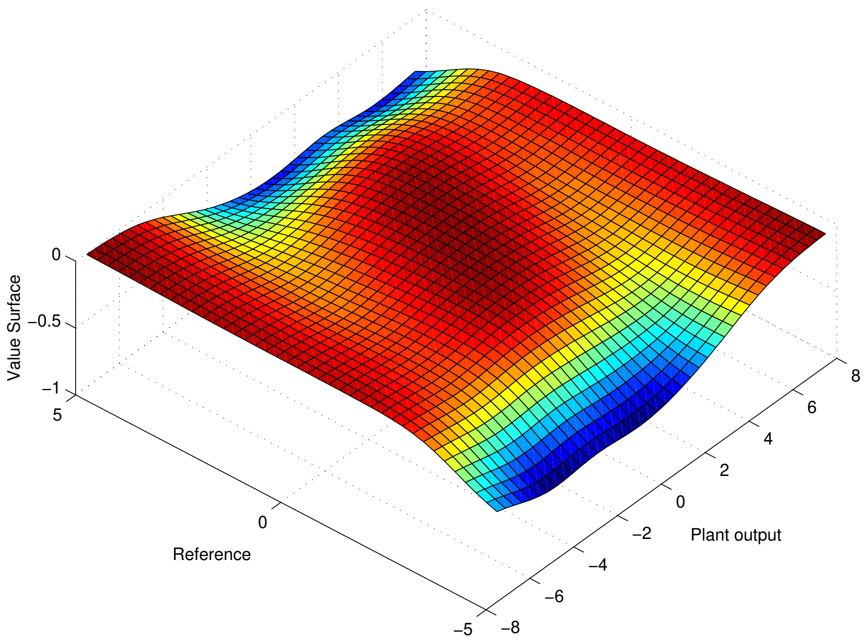

Figure 4.8: The Value Surface for Actor/Critic in Experiment 2

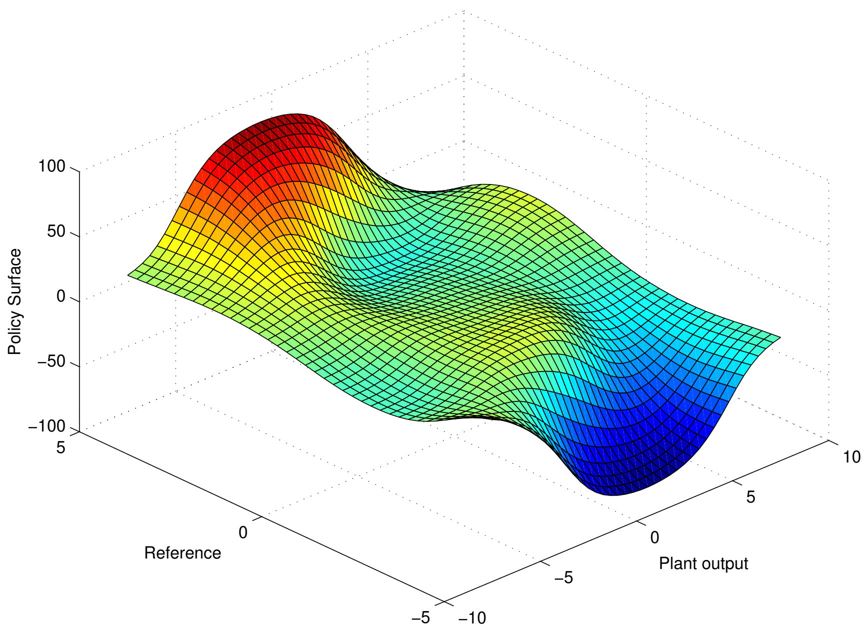

Figure 4.9: The Action Policy Surface for Actor/Critic in Experiment 2

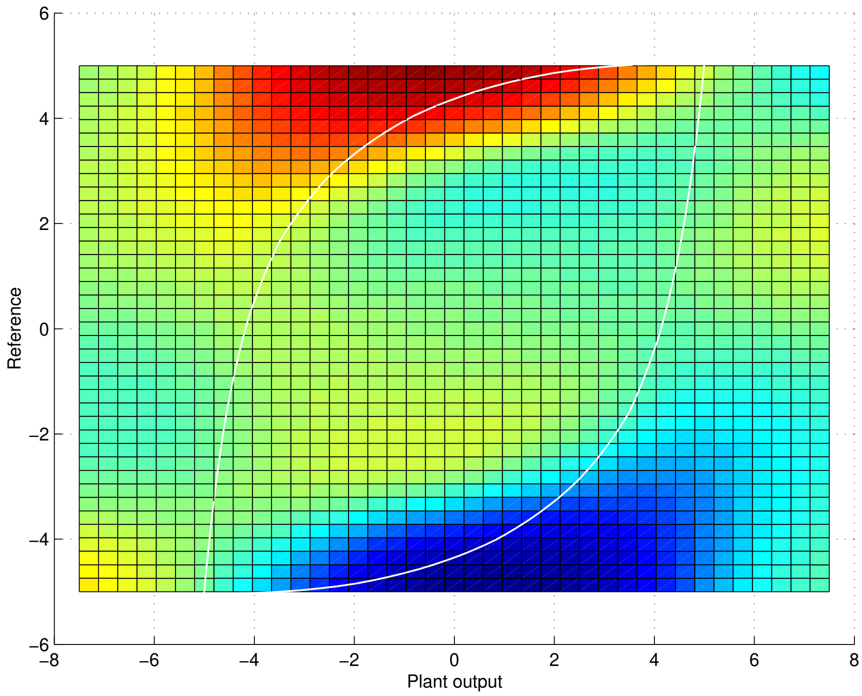

Figure 4.10: The trajectory of state transition on Policy Surface for Actor/Critic in Experiment 2

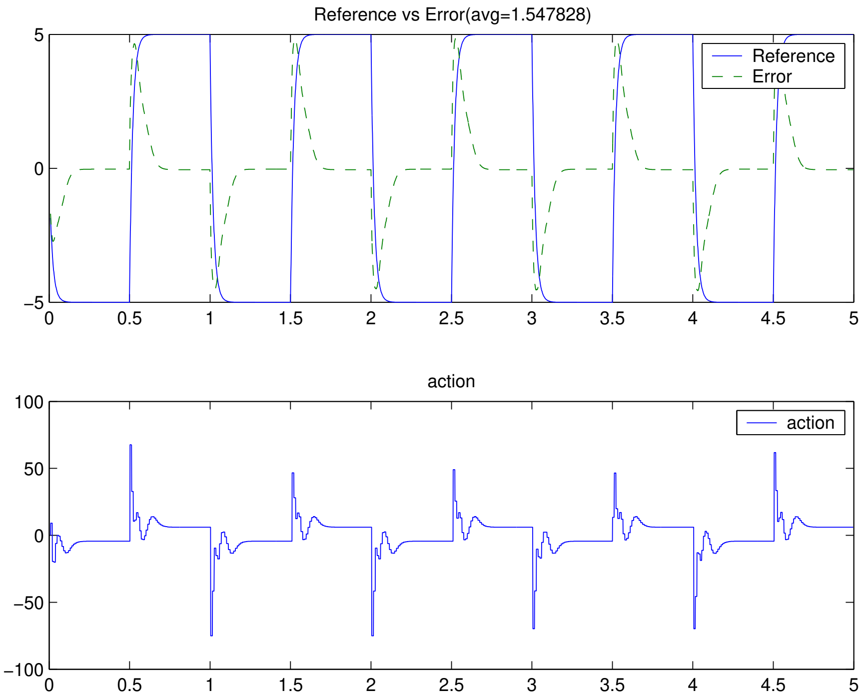

Figure 4.11: The system response with Value Gradient learning agent trained for 4000 seconds in Experiment 2

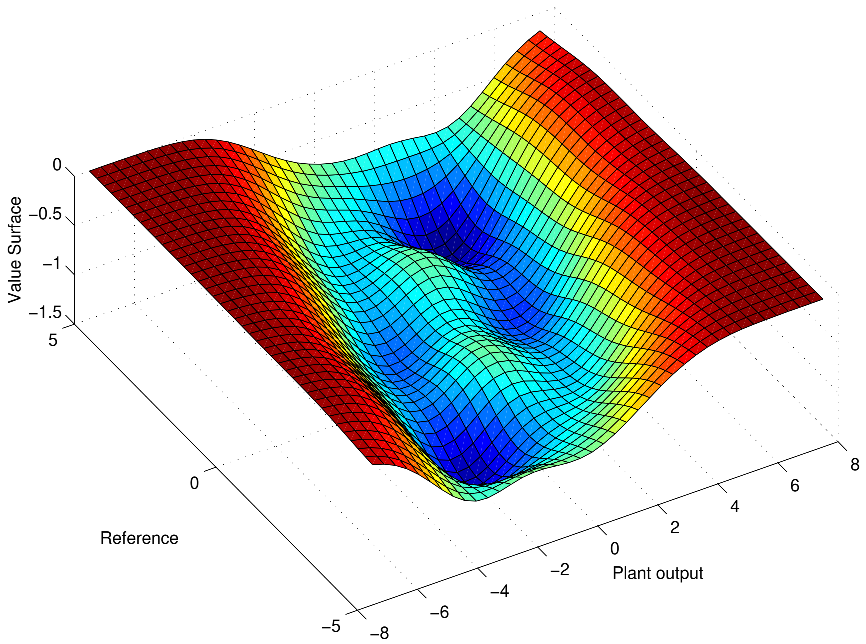

Figure 4.12: The Value Surface for Value Gradient learning agent in Experiment 2

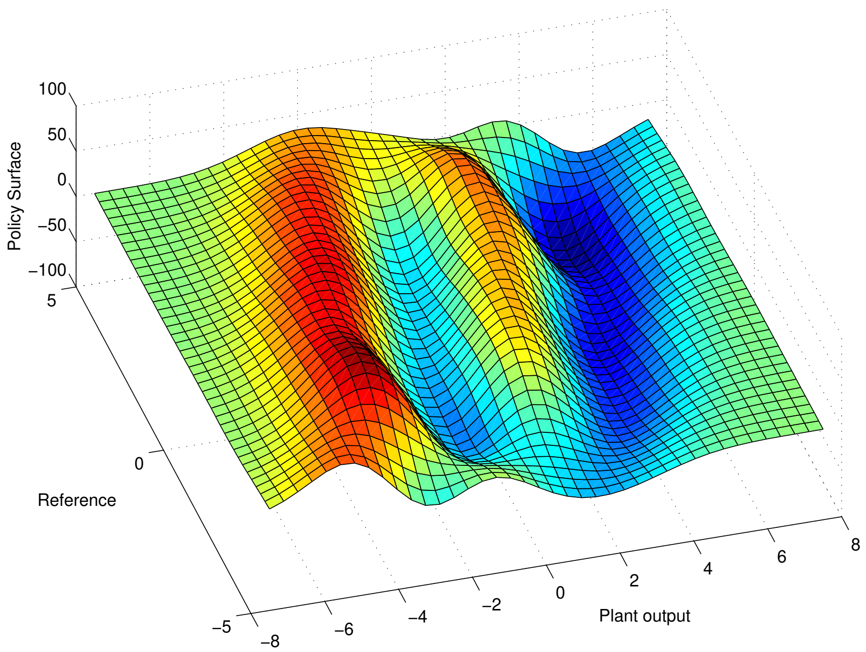

Figure 4.13: The action Policy Surface for Value Gradient learning agent in Experiment 2

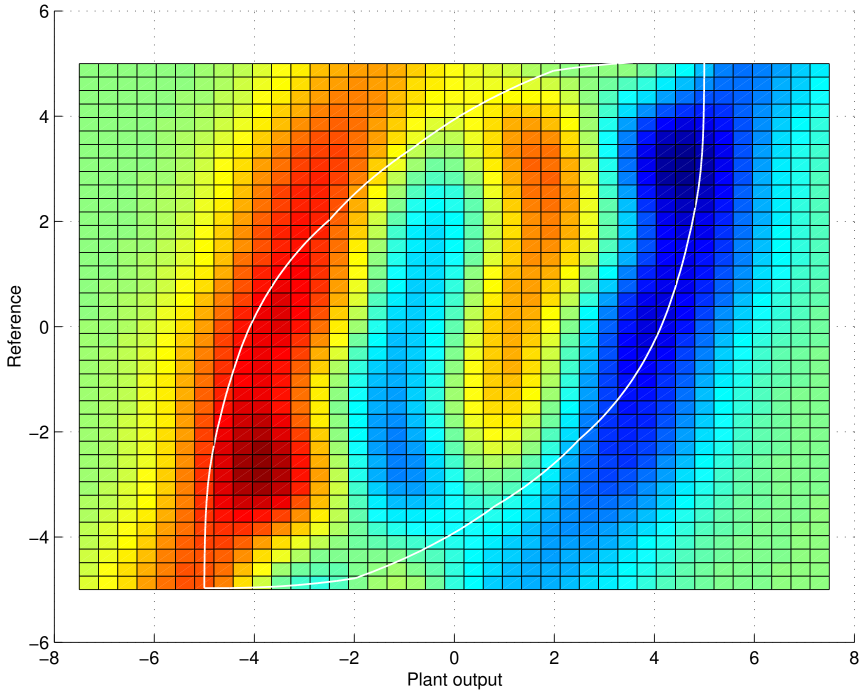

Figure 4.14: The trajectory of state transition on Policy Surface for Value Gradient learning agent in Experiment 2

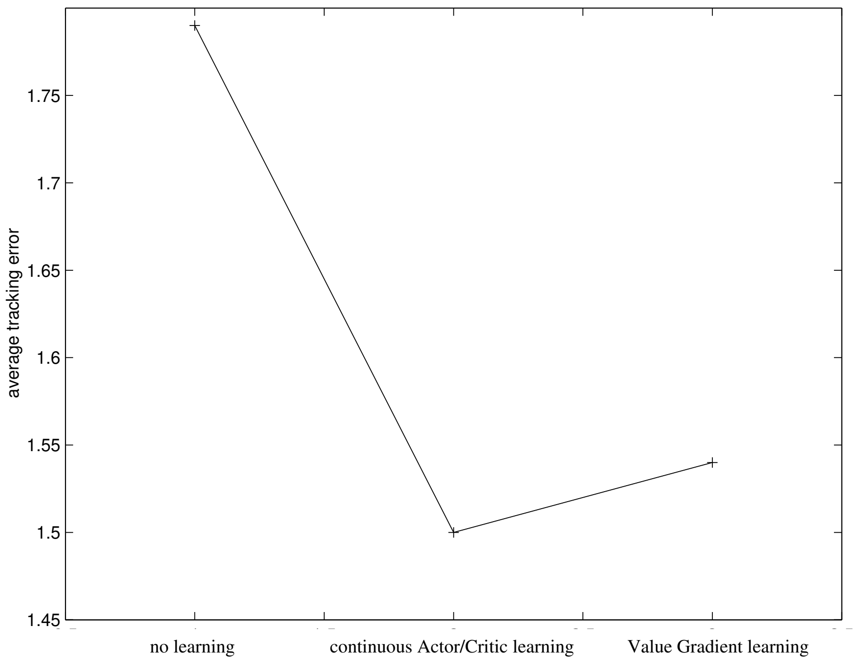

Figure 4.15: The average tracking error of no learning, continuous Actor/Critic learning, and Value Gradient learning in Experiment 2

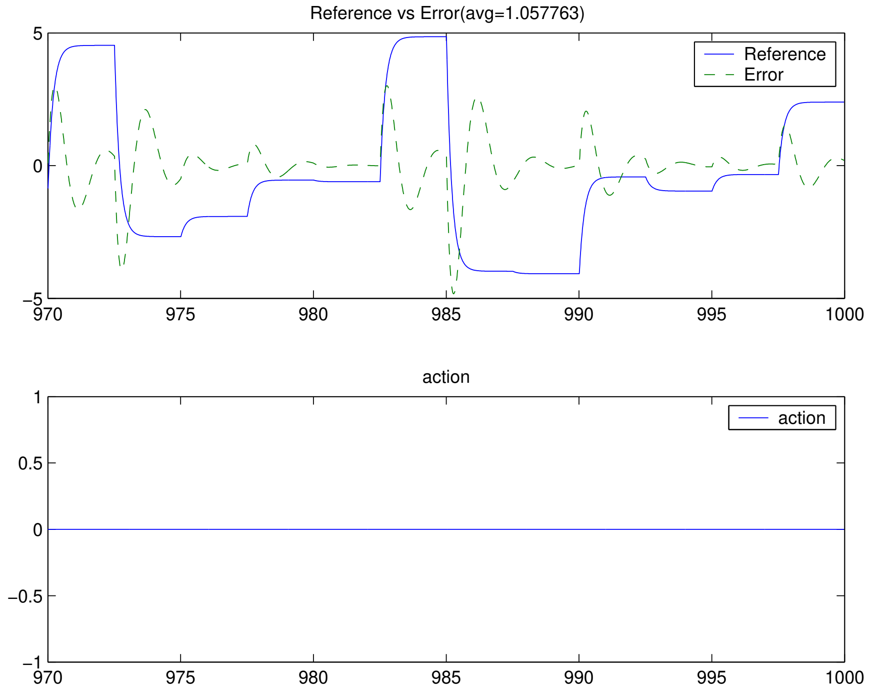

Figure 4.16: The system response without RL controller in experiment 3

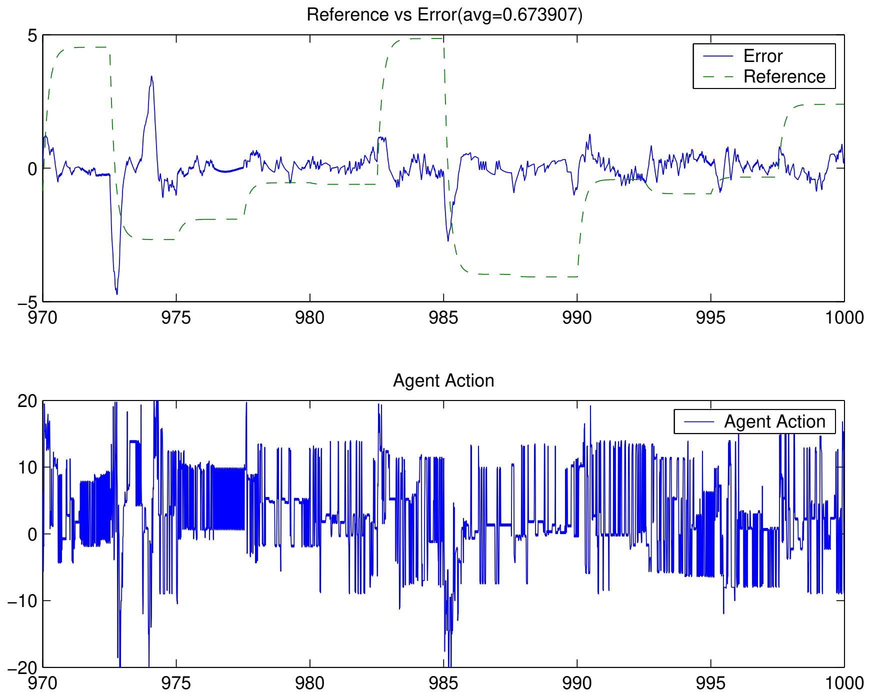

Figure 4.17: The system response with discrete RL controller trained for 10000 seconds in Experiment 3

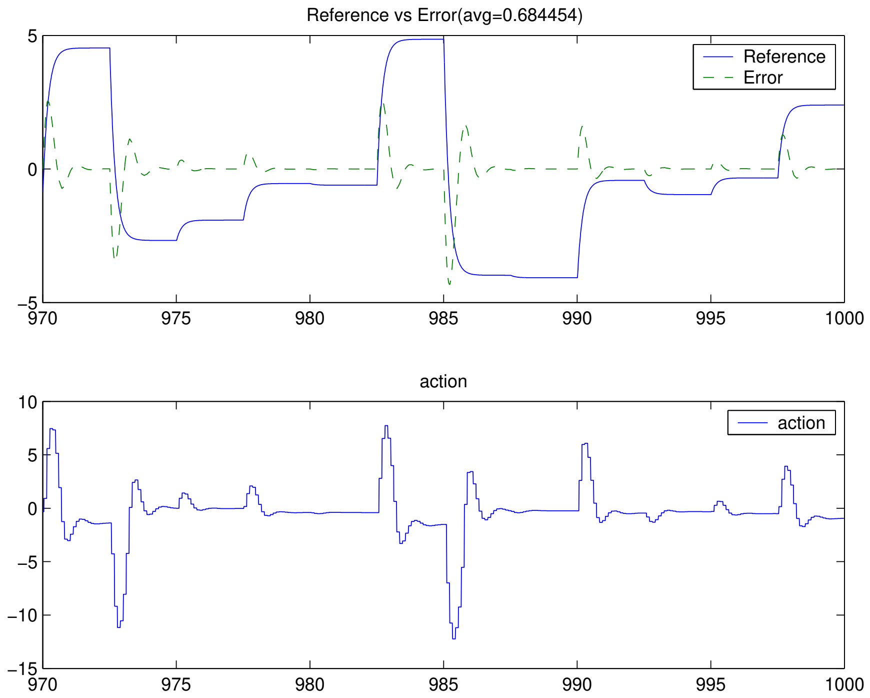

Figure 4.18: The system response with Actor/Critic trained for 10000 seconds in experiment 3

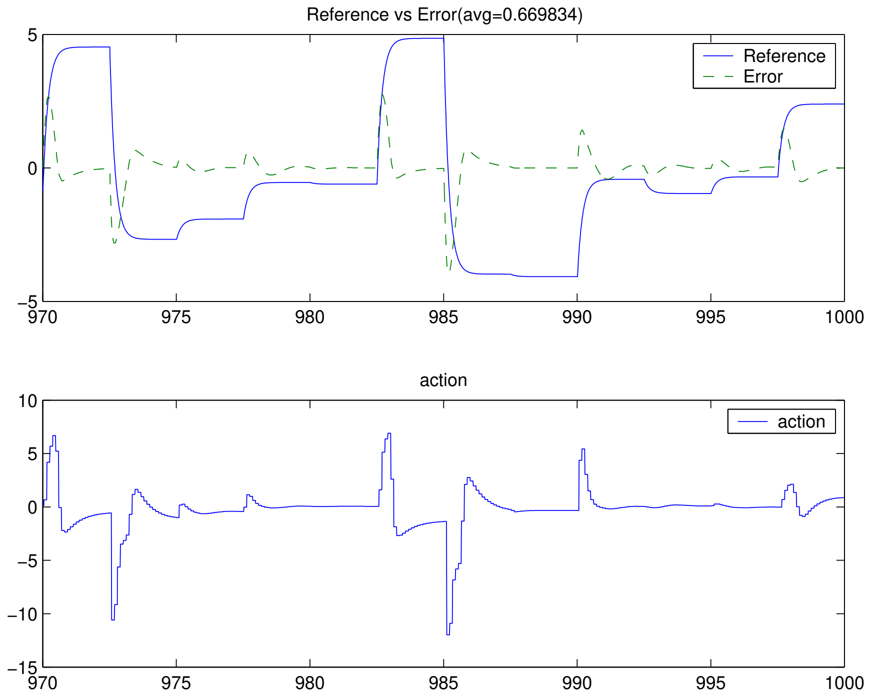

Figure 4.19: The system response with Value Gradient learning agent trained for 10000 seconds in experiment 3

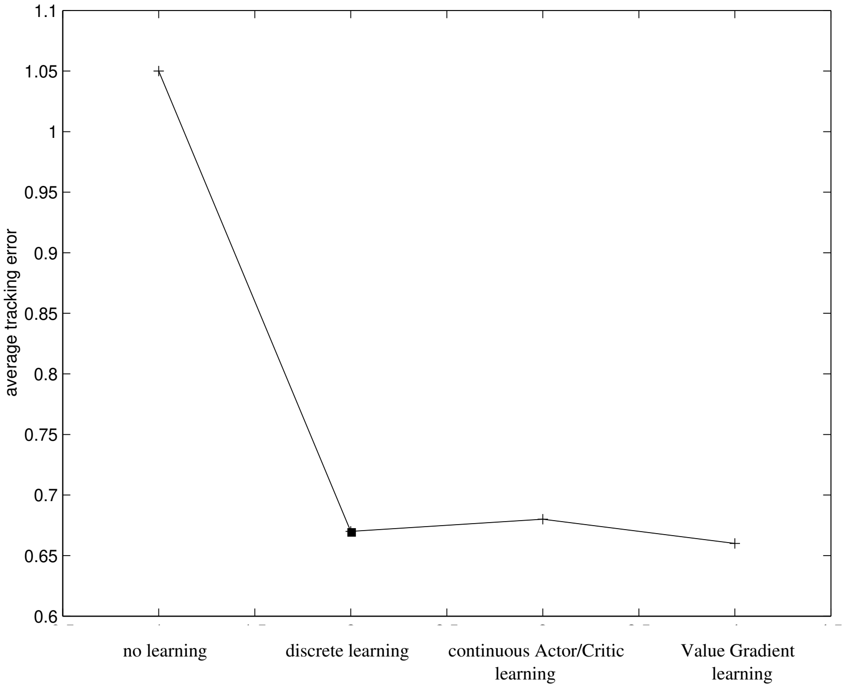

Figure 4.20: The average tracking error of no learning, discrete learning, continuous Actor/Critic learning, and Value Gradient learning in Experiment 3, the performance of discrete learning is marked with a black dot because its control sequence contains high frequency oscillations

Chapter 5

Conclusion and Future Work

The following progress were made in this thesis:

1. The cause of a learning problem that occurred when a reinforcement learning algorithm was applied to a control system with a PI controller was analyzed. The learning does not converge to optimal control because the learning problem is not Markov as the learning agent does not have access to some internal states of the system.

2. Based on the analysis of the cause, a new learning architecture was proposed based on an analysis of the statespace model of the feedback control system. Experiments showed that this new architecture outperformed the previous learning architecture.

3. Two continuous reinforcement learning algorithms were studied and applied to the feedback control problem. To adapt the algorithms to this specific problem, a new action cost function was proposed. Experiments were carried out to illustrate the mechanism of the continuous learning strategy.

4. The experiments showed that the two continuous reinforcement learning algorithms outperformed the discrete learning algorithm. The control sequences of the continuous learning agents are more smooth and contain no hazardous high frequency oscillations which were generated by the discrete learning agent.

Therefore, based on the findings in this thesis, we can improve the control performance using reinforcement learning techniques in feedback control systems.

However, this thesis only studied how to apply the new reinforcement learning architecture to a feedback control system with a simple plant $\frac{1}{s+1}$. There is still much work to do for a more complex physical control system.

The first problem is how to extract the states in the physical control system. Unlike the plant $\frac{1}{s+1}$, it might be difficult to analyze and access the internal states of a physical plant. The solution for this could be a Kalman filter or a recurrent neural network that can estimate the internal states based on the past plant outputs.

When the number of states is large, the "curse of dimensionality" becomes severe. There are two possible solutions to this problem. First, we can consider the learning problem as a kind of Partially Observable Markov Decision Process (POMDP) and utilize some reinforcement learning techniques that attack these POMDP problems (Parr & Russell, 1995). Second, we can utilize some function approximation techniques that can dynamically allocate bases in the space in an economical way while eliminating the approximation error.

As the learning always takes a very long time, it is possible to employ a parallel approach for the training. If it is possible to duplicate the model of the control system, several learning agents can be trained with these models in a parallel way. After the training, the learned knowledge can be synthesized and a more experienced learning agent can be generated. The training time could be reduced largely depending on how many copies of learning agents are trained.

In our experiments, the parameters were chosen based on experience. More work should be done to analyze how the selection of parameters influences the learning performance, so that the learning performance can be optimized.

Appendix A

Object Oriented Design for Reinforcement Learning

A.1 Principles and Experiences of the OOD

In reinforcement learning, the learning agent receives information from the environment and takes actions correspondingly, the environment is changed by the agent's action, and provide new feedback to the agent. This process of interaction between learning agent and environment is typical in most of the reinforcement learning problems, therefore it is convenient to model this class of learning problems with some design patterns when programming for intelligent agent learning.

We designed our intelligent learning system generally based the following principles:

1. The learning algorithms should be loosely coupled with the specific learning problem so that they can be reusable when they are applied to new learning problem.
2. The learning algorithms and environment should be reconfigurable with the minimal modification of code.
3. The interface for visualization should be flexible and machine-independent.

According to the first principle, the system is divided into three components: the Agent, the Environment, and the Simulator. The Agent implements the learning algorithm. The Environment implement different learning problem. The Simulator encapsulates the Agent and Environment, and mediates how they interact. An ideal implementation should allow users to replace the agent for different algorithms and replace the environment component for different learning problems with minimal modification of code. Besides, the system should also provide a sensory component which allows users to collect data on some internal variables during learning to study what is going on inside the system. The structure of the object-oriented design is shown in Figure A.1.

To accommodate the second principle, we created a data directory that contains the parameter configuration files for each object component. Users can reconfigure the learning algorithm and the system by modifying the parameters. When the learning algorithm is upgraded, and new parameters have to be added to the configuration file, the changes are confined within the component.

The Sensors collect data for visualization. The Sensors can visualize the data by themselves, or just store the data in a specified file. Commercial software, such as Matlab, can be employed to do the data analysis and visualization.

A.2 Implementation of Our Simulation for Continuous Reinforcement Learning in Feedback System

As our simulation system is not for commercial usage, we did not rigorously follow the design principles in implementation. However, a loose obedience of the design principles still brought us

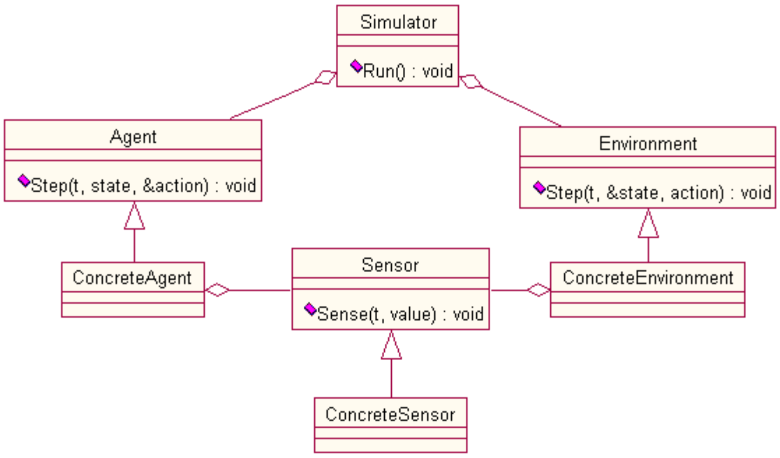

Figure A.1: The software architecture of interactive learning system

flexibility and saved us a lot of time when we were upgrading the algorithms and generating the experiment results. The following is a brief introduction on how our simulation system works.

A.2.1 Simulator

The Simulator defines the length of simulation and sampling rate, according to which, the method *Run()* implements a loop that allows the Agent to interact with the Environment by passing the information around in variables *state* and *action*. Besides, the simulator is also responsible for evaluating the system performance during the learning.

A.2.2 Agent

In our system, the Agent is composed of an Actor and a Critic. The Critic updates the value surface and provides information to Actor for calculating the best action. To approximate the value surface and policy surface, a function approximator *RbfSurface* is implemented. This function approximator can be reconfigured for approximation problems in a space of arbitrary dimensions. It also provide methods for calculating the derivatives with respect to the weights or the states, and generating the profile of the surface in any two dimensional subspace.

The Actor is responsible for evaluating the reward of its action. In the Value Gradient algorithm, the Actor can be reconfigured with different action reward functions, and the corresponding method for calculating the best action will be employed automatically.

A.2.3 Environment

The Environment is composed of a signal generator, a PI (or P) controller, and a plant that is defined by a statespace model. For each component, a configuration file is created and stored in the data directory. Therefore, users can reconfigure the Environment by specifying a different signal generator, controller with different parameter, and different plant. The Environment is responsible for evaluating the reward of current state, and calculates the derivatives of system dynamics for the *Value Gradient* learning.

A.2.4 Sensors

We developed a *Scope* sensor which can record data in arbitrary format. The data is stored to a file with a specified file name at the end of simulation. The data file can be loaded into the Matlab workspace. Users can easily analyze and visualize the data as they wish in Matlab. Another sensor we developed is a system performance Estimator. The Simulator utilizes this sensor to collect and calculate the average tracking error of the feedback system as an estimate of the system performance. Users can turn on and off a sensor by changing the configuration files.

A.2.5 Run the Simulation

Given the implementation of the above components, an example main() function is:

```
void main(){
  LearnPara *p=LoadPara("data/para_main.dat");
  Env *es=new EnvControlSystem(p->envpara,p->envresult,p->restart);
  Actor *actor=new Actor(p->actorpara, p->actorresult,p->restart);
  Critic *critic=new Critic(p->criticpara,p->criticresult,p->restart);
  Simulator *s=new Simulator(p->simupara,es,actor,critic,p->restart);
  s->run();
}
```

Including different header files for different implementations of the Actor, Critic, Simulator, and EnvControlSystem components in the *Main* program, users can easily compose simulation systems for different purposes with the corresponding configuration files.

Bibliography

Anderson, C., Hittle, D., Katz, A., & Kretchmar, R. (1996). Synthesis of reinforcement learning, neural networks, and PI control applied to a simulated heating coil. *Journal of Artificial Intelligence in Engineering*, *vol. 11*, pp. 423–431.

Baird, L. C. (1993). *Advantage updating* (Technical Report WL-TR-93-1146). Wright Laboratory, Wright-Patterson Air Force Base.

Boyan, J. A., & Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. *Advances in Nerual Information Processing Systems 7.*

Bradtke, S. J., & Duff, M. O. (1994). Reinforcement learning methods for continuous-time Markov Decision Problems. *Nerual Information Processing Systems conference 1994* (pp. 393–400).

Doya, K. (2000). Reinforcement learning in continuous time and space. *Neural Computation*, *vol. 12*, pp. 219–245.

Doyle, J. C., Francis, B. A., & Tannenbaum, A. R. (1991). *Feedback control theory.* Prentice Hall PTR.

Gullapalli, V. (1990). A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, *vol. 3*, pp. 671–692.

Howell, M. N. (2000). On-line pid turning for engine idle-speed control using continuous action reinforcement learning automata. *IFAC Journal of Control Engineering Practice*, *vol. 8*, pp. 147–154.

Jobling, C. P. (1999). State space modelling. Department of Electrical and Electronic Engineering, University of Wales Swansea, http://www-ee.swan.ac.uk/Courses/level2/ee206/StateSpace/.

Kavehercy, S. (1996). Reinforcement learning & artificial neural networks: The optimal control of the water vessel process. Master's thesis, University of Amsterdam.

Kretchmar, M. (2000). *A synthesis of reinforcement learning and robust control theory.* Thesis (ph.d.), Computer Sciences Department, Colorado State University, Fort Collins, CO, USA.

Miron, D. B. (1989). *Design of feedback control systems.* Harcourt Brace Jovanovich.

Morimoto, J., & Doya, K. (2000). Robust reinforcement learning. *Nerual Information Processing Systems 77, 2000.*

Munos, R. (1997a). A convergent reinforcement learning algorithm in the continuous case based on a finite difference method. *International Joint Conference on Artificial Intelligence 1997.*

Munos, R. (1997b). Reinforcement learning for continuous stochastic control problems. *Advances in Neural Information Processing Systems, 1997.*

Pareigis, S. (1997). Adaptive choice of grid and time in reinforcement learning. *Nerual Information Processing Systems-97.*

Parr, R., & Russell, S. (1995). Approximating optimal policies for partially observable stochastic domains. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*.

Russell, S., & Norvig, P. (1995). *Artificial intelligence: A modern approach*. Prentice Hall.

Skogestad, S. (1996). *Multivariable feedback control analysis and design*. John Wiley & Sons Ltd.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems 8* (pp. 1038–1044). The MIT Press.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT Press, Cambridge, MA.

Thrun, S. (1993). Issues in using function approximation for reinforcement learning. *Proceedings of the Fourth Connectionist Models School*. Hillsdale, NJ: Lawrence Erlbaum Publisher.

Watkins, C. (1989). *Learning from delayed rewards*. Doctoral dissertation, King's College, Cambridge, UK.

Weaver, S. E., Baird, L. C., & Polycarpou, M. M. (1998). Preventing unlearning during on-line training of feedforward networks. *Proceedings of the 1998 IEEE International Symposium of Intelligent Control, Gaithersburg, MD, Sept 14-17* (pp. 359–364).