Adaptive Control based on Adjustable Evaluation Function

Graduate School of Systems and Information Engineering

University of Tsukuba

March, 2007

Agus Naba

Acknowledgments

First of all, an enormous credit is due to Kazuo Miyashita, Ph.D., my advisor for four years. I have benefited tremendously from his guidance and support. Without his great patience, close collaboration and careful guidance, I would not be able to accomplish my doctor study and this dissertation. I would like to thank Prof. Seiichi Nishihara who helped me for things related to academic affairs. I would like also to thank Prof. Sadaaki Miyamoto, Prof. Seiji Yasunobu and Prof. Yukio Fukui. While serving as committee members, they provided excellent technical advice to make the dissertation a more coherent document. Thanks is due to all members of the committee for countless hours of document reading and editting.

I would like to thank Dr. Haruhisa Kurokawa, the group leader of the laboratory of the Distributed System Design Research Group of AIST where I worked for three and a half years. While having his own heavy works, he helped and shared to me his valuable experiences in doing good research and writing. I am also grateful to all members, secretaries, and technical staff of the laboratory for their kindness and willingness to help me anytime that make me feel at home. Having many useful interactions, academic, and otherwise in Tsukuba, I owe a special thanks to all of my tutors, friends, and others that I could not mention one by one.

I am deeply indebted to my parents and all of brother and sisters who provided support and motivation. I wish also to thank my parents-in-law. Finally, I devoted this dissertation to my beloved wife, daughter and boy.

Abstract

Every control design involves manipulating a controller so that a controlled plant behaves as desired. Various methodologies for manipulating the controller for certain well-defined classes of control problems have been well developed and proven. Despite great efforts, similar techniques are not available for more general classes of control problems involving nonlinear systems. Incorporating adaptive methods for manipulating the controller has been a widely used alternative.

In many adaptive control system designs, the control objective is defined as to minimize the plant output error (i.e., the error between a goal and an actual plant state), assuming that smaller plant output errors imply better instantaneous control performances or immediate rewards. However, in many control problems, executing the actions that are predicted to result in smaller plant output errors only can mislead to non-goal states. This is because smaller plant output errors do not always imply better instantaneous control performances, i.e., the plant output error is not always an *evaluative* information.

In Reinforcement Learning (RL), the evaluative information about action performances is assumed not readily available. In contrast with a customary in adaptive control designs where the actions are rewarded based on the control performance measure defined a priori by an engineer, RL views that the actions must be rewarded by the environment, not by the engineer. RL rewards the actions based on a value function, i.e, a function that represents "goodness" of a state or a state-action. The value function is learned only from the long-term consequences of executing actions (i.e., being given a reward or a penalty) in every iteration, and then used to decide control actions. A precise value function simply recommends to take an action that can lead to

the next state with the highest value. In general, a certain defect of RL is that many time-consuming trials-and-errors are often required to learn a precise value function.

In the dissertation, the author presents alternative approaches to solve the above problems. In general, the developed approaches are inspired by a policy search approach of RL in which a policy (i.e., a mapping from state to action) is represented using an independent function with its own parameters. The policy search approach tunes the policy parameters to improve the control performance as measured with the value function being learned. Similarly, the developed approaches tune the controller to improve the control performance as measured with the evaluation function. The evaluation function appropppriate for the developed approaches is required to

i) represent stable dynamics of the closed loop system (i.e., consisting of the plant and the controller) in term of the control performance, implying that it can predict the actions that lead to maximum total reward in the long run.

ii) provide a direct measure of its gradient with respect to the action by which tuning of the controller parameters using the gradient method is possible.

Such an evaluation function differs from the control performance measure widely used in many adaptive control designs. It includes *prediction* of the long-term consequences of executing action, while the control performance measure in those many adaptive control designs does not. The author supposes its role as similar to the role of the value function in RL.

Although finding the appropriate evaluation function seems as difficult as obtaining the precise value function in RL, major points made in this dissertation is that

i) the evaluation function that satisfies the above requirements can be approximated,

ii) approximation errors in the evaluation function can be coped with through on-line tuning of the approximate evaluation function,

iii) using the approximate evaluation function, the adaptive control schemes can be practical, reliable, and efficient.

To justify those claims, the author carried out several experiments using two types of the adaptive control schemes:

1) adaptive control with a fixed approximate evaluation function,

2) adaptive control with an approximate evaluation function adjusted online.

The first type of the proposed adaptive control is supposed to work for simple control problems in which the closer state to the goal state implies that the smaller actions are required. For the simple control problems, the evaluation function is often relatively easy to approximate and readily available, and therefore, it does not need to be learned. The second type is supposed to work for more complex control problems with the

assumption that the plant state is continuous everywhere in a state space and imprecise plant knowledge for the engineer to make a "good" guess of initial controller parameters is available. Approximation errors of the evaluation function is coped with through online adjustment. However, the adjustment of the approximate evaluation function does not need to wait for the long-term consequences of executing actions, i.e., the rewards from the plant, rather it uses readily available information about the state, the action, and the result of executing the action, i.e., the next state. Such an approach saves computation time for finding appropriate controller parameters. Finally, the author applies the proposed adaptive control methods to solve two benchmark control problems: a pole balancing problem and a cart-pole balancing problem. Experimental results on both control problems show that the first type of the proposed adaptive control schemes is effective for solving the pole balancing problem, but inapplicable for solving the cart-pole balancing problem. Nevertheless, the second type of the proposed adaptive control schemes is proved to be effective for solving the cart-pole balancing problem.

Table Of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

1.1 Problem

A goal of every control design is to manipulate a controller so that a controlled plant behaves as desired. By the term "plant" in the dissertation the author means a generic term for a system *controllable* by the manipulated controller. Various methodologies for designing the controllers for certain well-defined classes of control problems have been well developed and proven by control theorists since the early nineteenth century. Most of those methods are designed for solving control problems involving linear processes and certain types of cost functions, such as quadratic performance index. The controllers developed with those methods have been proven stable, have desired response characteristics, or perform optimally. However, despite great efforts, similar techniques are not available for more general classes of control problems involving nonlinear systems [1]. Incorporating adaptive methods into the controller has been one alternative to solve those control problems.

Conventional control techniques are developed based on mathematical model of the plant to be controlled. Those techniques work under the assumption that the plant model accurately replicates all relevant aspects of the plant behavior. This implies that the controller designed using those techniques should be able to perform well with the real plant. Consequently, the controller performance critically depends on the accuracy of the plant model. And, we can predict that when the controller exhibits poor

performance in the real applications, it is most likely due to uncertainty or unmodelled dynamics of the controlled plant. In many control problems, however, it is difficult to model the plant precisely. In these cases, the engineer usually relies on heuristics and the engineer's experiences in modelling the plant. The engineer also sometimes directly designs the controller without reference to any explicit plannt model, instead with domain-qualitatively specific knowledge of the plant behavior. Again, the engineer heavily relies on heuristics to determine the desired controller. Thus, the above facts strengthen the need of incorporating the adaptive methods into the controller.

There was extensive research on adaptive control since a half-century ago. Nevertheless, the adaptive control remains by no means a mature field where many developed approaches and algoritms are of ad hoc nature and good systematic approaches are still lacking [2]. In many adaptive control designs [2–6], a desired behavior of the plant is defined in terms of a setpoint or reference trajectory, respectively corresponding to a regulation problem in which a plant output has to stay at the setpoint, and a tracking problem in which the plant output has to follow the reference trajectory. In achieving that desired plant behavior, the common assumption made in those adaptive control designs is that the plant output errors (i.e., between an actual plant state and the set-point or the reference trajectory) getting smaller are *direct* indications that the actions taken are "correct", i.e., they will lead to the goal state eventually. Otherwise, the actions are to be "blamed". Or equivalently, smaller plant output errors are assumed to imply better instantaneous control performances or immediate rewards, and vice versa. Because the plant output errors are readily available, whatever an action it takes, the controller always knows without any "effort" whether the action is correct or to be blamed, only based on the plant output error due to application of that action. With these assumptions, the control objective is then defined simply as to minimize the plant output errors all times. There is no need to try a variety of other actions once an action is predicted to result in the smallest plant output error at the next time step.

In his book [7], Sutton gives an excellent illustration that minimizing error every time step is not the only way to achieve the goal state. Call the action predicted to result in the smallest plant output error at the next time step as a "greedy" action. If we think that a greedy action is the "best" action to take to lead to the goal state, we are said exploiting our current knowledge of "value" of the actions. If instead we select one of the nongreedy actions despite we know the greedy action, then we are said exploring our current knowledge of the value of the actions because this enables us to improve our estimate of the nongreedy action's value. Exploitation is the right thing

to do to find the "best" action only at a time instant, but exploration may produce smaller total plat output errors in the long run. For example, suppose that the greedy action's value is certainly known. While, several other actions are estimated to be almost good but with uncertainty. The uncertainty applies when at least one of those other actions is probably actually better than the greedy action, but we don't know which one. In such a situation, it may be better to explore the nongreedy actions and discover which of them are better than the greedy action. That is, the total plant output errors are larger in the short run, during exploration, but getting smaller in the long run because the exploration enables us to discover the better actions that can be exploited further. This can be referred to as "conflicting" between exploration and exploitation [7]. The above concept and its applications are also discussed in detail in [8–11] based on reinforcement learning (RL) framework. They convinces us that executing the actions that are predicted to result in smaller plant output errors only can mislead to non-goal states. This is because the plant output errors are more *instructional* rather than *evaluative*, i.e., smaller plant output errors do not always imply better instantaneous control performances.

In RL, the evaluative information about action performances (i.e., the value of the action) is assumed not readily available and not simply equivalent to the plant output errors. In contrast with a customary in adaptive control designs where the actions are rewarded based on the control performance measure provided a priori by an engineer, RL views that the actions must be rewarded by the environment, not by the engineer [12, 13]. The reward is usually a *weak* evaluative feedback information, such as simply a *failure* or *success* signal. Using the received rewards, RL learns on-line a value function, i.e., a function that represents "goodness" of a state or a state-action. The value function is originally of the form of look-up tables from which a policy (i.e., a mapping of states to optimal actions) is then deduced.

However, in general, any learning method including RL is of a fundamental problem that has been called the "credit-assignment" problem by artificial intelligence researchers. That is the problem of determining what individual actions or in general what parts of a complex interacting set of mechanisms, decisions, or actions deserve credit or blame for improvement or decrement in the overall performance of the learning system. The credit-assignment problem is especially acute when evaluative feedback occurs infrequently, for instance, upon the completion of a long sequence of actions [7,8]. Such a problem leads to a certain defect of RL, i.e., many time-consuming trials-and-errors are often required to learn the precise value function.

There have been large applications of RL to solve the control problems. The value-function approach has been the dominant approach which requires the use of function approximators such as neural networks, decision-trees, or instance-based methods to generalize the value-function [14]. All function approximation effort need to involve estimating and generalizing the value function. An action selection policy is then represented implicitly as the "greedy" policy with respect to the estimated values, e.g., as the policy that selects in each given state the action with the highest estimated value. The value-function approach has lead to many successful applications, but with several limitations. It is limited to find deterministic policies that make it only best applied to the control problems with constant goal states. The action selection policy can be very sensitive to an arbitrarily small change in the estimated value, which has been identified as a main obstacle to establishing convergence assurances of algorithms following the value-function approach. The examples of the approach with such shortcomings are Q-Learning, Sarsa, and dynamic programming methods [15–17]. Those approaches have been proved unable to converge to any policy for simple function approximators [14].

Alternatives to function approximation in RL have been algorithms based on "actor-critic" or policy-iteration architecture (see [8, 10, 11, 14, 18–20]) where the policy is represented using an independent function approximator with its own parameters. For short, such algortihms are referred to as a policy search approach in the dissertation. Using a gradient method, the policy search approach updates the policy parameters approximately proportional to a gradient of a performance measure with respect to those policy parameters. The performance measure is an approximation of the value function. The use of the gradient method is simply due to the fact that a precise value function simply recommends to take an action that can lead to the next state with the highest value, which is similar to the property of a Lyapunov function. A certain disadvantage of the policy search approach (also, of RL in general) is that the approach must learn the value function first. To obtain the best approximation of the value function, however, the approach requires time-consuming trial-and-error interactions between the controller and the plant. In addition, while accuracy of the value function is not guaranteed during the learning process, the gradient method requires precise value function anyway.

In the dissertation, the author presents alternative approaches to solve the above problems. In general, the proposed approaches are inspired by the policy search approach of RL but expected applicable for solving adaptive control[1] problems that involve modifying the controller to improve the control performance as measured with an evaluation function. The evaluation function is required to have a similar role as the role of the value function in RL, but readily available and used, or when it is not readily available, it can be improved through adaptive tuning without time-consuming trials-and-errors as in RL. However, it is impossible that we can obtain a general appropriate evaluation function which can be used to measure the control performance of any adaptive control. Therefore, finding an appropriate evaluation function and specifying the type of the adaptive tuning problems that can use that appropriate evaluation function are the main problems to be solved first in designing the adaptive control based on the proposed approaches.

1.2 Objective

In general, the author desires an adaptive control that "combines" two distinct design philosophies, i.e., the philosophies of classical adaptive control and RL. The classical adaptive control is usually not time-consuming due to not relying on the long-term rewards to improve the control performances, but its applicability is often limited with its representational inflexibility due to a presupposed plant model structure. In contrast, RL does not necessarily presuppose any plant model structure, i.e., it is model-free, and therefore, it is of high applicability. But, it relies entirely on the long-term value function to improve the control performance which leads to low efficiency. The adaptive control developed in the dissertation is desired to have increased applicability and efficiency as illustrated in Figure 1.1.

Although difficult to achieve during the author's doctor study, a final target of the author's research is to design adaptive control with the above properties where a controller is tuned using a gradient method to improve control performance as measured with an evaluation function. The most important task in achieving that target is to

[1] In the adaptive control designs, there is a customary to separate the *tuning* problems from the *adaptation* problems. In the tuning problems it is assumed that the parameters of the plant to be controlled are constant; in the adaptation problems it is assumed that the parameters are changing [2]. Although allowing the plant parameters to vary considerably slower than the parameter adaptation [3], most adaptive control methods are designed for the former problems. In the adaptive control problems considered in the dissertation, the unknown plant parameters are assumed *constant* or *almost constant* (or, *very slowly-changing*), i.e., they are more suited to deal with the tuning problems.

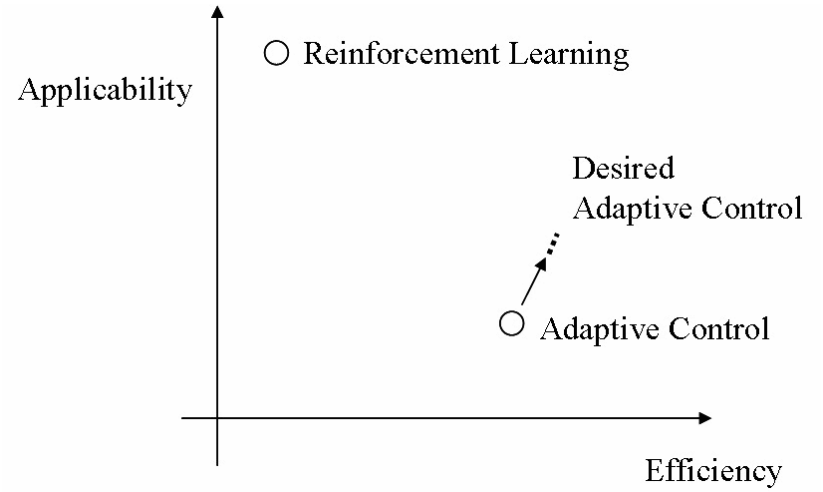


Figure 1.1: Philosophy of the desired adaptive control design. The design is desired to work *in between* two design philosophies, i.e., the philosophies of a typical adaptive control and RL.

find the appropriate evaluation function. The evaluation function approppriate for the adaptive control developed in the dissertation is required to

i) represent stable dynamics of the closed loop system (i.e., consisting of the plant and the controller) in term of the control performance, implying that it can predict the actions that lead to maximum total reward in the long run.

ii) provide a direct measure of its gradient with respect to the action by which tuning of the controller parameters using the gradient method is possible.

Such an evaluation function differs from the control performance measure widely used in many adaptive control designs, i.e., it includes *prediction* of the long-term consequences of executing action, while the control performance measure in those many adaptive control designs does not. The author supposes its role as similar to the role of the value function in RL.

Although finding the appropriate evaluation function seems as difficult as obtaining the precise value function in RL, major points made in this dissertation is that for certain types of control tuning problems

i) the evaluation function that satisfies the above requirements can be approximated, so that it is readily available and usable for evaluating and improving the controller in the adaptive control,

ii) approximation errors in the evaluation function can be coped with through on-line tuning of the approximate evaluation function,

iii) using the approximate evaluation function, the adaptive control schemes can be practical, reliable, and efficient.

Those certain type of the control problems are briefly described in the following section that outlines the strategies the author uses to accomplish the research objectives.

1.3 Strategy

The controller can be trained using a supervised learning method, i.e., the method that learns from examples, provided that good examples of control actions that improve the control performance as measured with the evaluation function are available. When the examples of the control actions are not known, appropriate behavior of the controller needs to be inferred from observations of the appropriate evaluation function. In such cases, two different classes of learning control methods to train the controller can be used, i.e., *indirect*[2] method and *direct* method [1]. The indirect method involves constructing a *model* of the evaluation function in a form that can be readily used to generate information for training the controller. Its implementation consists of two stages: constructing an adequate model of the evaluation function, which is similar to a problem of finding the precise value function in RL, and using the model to generate information for training the controller. In contrast, the direct method does not rely on the evaluation function of a certain model, rather a general form evaluation function. It involves actively applying to the plant random actions or the actions chosen stochastically and observing the consequences on the evaluation function. Policy search approach of RL [8, 14, 22] is one example of the direct method, which can be used to better describe how the direct method works. In the policy search approach of RL, the value function is not constructed using a certain model associated with any specific plant model or closed loop dynamics, rather a model-free scalar function of the reward received at the next time step plus a properly discounted value of the next state-optimal actions. After perturbing the the plant using the actions chosen randomly or deduced from the value function being learned, the approach observes consequences of executing those actions on the value function, uses those consequences to update the value function, and then updates the controller based on the updated value function.

[2]Adaptive control literatures [2,3,5,21] also uses the terms "indirect" and "direct" in control design methods, where the former term refers to determination of the controller parameters from a plant model constructed on-line via system identification, while the latter term refers to determination of the controller parameters without contructing an explicit plant model. The definition of both terms the dissertation refers to is the same as given by Gullapalli [1] who uses both terms in the RL designs for control.

A potential disadvantage of such a direct method is that the value function obtained through active perturbation can be less reliable to represent the stable dynamics of the closed loop system in terms of the long-term control performance measures, implying that the controller updated based on it can produce the actions that mislead to non-goal states. While, learning the precise value function requires many time-consuming trials-and-errors. In such situations, the direct method is often slower than the indirect method.

In the dissertation, as the attempts to obtain the aforementioned desired adaptive control, the author developes two types of the adaptive control schemes:
1) adaptive control with a fixed approximate evaluation function,
2) adaptive control with an approximate evaluation function adjusted online.
Both types of the proposed adaptive control are implemented using an approach similar to the indirect method. The difference is that the evaluation function is represented using a *model*[3] that can provide a direct measure of its gradient with respect to control action, rather than be used to generate requisite information for training the controller. The gradient is then used to tune the controller parameters.

The first type of the proposed adaptive control is supposed to work for simple control problems in which the closer state to the goal state implies that the smaller actions are required. For the simple control problems, the evaluation function is often readily available or unavailable but relatively easy to approximate using a simple evaluation model. Therefore, it does not need to be learned.

The second type is supposed to work for more complex control problems with the assumption that the plant state is continuous everywhere in a state space without drastical changes between time steps and imprecise plant knowledge for the engineer to make a "good" guess of initial controller parameters is available. Using those assumptions, the evaluation function is approximated using a model more general than that of the evaluation function in the first type of the proposed adaptive control. Nevertheless, for the complex control problems, parameters of the approximate evaluation function model are difficult to determine a priori. Hence, online adjustment of the evaluation functionn is necessary to cope with approximation errors and impreciseness of the evaluation function parameters. However, the adjustment of the approximate evaluation function does not need to wait for the long-term consequences of executing actions, i.e., the rewards from the plant, rather it uses readily available information about the state,

[3]In the dissertation, by "model" the author means a model that represents a stable dynamics of a closed loop system (consisting of the plant and the controller) in terms of long-term future control performance measures.

the action, and the result of executing the action, i.e., the next state. Such an approach saves computation time for finding appropriate controller parameters. Finally, the author applies the proposed adaptive control methods to solve two benchmark control problems: a pole balancing problem and a cart-pole balancing problem.

The idea of tuning the approximate evaluation function online and then using it for tuning the controller is not new as it can be found in many literatures (see [23–31]) which developed the methods for finding appropriate actions based on adaptive critic approach and approximate dynamic programming (AC/ADP). AC/ADP is essentially a juxtaposition of RL and Dynamic Programming (DP) ideas [31]. While DP derives the control actions via the optimal value function, AC/ADP utilizes an approximation of the optimal value function as the evaluation function to adjust the controller. Despite its successful applications to reduce inefficiency of RL, the adaptive control using AC/ADP is limited by the need of "crafting" an appropriate utility function (i.e., equivalent to the reward function in RL, *except* that the reward function is not "crafted" by the engineer but provided by the environment). The utility function is the only source of information required in AC/ADP to improve the approximate value function, which is defined simply as equal to the utility value plus the discounted approximate value of the next state or the next state-action. The utility function is often simply defined as a Eucleadian distance (or, the error) from the goal state [23, 26]. Therefore, it is potential to cause the same control difficulties as in the classical adaptive control methods whose goals are simply to minimize the error. Alternatively, different relative weightings might be used for some of the error components as done in [29,30] for specific control problems. However, other research of [31] provides examples of situations where seemingly minor changes in formulation of the utility function resulted in a dramatically different ADP convergence behavior and a controller design. Generally, defining an approparite utility function even for a specific control problem is not always an easy task. A common practice to cope with this problem is by performing some trial-and-error modifications of the utility function. The architecture of the adaptive control developed in the dissertation is inspired by that of the policy search approach of RL. But, the proposed adaptive control is designed not to rely on any utility or reward function to improve the approximate evaluation function. Nevertheless, no trial-and-error is required to improve the approximate evaluation function. Thus, it should be more practical and efficient than the adaptive control in those aforementioned literatures.

1.4 Organization of Dissertation

The relative merits of the typical adaptive control and RL are discussed in Chapter 2, where the author elaborates the tradeoffs between consequences due to the usage of readily available but non evaluative feedback information in the typical adaptive control design and those due to the usage of unreadily available but evaluative feedback information in RL. Detailed comparison of both approaches are also discussed at the last section of that chapter. The first type of the proposed adaptive control schemes, i.e., the adaptive control with a fixed approximate evaluation function, is presented in Chapter 3, followed with experimental results of its application to the pole-balancing problem using two types of controller, i.e., a linear state feedback controller and a fuzzy controller. Chapter 3 ends with the analysis of the experimental results. In Chapter 4, the author describes in detail the second type of the proposed adaptive control schemes, i.e., the adaptive control with on-line tuning of the approximate evaluation function. That chapter starts with explaining a general architecture of the proposed second type adaptive control, and a description of two types of the evaluation function used. The proposed second type adaptive control designs using the two types of the evaluation functions are then described in detail, followed by experimetal results of their applications to the cart-pole balancing problem. Chapter 4 ends with the discussion of the experimental results, comparisons between all the adaptive control developed in the dissertation and other past research. Finally, in Chapter 5 the author concludes the dissertation.

CHAPTER 2

Adaptive Control and Reinforcement Learning

This chapter reviews the fields underlying the discussion throughout this dissertation. Several important achievements on the traditional adaptive control as well as their limitations are identified. The notion of optimality and ways of learning optimal policy, which depend on the concept of value attributed to state and state-action, is then discussed. The discussion is focused on an overview of policy search methods of reinforcement learning in general and related work on gradient methods in policy search in particular.

2.1 Plant Knowledge Issue

A good understanding of a plant to be controlled is required by most advanced control methods where the plant dynamics is represented using Laplace transfer functions or differential equations. In many control problems, however, the plant dynamics may be too complex or its physical process is not well understood. Quantitative knowledge of the plant is then not available. This is referred to as a "black box" problem. In other control problems, some knowledge of the plant is available but we are not sure about its accuracy. This is referred to as a "gray box" problem. The last case is when quantitative knowledge of the plant is available. It is often a relatively simple task to design a controller for the plant in this case because we can use well-established control methods and tools. This is referred to as a "white box" problem [6].

Most learning methods, including RL, deal with the "black box" problem. In this chapter, discussion regarding the learning methods is focused on RL to obtain an understanding how importance the philosophy behind RL to be introduced into the adaptive control designs.

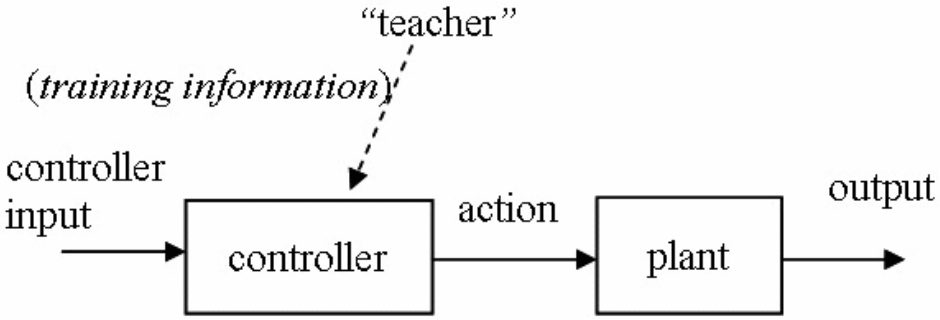


Figure 2.1: Controller training problem. The controller needs to be trained by a good "teacher" that supplies high quality training information to the controller.

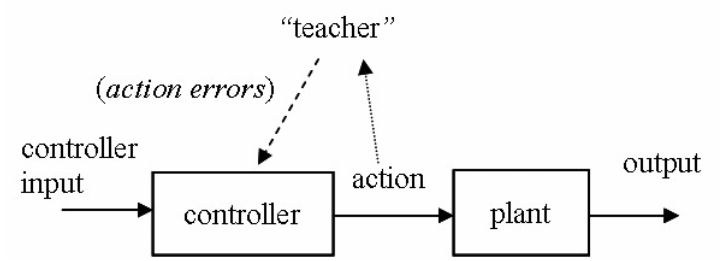
On the other hand, most adaptive control methods deal with the "gray box" problems, each of which has its own defects, depending on what assumption about "gray knowledge" of the plant is made to enable the adaptive control method to work. Adaptive control problems discussed in the dissertation fall into the "gray box" problem. They are discussed in detail in Chapter 3 and 4. This chapter discusses existing adaptive control methods as the background for the proposed adaptive control discussed in both next chapters. However, in general, when designing any controller it is an essential requirement to know in advance what kind of information will be available to the controller. Therefore, the following sections start with a discussion regarding types of the information available in various kinds of control problems and its affects to controller design and performance.

2.2 Training Information

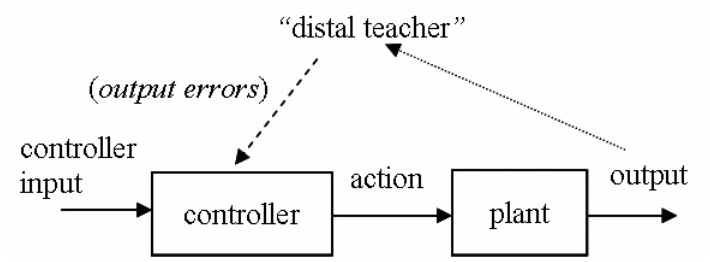
Every adaptive controller is required to manipulate plant input so that the plant behavior meets specified control objective. To accomplish such a task, the controller needs to be trained by a good "teacher" that supplies high quality information (see Fig.[1] 2.1). Finding the good teacher is a difficult task. On the other hand, the quality degree,

[1]Whether the controller training problem is open-loop or closed loop, it depends on the definition of the controller input, which can be separated from the definition of the "teacher". Hence, to avoid unnecessary details, the controller training problem is described with an incomplete diagram in

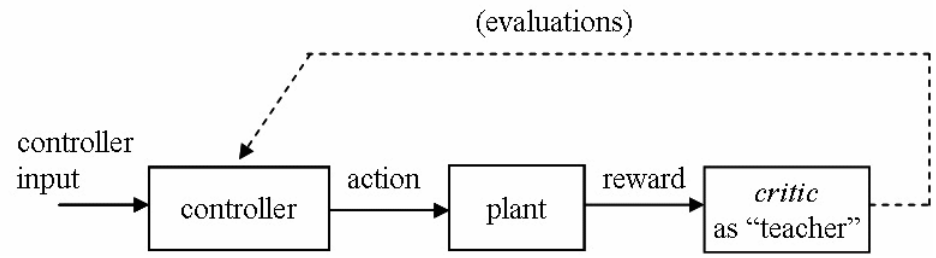
to which the available training information is informative about control performance, critically influences the controller's training [1, 32].



(a) Supervised Learning



(b) Distal Learning



(c) Reinforcement Learning

Figure 2.2: "Teacher" in controller training with (a) Supervised Learning (b) Distal Learning (c) Reinforcement Learning

By several researchers [1, 10, 21, 33–36], the quality of the training information supplied by the teacher is used as the criterion to classify approaches for training the controller, namely, two of them are *supervised learning* and *reinforcement learning* (RL) [1]. In the supervised learning, the teacher provides the controller for each given plant state with target actions (see Figure 2.2(a)). When executed at the given plant states, those target actions need to be assured to lead the desired behavior of the plant

Figure 2.1, i.e., of an "open-loop", but it should be a closed-loop in the complete diagram. Similar simplified descriptions are also used in the explanations that follow.

in terms of control performances. The supervised learning involves "memorizing" those target actions by minimizing the discrepancies between actual and the target actions (i.e., the action errors). Thus, the main task of the teacher in supervised learning is to tell the controller the right actions to improve the control performances. That is, the teacher's role is *instructional*.

In contrast, the teacher's role in RL is more *evaluative* rather than instructional [1, 21]. Because of such a role, the teacher in RL is sometimes called a *critic*. As can be seen in Figure 2.2(c), based on *reward* for the action the teacher or the critic in RL usually provides the controller simply with a scalar evaluation, leaving it to the controller to discover appropriate actions in order to obtain better evaluations in the future. The critic is required to provide the scalar evaluation that indicates how good the action taken is, not whether the action taken is the best or the worst one. But, it does not necessarily explicitly know the appropriate controller function, rather it needs to know the characteristics of the desired control behavior regardless of whatever type of the controller used.

Other researchers, Jordan and Rumelhart [37], define an intermediate approach for training the controller that shares aspects of both supervised learning and RL, i.e., *learning with a distal teacher* or simply the author calls it *a distal learning* (see Figure 2.2(b)). The term "distal" means as "far from the point of origin". And, the term "a distal teacher" means the teacher that provides the training information pertaining to distal information (i.e., the plant output errors), instead of to the actions [1]. The actions are *proximal* variables; that is, the variables that the controller controls directly, while the plant output errors are *distal* variables, the variables that the controller controls indirectly through the intermediary of the proximal variables [37]. The distal learning assumes target values are available for the distal variables but not for the proximal variables. This is exactly the case for most control designs, including adaptive control design, because many control problems are formulated either as the regulation problem, in which a plant output has to stay at the setpoint, or as the tracking problem, in which the plant output has to follow the reference trajectory.

Despite using different qualities of the training information, the tasks in both supervised and distal learning are considered as similar in [1]. Viewing the plant and the critic in RL as a composite system whose "output" is the evaluation provided by the critic, Gullapalli [1] also regards the tasks in RL as special cases of the tasks in the distal learning. With this perspective, the distal "error" in the RL tasks is then assumed to be any negative value if the critic's evaluation is to be minimized and any positive value if the critic's evaluation is to be maximized.

2.3 Adaptive Control

Most of control methods have been well developed for solving certain well-defined classes of control problems involving linear processes and certain types of cost functions. The controller in those control techniques are constructed based on mathematical model of the plant to be controlled. Common assumption made is that the plant model accurately replicates all relevant aspects of the plant behavior, and the controller designed using those techniques should be able to perform well with the real plant. Consequently, the controller performance critically depends on the accuracy of the plant model. While there are no similar techniques developed for general classes of nonlinear control problems despite substantial efforts, in most control problems it is difficult to model the plant precisely. Incorporating adaptive methods in constructing the controller has been one alternative to solve those control problems.

The adaptive control methods typically assume a plant model structure. The basic idea behind the adaptive control is to estimate on-line the uncertain or unknown plant parameters (or equivalently, the corresponding controller parameters), and use those estimated parameters to compute the control action [3]. Therefore, an adaptive control system can be considered as a control system with on-line parameter estimation. Based on characteristic of the plant parameters to be estimated, it is a customary to separate the parameter estimation problems as the tuning problem and the adaptation problem. When the true parameters of the plant to be controlled are constant, the parameter estimation problem is the tuning problem, otherwise the adaptation problem. Although in practise many control problems include the unknown plants whose paramaters are time-varying, most of the adaptive control methods are developed for solving the tuning problem. This is to avoid mathematical difficulties. Consequently, for those adaptive methods to be applicable in the practise, the time-varying plant parameters must vary considerably slower than the parameter adaptation.

In the following, the author discusses two main approaches widely used for constructing and training the adaptive controller, i.e., model-reference adaptive control method and self-tuning method. The discussion of the former method is accompanied by the approaches commonly used for determining adaptation rules for adjusting the controller parameters, i.e., the gradient method and Lyapunov method. Examples of their applications are also provided to show the difficulties of their implementation in the real practices.

2.3.1 Model-Reference Adaptive Control (MRAC)

MRAC was originally developed to solve a control problem in which the specifications of desired closed loop behavior are given in terms of a *reference model*. A general configuration of MRAC can be represented by Figure 2.3. MRAC is composed of four parts: a *plant* whose parameters are unknown, the reference model that generates the desired plant output, a *feedback controller* containing adjustable parameters, and an *adaptation rule* for updating the adjustable parameters of the controllers.

The *plant* in MRAC is typically assumed to have a known structure of the plant, but its parameters are unknown. For linear plants, "known structure" means that the number of "poles" and "zeros" of the linear differential equation of the plant are assumed to be known, but the locations of the poles and the zeros are unknown. For nonlinear plants, it is equivalent to a known structure of the dynamics equation, but its parameters are not known.

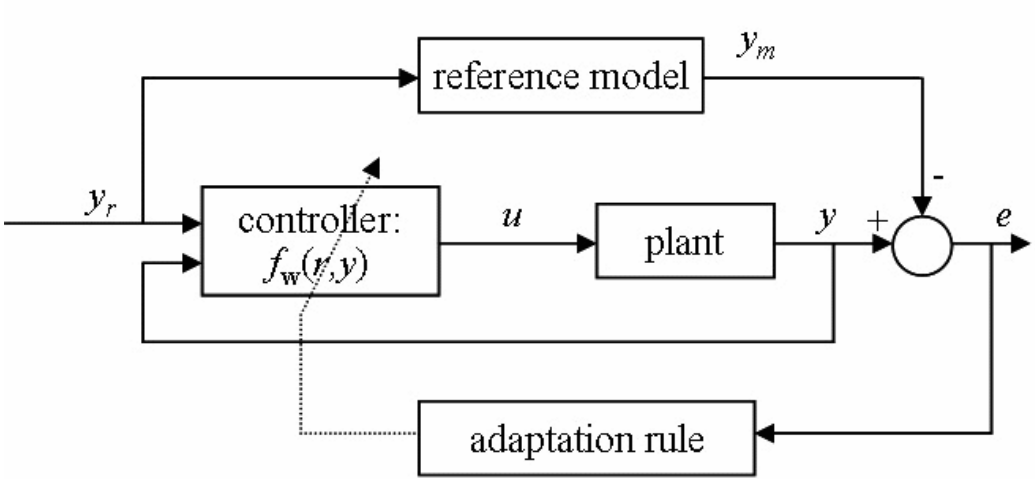


Figure 2.3: Model Reference Adaptive Control. y_m: model reference output, y_r: reference or command signal, y: plant output, $u = f_{\mathbf{w}}(y_r, y)$: plant input, $\mathbf{w}$: adjustable controller parameter vector, e: plant output error.

The *reference model* is used to specify how the plant outputs should ideally respond to the desired outputs. In other words, the model is used to generate a desired trajectory the plant output should track as closely as possible. The choice of the reference model is an important part of the MRAC design, implying that the reference model can be choosen arbitrarily. The choice of the reference model has to fullfil two requirements. It should reflect the desired control performance specifications in terms of such as time constant or rise time, settling time, overshoot, or frequency domain characteristics. On

the other hand, those desired performance specifications must be achievable, implying there must be some inherent constraints on the the reference model structure, i.e., in terms of order and relative degree of the model with respect to the assumed structure of the plant model.

The controller is of adjustable parameters. In general, MRAC requires the controller to have perfect tracking capacity so that tracking convergence is possible. That is, when the exactly known parameters of the plant are available and used to construct the controller, the controller should make the plant output identical to the output of the reference model. The controller is said to be linearly parameterized if it is constructed linear in terms of the adjustable paramaters. Most adaptive control designs use linearly parameterized controller, i.e., the linear state feedback controller.

The *adaptation rule* is required to adjust the controller parameters so that the plant output is driven to the reference model output as closely as possible. That is, the objective of the adaptation is to make the tracking error converge to zero. There are many formalisms that can be used to develop the adaptation rules in MRAC. Two of them are discussed here, i.e., the gradient method and the Lyapunov method.

2.3.1.1 Gradient Method

The gradient method is a fundamental idea in MRAC. The basic idea behind the gradient method is as follows. Assume that we attempt to adjust the controller parameters so that the errors between the outputs of the plant and the reference model converge to zero. Let us use e to denote the errors and $\mathbf{w}$ to denote the controller parameter vector to be adjusted. Then, define the criterion

$$J(\mathbf{w}) = \frac{1}{2}e^2, \tag{2.1}$$

assuming that smaller $J(\mathbf{w})$ implies better control performance of MRAC, and therefore, $\mathbf{w}$ are considered appropriate. To make $J(\mathbf{w})$ small, the parameter vector $\mathbf{w}$ is changed in the direction of the *negative* gradient of J, i.e.,

$$\dot{w} = -\gamma\frac{\partial J}{\partial w} = -\gamma e\frac{\partial e}{\partial w}, \tag{2.2}$$

where w is an element of the parameter vector $\mathbf{w} = \{w\}$ and γ denotes an adaptation rate. The derivative $\partial e/\partial w$ is called *sensitivity* derivative of the system. It can be evaluated under the assumption that the true parameters w are constant or changing much more slowly than other parameters in the system. The update rule of (2.2), where

$\partial e/\partial w$ is the sensitivity derivative, is commonly referred to as *MIT rule*. This name is chosen because the MIT rule is the result of the work done at the Instrumentation Laboratory in Massachusetts Institute of Technology (MIT) [2].

In the MRAC designs it is common to represent the criterion in term of the error e. The first MRAC was implemented to minimize the criterion

$$J(\mathbf{w}) = |e|$$

and the update rule becomes

$$\dot{w} = -\gamma \frac{\partial e}{\partial w} \operatorname{sign}(e).$$

Moreover, the gradient update rule is sometimes implemented in even simpler form

$$\dot{w} = -\gamma \operatorname{sign}\Big(\frac{\partial e}{\partial w}\Big) \operatorname{sign}(e),$$

which is called the *sign-sign* algorithm.

For clarity, the application of the gradient method to the MRAC designs are illustrated in the following examples (see [2] for more examples).

Example 2.1:
The MRAC design developed in this example is exactly what Figure 2.3 describes. Assume that the plant dynamics is of a model structure

$$\dot{y} = -ay + bu, \tag{2.3}$$

where u is a control variable, y the plant output, a and b are constants. Suppose that the plant is desired to behave according to the reference model:

$$\dot{y}_m = -a_m y_m + b_m y_r, \tag{2.4}$$

where y_r is the reference signal. The reference model output can be represented as

$$y_m = \frac{b_m}{p + a_m}\, y_r,$$

where $p = d/dt$ is the differential operator. There is a caveat in choosing the design paramater a_m. If $a_m > a$, dynamics of the desired model is "faster" than that of the plant, and vice versa. This implies that a_m cannot be chosen arbitrarily, i.e., its choice needs some a priori knowledge of the plant.

Suppose that the controller can be determined as

$$u = cy_r - dy,$$

where

$$c = \frac{b_m}{b},\ d = \frac{a_m - a}{b}$$

are the controller parameters. Using such a controller, the plant output corresponds to

$$y = \frac{bc}{p + a + bd}\, y_r.$$

The control design problem is to make the plant output y "follow" the reference model output y_m. This problem is solved by adjusting c and d so that $(a + bd) \to a_m$ and $bc \to b_m$.

Define the error

$$e = y - y_m.$$

The sensitivity derivatives are obtained by taking the partial derivatives of the error with respect to c and d, i.e.,

$$\begin{aligned} \frac{\partial e}{\partial c} &= \frac{b}{p + a + bd}\, y_r \\ \frac{\partial e}{\partial d} &= -\frac{b^2 c}{(p + a + bd)^2}\, y_r = -\frac{b}{p + a + bd}\, y. \end{aligned} \tag{2.5}$$

Hence, the determination of the adaptation rules for adjusting c and d using the gradient approach seems straightforward. However, those sensitivity derivatives in (2.5) cannot actually be used to determine the adaptation rules for adjusting c and d because they require a and b to be known. Approximations are therefore required to obtain realizable parameter adaptation rules (see [2] for the heuristic approximations that may be used).

The above simple example applies only to the linear plants. It shows that the gradient method can be used to implement the adaptation rules for MRAC. However, solving adaptation rules using the gradient approach is a hard task, despite under the assumption that a linear model structure of the plant is known. Adaptive control theorists have devoted considerable attention to determining the adaptation rules using some approximations based on heuristic approaches.

Despite its limitations, the gradient method based on the MIT rule has been claimed to be applicable to optimization of more general criterion function than the criterion of (2.1). In the MRAC designs, it is implemented using a straightforward principle. Specify first a model and a controller with adjustable parameters, and the control performance is given as a criterion function defined a priori. The adaptation rules for updating the controller parameters are obtained by computing the gradient of the

criterion function with respect to the parameters and making the changes of the parameters in the negative gradient of the criterion function. Such an approach has the same problems as the MIT rule, i.e., the plant model parameters need to be known. To make this unrealistic approach realizable, some approximations have to be made as well.

2.3.1.2 Lyapunov Method

Many well-developed approaches are available for the stability analysis of linear, time-invariant control systems, such as Nyquist stability method and Routh's stability method. However, it is extremely difficult or impossible to perform stability analysis for nonlinear systems and/or time-varying systems. Lyapunov introduced a direct method (a.k.a. Lypunov's second method) to investigate the stability of a solution to continuous-time nonlinear systems [2, 3, 38]. The author simply uses the term "Lyapunov method" in this dissertation to refer to the Lyapunov's second method.

From a point of view of the classical theory of mechanics, it is intuitively understandable that a vibratory system is stable if its total energy (in the form of a *positive* function) continually decreases (which means that the energy function must have negative time derivative) until an equilibrium state is reached. Based on this fact, Lyapunov makes a generalization that if the system has an asymtotically stable equilibrium, then the stored energy of the system decays as time increases until it finally reaches its minimum value at the equilibrium state. However, there is no straighforward way of defining an "energy function". To overcome this difficulty, a fictitious energy function called a *Lyapunov function* is introduced by Lyapunov. This idea is more general than that of the energy function and more widely applicable where Lyapunov function is not necessarily unique. Any scalar function satisfying the hypotheses of Lyapunov's stability theorem (to be given later) can serve as Lyapunov functions. For simple systems, it is sometimes easy to guess or approximate suitable Lyapunov functions. But, this is not true for complicated systems.

Lyapunov function depends on the state variables $x_1, x_2, \cdots, x_n$, and time t. It is denoted by $V(x_1, x_2, \cdots, x_n, t)$ or simply $V(\mathbf{x}, t)$ or $V(\mathbf{x})$ (i.e., when Lyapunov function does not include t explicitly). The Lyapunov method focuses on dealing with the sign behavior of V and that of its derivatives. Those signs give us information regarding the stability or asymtotic stability without requiring us to solve the stability problems directly based on the system dynamics, either linear or nonlinear. The formalism of the Lyapunov method is given below.

Suppose that a system is described by

$$\dot{\mathbf{x}} = f(\mathbf{x}, t) \tag{2.6}$$

where $\mathbf{x}$ denote the system state and

$$f(\mathbf{0}, t) = \mathbf{0}, \text{ for all } t.$$

For the above system, define a Lyapunov function candidate $V(\mathbf{x})$ (in the form of a scalar function) with an equilibrium point at $\mathbf{x} = \bar{\mathbf{x}}$ that satisfies the conditions

1. $V(\mathbf{x})$ is continuous and differentiable everywhere in state space,
2. $V(\mathbf{x}) > V(\bar{\mathbf{x}})$ for all $\mathbf{x}$ in state spape and $\mathbf{x} \neq \bar{\mathbf{x}}$.

The Lyapunov function candidate, V, is a Lyapunov function if

$$\dot{V}(\mathbf{x}) \leq 0 \text{ for all } \mathbf{x} \text{ in state space.} \tag{2.7}$$

This implies that the Lyapunov function is a "bowl"-shaped funtion with an unique equilibrium point of state. It has trajectories that never move up the surface of the bowl, rather always move downward on the bowl surface. The above definition of the Lyapunov function facilitates the following Lyapunov's stability theorem [2, 3, 38–40]:

Theorem 1 *If V is a Lyapunov function that exists for the system (2.6), then the system is stable. When $\dot{V}(\boldsymbol{x}) < 0$, then the system is asymtotically stable.*

Although the Lyapunov method is very useful and powerful for dealing with stability problems of nonlinear systems, there is no straighforward method or step-by-step method to find suitable Lyapunov function, leaving the stability analysis of many nonlinear systems remains as a hard task. Considerable ingenuity and experience prove to be very important in determining the stability of many nonlinear systems using the Lyapunov method. And in general, none of adaptive control based on analytically derived fixed Liapunov functions can be regarded as universally applicable. In summary, the Lyapunov method is only an alternative way to investigate the system stability. Nevertheless, it might be able to answer the question regarding the stability of nonlinear systems when other methods fail.

The following example illustrates the applications of the Lyapunov method to the MRAC design for a linear plant. It uses the same case as in the previous example.

Example 2.2:
Consider the same problem as in Example 2.1. The task now is to determine the adaptation rules for updating c and d based on the Lyapunov method. Introduce the error:

$$e = y - y_m.$$

Subtituting the assumption model of (2.3) and the reference model of (2.4) into time derivative of e, we obtain

$$\dot{e} = -a_m e + (a_m - a - bd)y + (bd - b_m)y_r.$$

When a and b are known, we can set the best controller parameters as $c = b/b_m$ and $d = (a_m - a)/b$ (i.e., equivalent to $a_m = a + bd$ and $b_m = bc$) so that we obtain

$$\dot{e} = -a_m e,$$

i.e., the error goes to zero. Actually, this implies existence of a Lyapunov function of the form

$$V(e) = \frac{1}{2}e^2.$$

The proof is trivial. Taking time derivative of $V(e)$, we obtain

$$\dot{V}(e) = e\dot{e} = -a_m e^2 < 0,$$

that means that e goes to zero asymtotically.

Because a and b are not known, the controller parameters $c = b/b_m$ and $d = (a_m - a)/b$ are not realizable. The problem in the MRAC design is to adjust c and d (equivalent to adjusting a and b) so that e goes to zero. At the same time, it is desired that $(a + bd) \to a_m$ and $bc \to b_m$, i.e., the plant is desired to "follow" the reference model. To solve this problem, first introduce a Lyapunov function candidate:

$$V(e, c, d) = \frac{1}{2}\Big(e^2 + \frac{1}{b\gamma}(bd + a - a_m)^2 + \frac{1}{b\gamma}(bc - b_m)^2\Big).$$

Taking time derivative of $V(e, c, d)$, we obtain

$$\begin{aligned}
\dot{V} = &\quad e\dot{e} + \frac{1}{\gamma}(bd + a - a_m)\dot{d} + \frac{1}{\gamma}(bc - bm)\dot{c} \\
= &\quad -a_m e^2 + \frac{1}{\gamma}(bd + a - a_m)(\dot{d} - \gamma y e) \\
&\quad + \frac{1}{\gamma}(bc - b_m)(\dot{c} + \gamma y_r e).
\end{aligned}$$

If c and d are adjusted according to the following rules

$$\begin{aligned} \dot{c} = & \quad -\gamma y_r e, \\ \dot{d} = & \quad \gamma y e, \end{aligned}$$

we obtain

$$\dot{V} = -a_m e^2 < 0,$$

which implies that $V \to 0$ as $t \to \infty$, equivalent to $e \to 0$, $(a + bd) \to a_m$ and $bc \to b_m$.

The above example shows that the Lyapunov method can be used to implement the adaptation rules for the MRAC design for the simple linear plant. As in the previous example, however, the above MRAC design method is limited by the need to assume a known structure of the linear plant model. In addition, even if the assumption plant model used is simple, the above MRAC design method involves defining quite complicated Lyapunov function candidate from which the adaptation rules for updating the controller parameters need to be found. From a practical point of view, the problem of finding suitable Lyapunov function analytically becomes more and more difficult as one tries to control more and more complex nonlinear systems.

2.3.2 Self-Tuning Control (STC)

Using known model plant parameters, non-adaptive control methods determine a set of desired controller parameters associated with desired control performance. In adaptive control, controller parameters adjusted all the time are desired to converge to that set of the desired controller parameters even when the plant model is unknown. An adaptive control with this property is called *self-tuning control* (STC).

Under the assumption that a model structure of the plant is known but its paramaters are unknown, the basic idea behind STC is to separate the estimation of the unknown plant model parameters from the design of the controller (see Figure 2.4). The unknown parameters are estimated on-line using recursive estimation methods. Although being estimated, those parameters are treated as if they are true. This idea is referred to as *certainty equivalence principe*. The estimation of the parameters can be considered simply as the problem of finding a set of parameters that *fits* input-output data from the plant. This is different from parameter adaptation in MRAC where the parameters are adjusted so that the error between the output of the plant and the reference model converges to zero. Many methods are available for estimating the parameters of linear plant models. The most popular estimation method is the least squares method and its extensions.

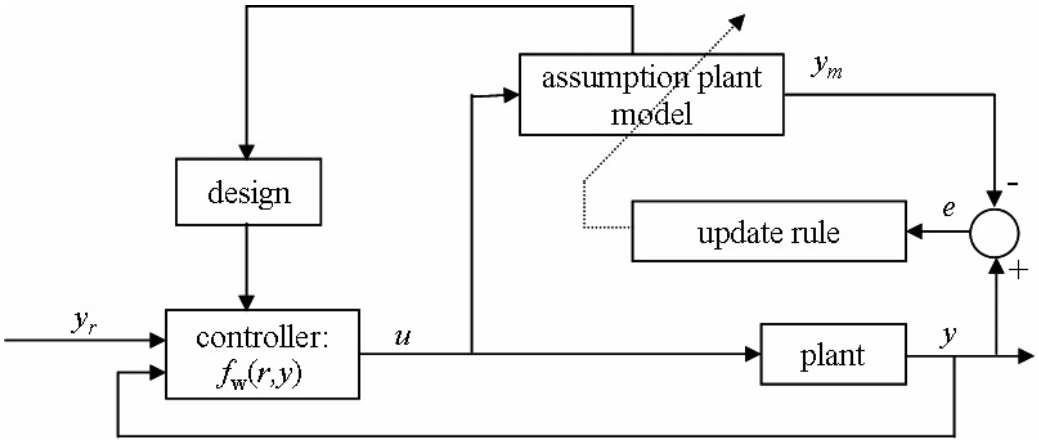


Figure 2.4: Self-tuning controller whose parameters are determined using online estimated plant parameters.

The desired specifications of the closed loop system in STC are determined based on the online estimated plant model parameters. This is done in the block "design" in Figure 2.4. The choice of the control design method used in the block "design" depends on the desired specifications of the closed loop system. Many control design methods are available for linear plants, such as pole-placement, PID, LQR (Linear Quadraric Control), and minimum variance control.

In the STC designs, determining the controller parameters from the estimated plant model parameters is called *indirect* method. In contrast, it is possible to eliminate the need of translating the estimated plant model parameters into the control parameters. Note that the parameters of the controller and the plant are related to each other for a specific control method. This means that the plant model may be reparameterized using the controller parameters (which are also unknown, of course). After reparameterization of the plant model, standard techniques for estimating the plant model parameters are used. This approach is referred to as *direct* method since there is no need of translating the estimated plant model parameters into the control parameters. The similar way can be used to consider the indirect method and the direct method in MRAC [3].

2.3.3 Comparing MRAC and STC

The above discussions describe how MRAC and STC are developed from different perspectives. While in the MRAC designs the desired specifications of the closed loop

system are included in the definition of the reference model, in STC the desired specifications of the closed loop system are determined based on the online estimated plant model parameters. To achieve the desired specifications, the parameters in MRAC systems are updated so as to minimize the error between the plant output and the reference model output, while in STC systems updated so as to fit the input-output measurement data from the plant. This implies that the STC systems are more heavily dependent on appropriateness of the assumption plant model than the MRAC systems. Consequently, the controller performance of the STC systems critically depends on the accuracy of the estimated plant model. In contrast, this is not neccessarily the case for the MRAC systems because they only focus on minimizing the plant output error, not fitting the assumption plant model with the actual plant. Despite their differences, both MRAC and STC are related each other. As can be seen from Figures 2.3 and 2.4, they have two loops, one for control and another for parameter estimation. According to a theoretical viewpoint, MRAC and STC controller can be placed under a unified framework [2,3]. Finally, certain similarity of them is that they are limited by the need of assuming a known *fixed* structure of the plant dynamics model.

2.4 Reinforcement Learning

The aforementioned adaptive control designs (and, control designs in general) assume that the plant output errors (i.e., between the actual plant output and the model output) are *direct* indications for control performances, i.e, smaller plant output error is direct indication for better control performance. Under this assumption, the control objective is then defined simply as to minimize the plant output errors all times. There is no need to try a variety of other actions that might be better to take once an action is predicted to result in the smallest plant output error at the next time step. As discussed in Chapter 1, however, executing the actions that are predicted to result in smaller plant output errors only might mislead to non-goal states. This is because the plant output error is purely *instructive* feedback. It only says about whether the action was correct or incorrect, that is, whether it was a best action or not, but it does not tell *how good* the action was. The notion of "how good" here refers to future plant output errors that can be expected due to executing the action. In other words, the plant output errors do not always imply appropriate instantaneous control performance measures. Reinforcement learning (RL) has been an alternative way of dealing with this issue.

As discussed previously, the most important feature distinguishing RL from other types of learning as well as adaptive control is that it uses *evaluative* feedback that evaluates the actions taken rather than instructs by giving correct actions (or, target plant outputs). Evaluative feedback indicates how good the action taken is, but it does not tell whether the action is the best or the worst one. Using evaluative feedback, RL does not necessarily take the actions that minimize the plant output errors in the short run, because it can predict and take the actions that will result in much smaller plant output errors in the long run. Brief explanation of RL is given as follows.

2.4.1 Basic Structure

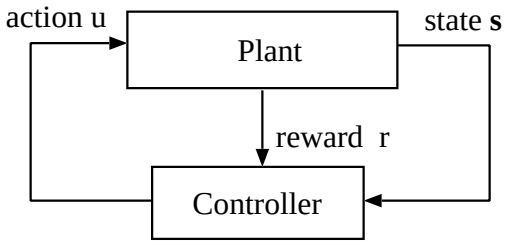


Figure 2.5: Basic structure of reinforcement learning control problem. In the theory of reinforcement learning, the controller corresponds to an agent and the plantcorresponds to an environment.

A standard model of reinforcement learning is depicted in Figure 2.5. The author represents a basic structure of RL for solving control problems using two components: a *controller* and a *plant*. In the theory of RL, the controller corresponds to an agent and the plant corresponds to an environment. Note that the "controller" in Figure 2.5 includes the critic. Therefore, it not only produces actions but also evaluates the actions it executes. The "plant" not only produces outputs but also rewards for the actions applied to it.

At a discrete time step t, given a state $\mathbf{s}_t$, the controller applies an action u_t to the plant and receives a *reward* r_{t+1} in the next time step $t+1$. Let $0 < \gamma < 1$ be a discount factor against a future reward. The goal of the controller is to find an optimal *policy* that determines a sequence of actions $\vec{u}_t = (u_t, u_{t+1}, u_{t+2}, ...)$ maximizing the total amount of the discounted reward received in the long run:

$$V(\mathbf{s}_t) = \max_{\vec{u}_t} \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \tag{2.8}$$

which is referred to as a *value function*.

By its definition in (2.8), the value function tells what is good in the long run, in contrast with the reward that tells what is good in an immediate consequence. That is, the reward represents the immediate, intrinsic desirability of plant states, and the value represents the long-term desirability of states after taking into account the states that are likely to follow, and the rewards available in those states. For example, a state might be always of a low immediate reward but still of a high value if it is regularly followed by other states that yield high rewards. Or, the opposite could be true.

We can say that the rewards are primaries, whereas the values, as predictions of the rewards, are secondaries. There could be no values without rewards, and the only purpose of estimating the values is to achieve more reward. Nevertheless, the controller are most concerned only with the values when deciding and evaluating the actions. Actions must be chosen so that the states are of highest values, *not* highest rewards, because these actions result in the greatest amount of rewards over the long run [7].

There are two distinct ways of finding the optimal policy based on the value function, i.e., *value-function approach* and *policy-search approach*, that are explained in the following.

2.4.2 Value-Funtion Approach

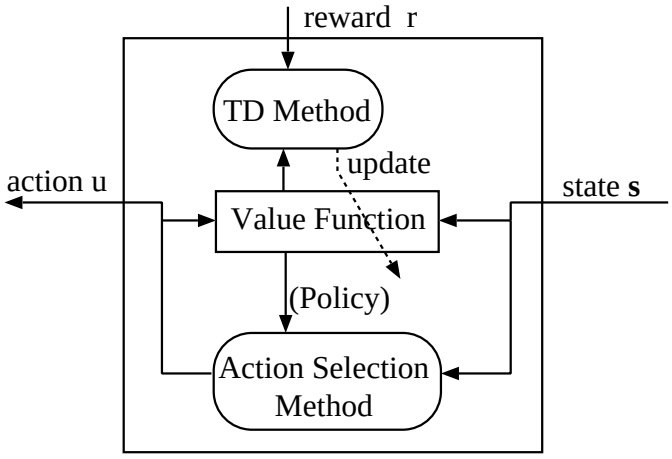


Figure 2.6: Value function approach.

The equation (2.8) defines a *state-value function* measuring the maximum possible sum of rewards the controller could receive when it starts from $\mathbf{s}_t$ and performs a sequence of actions $\vec{u}_t$. This function is the solution of the following Bellman equation:

$$V(\mathbf{s}_t) = \max_{u_t} \left[r_{t+1} + \gamma V(\upsilon(\mathbf{s}_t, u_t)) \right], \tag{2.9}$$

where $\upsilon(\mathbf{s}_t, u_t) = \mathbf{s}_{t+1}$ is a *plant dynamics function*.

From (2.9), one can deduce the optimal policy:

$$u^*(\mathbf{s}_t) = \arg\max_{u_t} \left[r_{t+1} + \gamma V(\upsilon(\mathbf{s}_t, u_t)) \right]. \tag{2.10}$$

When the controller is not given the plant dynamics $\upsilon(\mathbf{s}_t, u_t)$, the equation (2.10) is useless. Alternatively, the controller can use a *state-action-value function* (a.k.a. *Q-function*):

$$Q(\mathbf{s}_t, u_t) = r_{t+1} + \gamma Q(\mathbf{s}_{t+1}, u^*(\mathbf{s}_{t+1})), \tag{2.11}$$

where $u^*(\mathbf{s}_{t+1})$ is the optimal policy deduced by

$$u^*(\mathbf{s}_{t+1}) = \arg\max_{u_{t+1}} Q(\mathbf{s}_{t+1}, u_{t+1}). \tag{2.12}$$

Initially, the optimal policy is not available when the Q-function is not learned yet. As the controller interacts with the plant, the Q-function is updated using the *Temporal Difference (TD) method* [41]. To improve the Q-values during learning, the action cannot always be picked up from the current optimal policy. Random actions of a small fraction, ϵ, of the time must also be chosen for exploration. Such action selection method is known as an ϵ-greedy action selection. Figure 2.6 describes how all these processes interact in the controller.

Using the TD method, an update formula for the Q-function is as follows:

$$Q(\mathbf{s}_t, u_t) \leftarrow Q(\mathbf{s}_t, u_t) + \alpha \delta_t, \tag{2.13}$$

where α is a *learning rate*, and δ_t is a *TD error*:

$$\delta_t = r_{t+1} + \gamma Q(\mathbf{s}_{t+1}, u^*(\mathbf{s}_{t+1})) - Q(\mathbf{s}_t, u_t). \tag{2.14}$$

The above formulas only hold for a discrete representation of states and actions. The Q-function can be simply represented by a look-up table that maps a state-action pair

to its value. If states and actions are represented continuously, a function approximator such as a neural network must be used as the Q-function. In such a case, the TD method is used to update the weights (not a Q-value) of the Q-function approximator.

Let Q_Ω be a function approximator of the Q-function with a weight vector $\Omega = \{\omega\}$. The TD method updates ω of the Q-function Q_Ω as follows:

$$\begin{aligned} \omega_t &\leftarrow \omega_t + \eta \delta_t \mathbf{G}_t \\ \mathbf{G}_t &\leftarrow \lambda\gamma \mathbf{G}_t + \frac{\partial Q_{\Omega_t}(\mathbf{s}_t, u_t)}{\partial \omega_t}, \end{aligned} \tag{2.15}$$

where η is a learning rate and $0 \leq \lambda \leq 1$.

2.4.3 Policy Search Approach

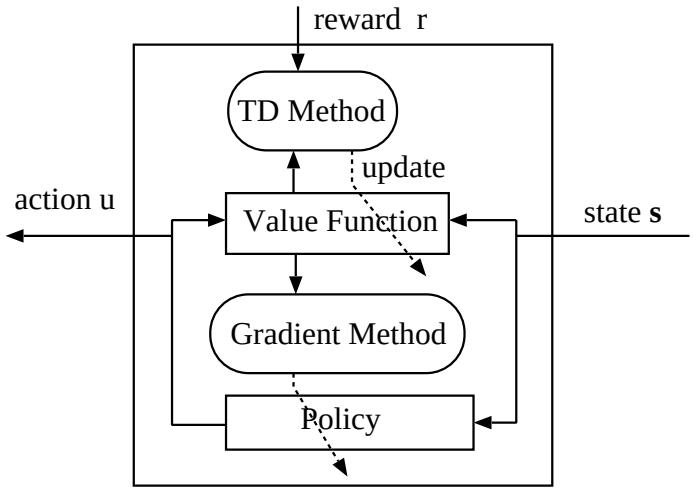


Figure 2.7: Policy search approach.

Figure 2.7 describes how the controller works based on the policy search approach. In the value function approach, the action-selection policy is implicitly represented by the estimated value function. But, in the policy search approach, the policy is directly approximated with its own parameters and can be represented by any function approximator such as a neural network or a fuzzy system, whose input is a state of the plant and whose output is an action to the plant.

Like in the value function approach, the controller using the policy search approach must first estimate the value function. But, the value function is not directly used to

decide an action. Instead, it is used as an evaluation function for tuning the parameters of the policy.

Let $u = \pi_{\mathbf{w}}(\mathbf{s})$ be a policy with a parameter vector $\mathbf{w} = \{w\}$ and ρ be the performance measure that may be represented as a function of the TD-error (i.e., $\rho(\delta)$) or the Q-function (i.e., $\rho(Q)$). Using $\rho(\delta)$ as the evaluation function to be minimized, the policy search approach updates w by a gradient descent method as follows:

$$w_t \leftarrow w_t - \eta \frac{\partial \rho(\delta_t)}{\partial w_t} \tag{2.16}$$

(see [20] for its details and modified versions).

Similarly, using $\rho(Q)$ as the evaluation function to be maximized, the policy search approach updates w by a gradient ascent method as follows:

$$w_t \leftarrow w_t + \eta \frac{\partial \rho(Q(\mathbf{s}_t, u_t))}{\partial w_t} \tag{2.17}$$

(see [14] for an example of the exact solution of $\partial \rho(Q(\mathbf{s}_t, u_t))/\partial w_t$).

Alternatively, using $\rho(Q)$ as the evaluation function under the assumptions that both $\partial \rho(Q)/\partial u$ and $\partial u/\partial w$ exist, the policy search approach can update w as follows:

$$w_t \leftarrow w_t + \eta \frac{\partial \rho(Q(\mathbf{s}_t, u_t))}{\partial u_t} \frac{\partial u_t}{\partial w_t}. \tag{2.18}$$

Berenji and Khedkar [10, 11] have proposed an update rule similar to (2.18). They used $V(\mathbf{s}_t)$ as the evaluation function to be maximized to tune the weights of the policy using the following update rule:

$$w_t \leftarrow w_t + \eta \frac{\partial V(\mathbf{s}_t)}{\partial u_t} \frac{\partial u_t}{\partial w_t}. \tag{2.19}$$

Based on the update rule (2.19), the goal of executing action u_t is to maximize $V(\mathbf{s}_t)$.

Since the partial derivative $\partial V(\mathbf{s}_t)/\partial u_t$ does not exist, it is unclear how to compute $\partial V(\mathbf{s}_t)/\partial u_t$ in (2.19). Nevertheless, in [10], under the assumption that $V(\mathbf{s}_t)$ is quite indirectly dependent on u_t, $\partial V(\mathbf{s}_t)/\partial u_t$ is approximated as follows:

$$\frac{\partial V(\mathbf{s}_t)}{\partial u_t} \approx \frac{\triangle V(\mathbf{s}_t)}{\triangle u_t} = \frac{V(\mathbf{s}_t) - V(\mathbf{s}_{t-1})}{u_t - u_{t-1}}. \tag{2.20}$$

It seems that the update rule (2.19) is not efficient to improve the weights of the policy. In (2.19), while the accuracy of $V(\mathbf{s}_t)$ is not guaranteed during the learning

process, the computation of $\partial V(\mathbf{s}_t)/\partial u_t$ requires accurate value of $V(\mathbf{s}_t)$. On the other hand, $V(\mathbf{s}_t)$ only represents the value of the state $\mathbf{s}_t$, whatever the action u_t is applied. Hence, there will be situations where we can not assume that $V(\mathbf{s}_t)$ is dependent on u_t either directly or indirectly. When such situations occur, we can not use even the approximation of $\partial V(\mathbf{s}_t)/\partial u_t$.

2.4.4 Defects of Reinforcement Learning

Despite many successful applications of reinforcement learning using the above approaches [8, 10, 11, 22, 23, 36, 42], reinforcement learning has several difficulties when applied to the control problem, which are as follows:

1. The value function is not readily available. To obtain the best approximation of the value function, the controller must do many trial-and-error interactions with the plant.

2. When the value function approach is used, action selection can be very sensitive to an arbitrarily small change in the estimated value function.

3. An optimal value function is only associated with a *fixed* goal state, and in turn the optimal actions derived from the value function are only best applied to achieve the fixed goal state. Consequently, when the goal state is changed, the current optimal value function may be no longer useful for the controller to determine the optimal actions.

2.5 Comparing Adaptive Control and Reinforcement Learning

Adaptive control and RL are distinct in the ways of improving control performances. Adaptive control is not time-consuming due to not relying on the long-term evaluations (rather plant output errors) to improve the control performances. But, its applicability is typically limited due to presupposing a plant model structure that must meet very restrictive assumptions. In contrast, RL does not necessarily presuppose any plant model structure, i.e., it is model-free, and therefore, it is of high applicability. But, it relies entirely on the long-term values of states or states-actions to improve control performances, which leads to low efficiency of RL in terms of time-consuming trials-and-errors required for estimating value function. Thus, the distinct ways of improving control performance make adaptive control and RL distinct in applicability and efficiency.

As discribed above, methods for solving adaptive control problems can be divided into two classes: indirect method and direct method. The former refers to a method in which parameters of the controller are to be computed from those of the plant model estimated online. The latter refers to a method in which parameters of the plant model to be estimated on-line is reparameterized using those of the controller so that there is no need to translate the parameters of the plant model into those of the controller. Thus, both methods are essentially the same.

Somewhat different way can be used for classifying methods for solving control problems involving RL. Basically, the methods for solving control problems involving RL are required to *infer* appropriate control actions from the evaluation function that do not pertain to the target actions or plant outputs directly. To accomplish such a task, a method for bridging the gap between training information that is sufficiently available but less evaluative and training information that is evaluative but not sufficiently available is necessary. Researcher have categorized the methods for bridging this gap into two classes [1]. *Direct* method, i.e., the method that involves actively applying actions chosen stochastically and observing the consequences on the evaluation function of a general form (i.e., independent of the plant model). *Indirect* method, i.e., the method that involves constructing a *model* of the evaluation function in a form that can be readily used to generate requisite information for training the controller. The term "model" in indirect method refers to an evaluation function model that includes the plant model and the critic. Implementation of indirect method consists of two stages: constructing an adequate model of the evaluation function, and using the model to generate the training information required to modify the control behavior. Based on the above way of classifying the methods for solving control problems involving RL, all the aforementioned descriptions of RL refer to direct method because the value function used as the evaluation function is of a general form, i.e, not associated with a certain plant dynamics.

Advantages of direct method over indirect method are straightforward. They do not entail the cost of constructing evaluation function model. Therefore, they are computationally simple and no delay in training the controller. They might be able to perform well even if the plant is non-stationary (i.e., its parameters are changing) as they rely only upon the plant itself to obtain requisite training information for updating the controller. However, in general, a potential disadvantage of direct method is that the evaluation function obtained through active perturbation can be less reliable to represent the stable dynamics of the closed loop system in terms of the long-term control peerformance measures. This implies that the controller updated based on it

can produce the actions that mislead to non-goal states. On the other hand, learning the precise value function requires many time-consuming trials-and-errors. In such situations, the direct method is often slower than the indirect method.

Actually, indirect method is perceived as having advantages over direct method when the evaluation function model is available a priori. An accurate model implicitly provides with substantial information regarding the plant to be controlled. If an adequate model is not known, it may be constructed in various ways, but then it is improved through interactions with the plant.

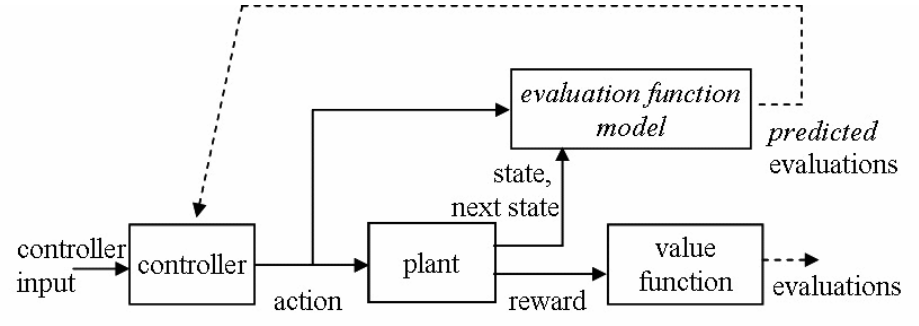


Figure 2.8: Indirect method of proposed adaptive control.

Although it is clear from the above discussion that each method has its own advantages and disadvantages and that no method is universally applicable, the author *mimics* indirect methods in RL in presenting new adaptive control schemes in the dissertation. As mentioned in Chapter 1, the main objective of the dissertation is to present alternative adaptive control by "combining" design philosophies of classical adaptive control and RL. As described in Figure 2.8, the alternative adaptive control involves modifying the controller behavior so that its control performances get improved as measured with an evaluation function model. The author supposes the model in Figure 2.8 as an *approximation* of the actual value function. The model produces its output only based on the state, the action, and the result of executing the action (i.e., the next state), but *not* based on the actual rewards. Implementation of such an approach for designing adaptive control is discussed in Chapter 3 and 4.

CHAPTER 3

Adaptive Control with Fixed Approximate Evaluation Function

The limitations of typical adaptive control and RL discussed the previous chapter motivate the author to seek alternative approach for tuning a controller. This chapter presents an alternative approach for tuning a controller for simple control problems for which we can approximate a value function of the problem state or state-action. The approach is an intermediate result of the author's work.

3.1 Introduction

In reality, many control problems can be modeled as a combination of simple sub-problems. When a problem is relatively simple, a human engineer can make an easy guess about "goodness"' of problem state using its distance to the goal state. And, the action must be close to zero in the steady states near the goal state. For such a problem, instead of using RL, the author assumes that we can use the distance of the problem states to the goal state and actions as an *approximated evaluation function* for tuning the controller. The controller is tuned with the gradient method in its parameter space. Since the value function of the problem states does not need to be learned, this method can be readily applied to a problem with the changing goal.

Of course, the prerequisite about availability of the approximated evaluation function restricts applicability of the proposed method. But, the purpose of this chapter is to show that, for a simple control problem such as a pole-balancing problem, the

controller can be tuned efficiently and smoothly using the conjectured evaluation function and a simple parameter tuning algorithm. Two kinds of controller tuned with that simple parameter tuning algorithm, i.e., a state feedback controller and a fuzzy controller, are discussed in this chapter.

Various kinds of adaptive method for tuning state feedback controller and fuzzy controller have been also proposed in many literatures (see [5, 43–49]). Those methods require the plant model with known structure. The adaptive control method presented in this chapter is a novel model-free approach by which the overall control scheme does not depend on any plant structure (see [50–52]).

3.2 Architecture

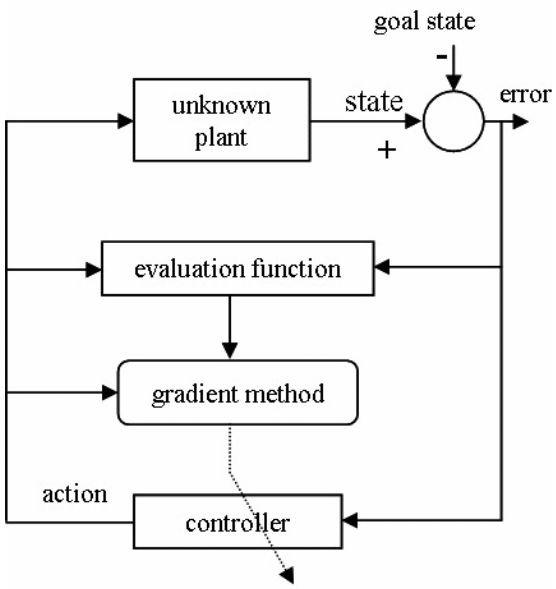


Figure 3.1: Control with Approximated Policy Search Approach

Figure 3.1 describes general architecture of the controller tuning method presented in this chapter. In the dissertation, the method is called CAPS (Controller with Approximated Policy Search). Specific name is used to call the method when the controller is replaced with a certain kind of controller. When the controller is replaced with a *state feedback controller*, the method is called by LCAPS (Linear Controller with Approximated Policy Search). When the controller is replaced with a *fuzzy controller*, the method is called by FCAPS (Fuzzy Controller with Approximated Policy Search).

CAPS tunes parameters of the controller based on the evaluation function. But, it does not use the Q-function as the evaluation function because considerable efforts are required for learning the Q-function even for a simple control problem. Instead, it represents an approximated evaluation function as a function of a Euclidean distance from a goal state and a weighted action, which is assumed readily available and exploitable. In general, the architecture of CAPS is similar to the policy search approach described in Fig. 2.7. The difference is that CAPS is not required to learn the evaluation function.

3.3 Approximate Evaluation Function

Let us use $u = f_{\mathbf{w}}(\mathbf{s})$ to denote either the state feedback controller or the fuzzy controller where $\mathbf{w} = \{w\}$ is a parameter vector. In CAPS, initially all the elements of $\mathbf{w}$ of $f_{\mathbf{w}}$ are set to zero, which makes $f_{\mathbf{w}}$ far from the optimum at the beginning. The gradient method is used to adjust the values of $\mathbf{w}$, and to make $f_{\mathbf{w}}$ the optimal policy by tuning $\mathbf{w}$, a "good" evaluation function is needed.

In many adaptive control designs [2–6], the "good" evaluation function is defined a priori. The common assumption made is that the plant output errors (i.e., between actual and desired plant state) getting smaller are *direct* indications that the actions taken are "correct", i.e., they will lead to the goal state eventually. Otherwise, the actions are to be "blamed". In other words, those adaptive control designs assume a priori that smaller plant output errors imply better instantaneous control performances or immediate rewards, and vice versa.

In general, the author thinks that typical evaluative information expected after executing an action should be at least a function of the action itself, a state at which the action is executed, and the result of executing the action, i.e., the next state. Based on this philosohy, the plant output errors should be considered as *non-evaluative*. And, for this reason, in many control problems executing the actions that are predicted to result in smaller plant output errors only can mislead to non-goal states. Actually, this is exactly the issue addressed in RL. As discussed in Section 2.4, several researchers proposed different forms of the evaluation function good for control using RL. But, in their research, the evaluation functions are based on the learned value function, which the author thinks are unsuitable for the control problems.

For CAPS, rather than using a learned value function (or the Bellman residual) as an evaluation function, the author developes an approximated value function of the form:

$$P(\mathbf{s}(t), u(t)) = \frac{1}{2}(e^2(\mathbf{s}(t + \triangle t)) + \alpha u(t)^2), \tag{3.1}$$

where $e(\mathbf{s}(t+\triangle t))$ denotes a Euclidean distance between a next state $\mathbf{s}(t+\triangle t)$ and a goal state. t denotes continuous time. $\triangle t$ is elapsed time between time steps. α is a weighting factor for an action $u(t)$(i.e., $f_{\mathbf{w}}(\mathbf{s}(t))$) executed at the state $\mathbf{s}(t)$.

In proposing $P(\mathbf{s}(t), u(t))$, the author is motivated by the aforementioned philosophy of the typical evaluative information. The author supposes that $P(\mathbf{s}(t), u(t))$ has a role similar to the state-action-value function $Q(\mathbf{s}(t), u(t))$. Given $\mathbf{s}(t)$, CAPS derives the action $u(t)$ and executes it, and its performance is represented as $P(\mathbf{s}(t), u(t))$. But, CAPS does not know the performance $P(\mathbf{s}(t), u(t))$ before it obtains the result of applying $u(t)$ to the plant, i.e., the next state $\mathbf{s}(t+\triangle t)$. CAPS uses $P(\mathbf{s}(t), u(t))$ as the evaluation function at the next time step $t+\triangle t$ to evaluate the action $u(t)$ executed at $\mathbf{s}(t)$ and then updates the weights of its controller. CAPS assumes that smaller $P(\mathbf{s}(t), u(t))$ implies better performance. And, in the steady states near the goal state the actions $u(t)$ must be close to zero.

Of course, the evaluation function P restricts applicability of CAPS. It is supposed to be useful when the control problem is simple, i.e., the closer state to the goal state implies that the smaller actions are required. Nevertheless, a certain advantage of the evaluation function P is its simplicity. When an appropriate definition of the Euclidean distance of the problem states can be determined, it can be readily used to optimize the policy by the gradient method. But, of course, it is not a "true" value function and may not be a "good" evaluation function.

Given a precise value function, an optimal action is the action that leads to the next state with the highest value of the state or the state-action. This means that we can directly go to the goal state along the shortest trajectory on the value function. But this is not true if we use P instead of the correct value function. Sometimes, even if a current state is close enough to the goal state, the shortest trajectory on P may be not the best path to go. And, this is most likely true in the complicated control problems with the twisted value function surface. Nevertheless, many realistic control problems can be thought of having a smooth value function, especially in the neighborhood of the goal states. Hence, a simple approximated value function P still has a chance of being used as an *approximated* evaluation function to tune an action-selection policy.

CAPS learns to produce appropriate actions by adjusting $\mathbf{w}(t)$ using the evaluation function $P(\mathbf{s}(t), u(t))$. This adjusment of $\mathbf{w}(t)$ takes effect on both the action $u(t) = f_{\mathbf{w}(t)}(\mathbf{s}(t))$ and the evaluation function $P(\mathbf{s}(t), u(t))$. In the following explanation, for clarity, we use the notation $P_{\mathbf{w}(t)}$ in place of $P(\mathbf{s}(t), u(t))$ to represent how good $\mathbf{w}(t)$

at the state $\mathbf{s}(t)$, and rewrite (3.1) as follows:

$$P_{\mathbf{w}(t)} = \frac{1}{2}D^2(t), \quad D(t) = \sqrt{e^2(\mathbf{s}(t+\triangle t)) + \alpha u(t)^2}. \tag{3.2}$$

CAPS in Fig. 3.1 adjusts an element of $\mathbf{w}(t)$ by using a gradient descent method as follows:

$$\frac{dw(t)}{dt} = -\eta \frac{\partial P_{\mathbf{w}(t)}}{\partial w(t)}, \tag{3.3}$$

where η is a positive-definite step size, and $w(t)$ denotes an element of $\mathbf{w}$ at time t. This is a natural strategy if the partial derivative of $P_{\mathbf{w}(t)}$ with respect to $w(t)$ exists. In such a case, $\mathbf{w}(t)$ can usually be assured to converge to a local optimal point of the evaluation function $P_{\mathbf{w}(t)}$. Unfortunately, it is impossible to get this derivative. To solve this problem, the author applies the following chain rule:

$$\frac{\partial P_{\mathbf{w}(t)}}{\partial w(t)} = D(t)\frac{\partial D(t)}{\partial w(t)} = D(t)\frac{\partial D(t)}{\partial u(t)}\frac{\partial u(t)}{\partial w(t)}. \tag{3.4}$$

In this equation, $\partial u(t)/\partial w(t)$ depends on the kind of controller used. For simplicity, the author uses $\phi_{w(t)}$ to denote this partial derivative.

The partial derivative $\partial D(t)/\partial u(t)$ is still difficult to compute because the plant dynamics function is not given. Hence, the partial derivative is approximated as,

$$\frac{\partial D(t)}{\partial u(t)} \approx \frac{\triangle D(t)}{\triangle u(t)} = \frac{D(t) - D(t-\triangle t)}{u(t) - u(t-\triangle t)}. \tag{3.5}$$

Substituting this approximation into (3.4), and using (3.3), we obtain the following update rule:

$$w(t+\triangle t) = w(t) - \eta D(t)\frac{\triangle D(t)}{\triangle u(t)}\phi_{w(t)}\triangle t. \tag{3.6}$$

Tha author calls the method of tuning $\mathbf{w}$ based on the update rule (3.6) as the *Approximated Gradient Descent Method* (AGDM).

3.4 Coping with Approximation Errors

Equation (3.5) is a crude approximation because any state change between consecutive time steps is missed and ignored. Hence, the update rule (3.6) needs modifications to cope with the approximation errors.

In Equation (3.5), when $u(t) - u(t-\triangle t)$ becomes very small as the plant state approaches to the goal state, the approximated gradient might get unacceptable. To

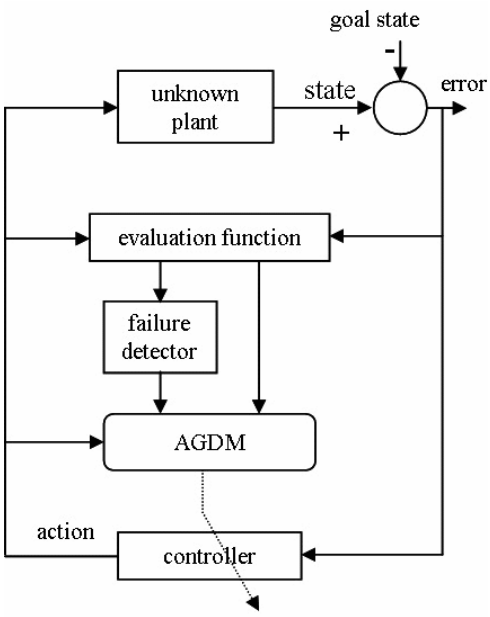


Figure 3.2: CAPS with failure detector.

avoid such deleterious influences of possible errors in Equation (3.5) when tuning the values of $\mathbf{w}(t)$, we change the update rule (3.6) as follows:

$$w(t+\triangle t) = w(t) - \eta D(t)\,\mathrm{sign}\Big(\frac{\triangle D(t)}{\triangle u(t)}\Big)\,\phi_{w(t)}\triangle t. \tag{3.7}$$

In this update rule, only slight changes are made to the values of $w(t+\triangle t)$, no matter how big the value of $\triangle D(t)/\triangle u(t)$ is.

And, Figure 3.2 depicts the modified CAPS where a *failure detector* is introduced to tune $\mathbf{w}(t)$ only when it is necessary and appropriate. The failure detector outputs 1 whenever the evaluation function $P_{\mathbf{w}(t)}$ is getting worse, and outputs 0 otherwise. Let $f(t)$ be a symbol for such outputs, then we can define $f(t)$ as follows:

$$f(t) = \begin{cases} 1, & \text{if } P_{\mathbf{w}(t)} > P_{\mathbf{w}(t-\triangle t)}, \\ 0, & \text{otherwise.} \end{cases} \tag{3.8}$$

By incorporating the failure detector, the AGDM updates $w(t)$ as follows:

$$w(t+\triangle t) = w(t) - f(t)\eta D(t)\,\mathrm{sign}\Big(\frac{\triangle D(t)}{\triangle u(t)}\Big)\,\phi_{w(t)}\triangle t. \tag{3.9}$$

The intuitive explanation on the role of $f(t)$ is as follows: If a current state is getting closer to the goal state, then the value $P_{\mathbf{w}(t)}$ must be decreasing and it means the current $\mathbf{w}(t)$ is "good". In such a good situation, it is reasonable to keep $\mathbf{w}(t)$ unchanged by setting $f(t) = 0$ because any attempt to update $\mathbf{w}(t)$ might make the situation worse. On the contrary, if the current state is stepping away from the goal state, then the value $P_{\mathbf{w}(t)}$ must be increasing. It means the current $\mathbf{w}(t)$ is "bad" and must be updated to decrease $P_{\mathbf{w}(t)}$ by setting $f(t) = 1$.

The update rule of (3.9) is generic in that it is applicable for tuning any kind of controller provided that the derivative of the action with respect to the controller parameters (i.e., $\phi_{w(t)}$) is given. In the following sections, the author discusses applications of the update rule of (3.9) for adjusting the linear state feedback controller and the fuzzy controller.

3.5 Linear Controller with Approximated Policy Search Approach (LCAPS)

In LCAPS, actions are produced by a linear state feedback controller. This section discusses first the defintion of the linear state feedback controller, followed by the derivation of update rule for the linear state feedback controller in LCAPS. Simulation results of applying LCAPS for solving a bencmark problem, i.e., a pole-balancing problem, are then given.

3.5.1 Linear State Feedback Controller

As implied by its name, the linear state feedback controller is linear in *state feedback*. In control theory, the term "state" refers to a state of a dynamic plant, or more general, a dynamic system. Ogata [38] defines the state of the dynamic system as the smallest set of variables (called *state variables*) such that the knowledge of these variables at $t = t_0$, together with the knowledge of the system input for $t \geq t_0$, completely determines the behavior of the plant for any time $t \geq t_0$. Such a definition of the state is by no means limited to physical systems, rather also applicable to biological, economics, social systems, and others.

Further, the term "state variables" means the variables making up the smallest set of variables that determine the state of the dynamic system. Suppose that at least n variabels $s_1, s_2, s_3, \cdots, s_n$ are required to completely describe the dynamic system behavior so that once the system input is given for $t \geq t_0$ and the initial state at $t = t_0$

is specified, the future state of the system is completely determined. These n variabels constitute a set of state variables.

Other term related to the term "state" is *state vector*. If the n state variables $s_1, s_2, s_3, \cdots, s_n$ are used to completely describe the system behavior, then they are considered as the n components of a state vector that can be written as $\mathbf{s} = [s_1\ s_2\ s_3\ \cdots\ s_n]^T$. Thus, the state vector determines *uniquely* the system state $\mathbf{s}(t)$ for all $t \geq t_0$, once the state at $t = t_0$ is given and the system input for $t \geq t_0$ is specified. In many control literatures, "state vector" is rarely mentioned, rather simply "state" to refer to the state vector.

Generally, the state need not be composed of physically measurable or observable state variables. Any variable that does not represent physically measurable quantity can be chosen as a state variable. Nevertheless, in real control, it is convenient to choose only easily measurable quantities for the state variables. Many control methods require completely measureable state variables.

In the dissertation, by the linear state feedback controller the author means a controller that computes its output simply as a total sum of all weighted state variables of the state feedback. This definition implies that for the controller to produce its outputs, all state variables must be measurable. Let $\mathbf{w}(t) = \{w(t)\}$ denote a weight column vector of the linear state feedback controller at time t. Similarly, let $\mathbf{s}(t) = \{s(t)\}$ denote a column state vector of the plant. Using the above definition, the linear state feedback controller can be written as follows

$$u(t)(\mathbf{s}(t)) = \mathbf{w}(t)^T \mathbf{s}(t). \tag{3.10}$$

From (3.10), we obtain

$$\phi_{w(t)} = \frac{\partial u(\mathbf{s}(t))}{\partial w(t)} = s(t). \tag{3.11}$$

Hence, the update rules for tuning the linear state feedback controller in LCAPS can be obtained by substituting $\phi_{w(t)}$ of (3.11) into (3.9).

3.5.2 Experimental Results

To evaluate performances of LCAPS, a cart-pole balancing plant as shown in Fig. 3.3 is used as a benchmark problem for the experiments. The cart-pole plant has four state variables: $s_1 = \theta$ (angle of pole with the vertical), $s_2 = \dot{\theta}$ (pole angular velocity), $s_3 = x$ (cart position on a track), and $s_4 = \dot{x}$ (cart velocity). But, in this simulation, only the first two state variables are taken into consideration to make comparison with

the past research results [23, 36, 42]. Dynamic equation of the cart-pole plant are as follows:

$$\begin{aligned}
\dot{s}_1(t) &= s_2(t), \\
\dot{s}_2(t) &= \frac{g \sin s_1(t) + \cos s_1(t) \left(\dfrac{-u(t) - mls_2^2(t) \sin s_1(t)}{m_c + m} \right)}{l \left(\dfrac{4}{3} - \dfrac{m \cos^2 s_1(t)}{m_c + m} \right)}, \\
\dot{s}_3(t) &= s_4(t), \\
\dot{s}_4(t) &= \frac{u(t) + ml \left(s_2^2(t) \sin s_1(t) - \dot{s}_2(t) \cos s_1(t) \right)}{m_c + m},
\end{aligned} \tag{3.12}$$

where g represents the acceleration of gravity, m_c is the cart mass, m is the pole mass, l is the half-pole length, and u is the force applied to the cart. In (3.12), the coefficients of friction of the pole on the cart and the cart on the track are ignored. The cart-pole plant dynamics of (3.12) is almost linear when the pole angle is near the upright position and the cart is near the center. In contrast, its nonlinearity increases drastically when the pole angle is of large values.

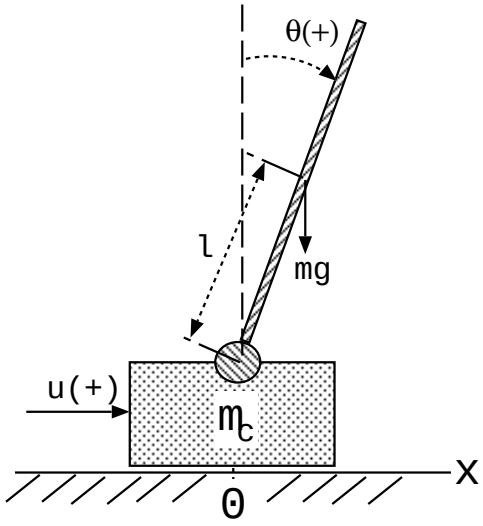


Figure 3.3: Cart-pole plant

For these experiments, the cart-pole plant parameters are set as follows. $g = 9.81$ ms^{-2}, $m_c = 1.0$ kg, $m = 0.1$ kg, and $l = 0.5$ m. The above cart-pole plant dynamics were then simulated using the 4th-order Runge-Kutta method with a time step of $\triangle t = 10$ ms.

In the experiments, the author considers three types of problems: (1) *Set-point problems* where the goal state is fixed, (2) *Tracking problems* where the goal state is

changing, and (3) *Swinging-up problems* where the goal state is fixed or changing, but the pole is in a downward position at rest initially.

In this section, the linear state feedback controller using the update rule (3.7) is referred to as LCAPS, and the system with a failure detector using the update rule (3.9) is referred to as LCAPS/FD.

All the elements of $\mathbf{w}$ in the linear state feedback controller in LCAPS are initialized to zero and the controller output is limited within the range of $[-20, 20]$ N. The learning rate is chosen as $\eta = 5000$, while α is set to 1×10^{-5}, except for the swinging-up problems where α is set to 1×10^{-6}.

3.5.2.1 Set-point Problems

In the set-point problems, the goal is to balance the pole in an upright position (i.e., $\theta_{goal} = 0$ deg and $\dot{\theta}_{goal} = 0$ deg/s).

The angular responses due to the application of both LCAPS and LCAPS/FD to the plant are shown in Fig. 3.4(a). These graphs show that LCAPS/FD successfully balanced the pole initialized at 30 deg, but LCAPS failed.

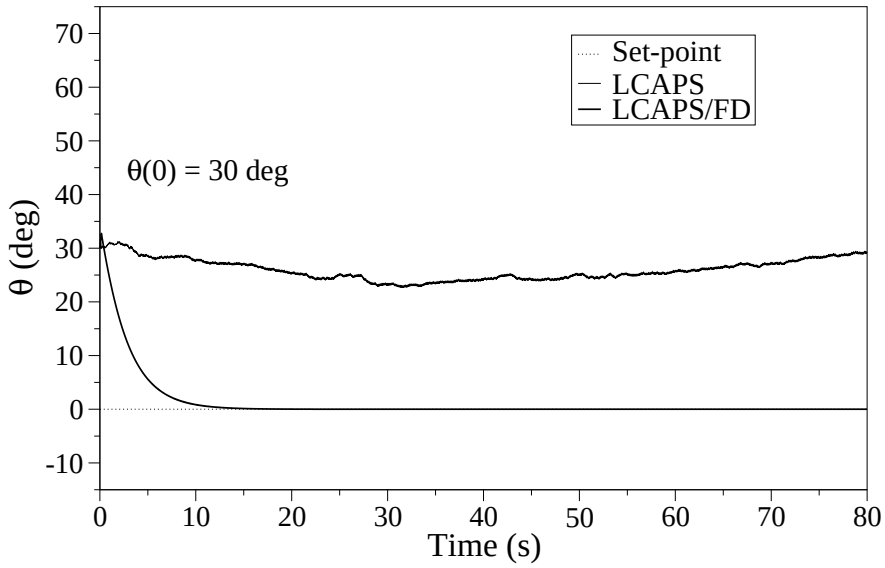
The effectiveness of applying the failure detector in LCAPS/FD can be shown also in the produced action behaviors. LCAPS/FD (see the graph in Fig. 3.4(b)) produces smooth action behaviors, so that LCAPS/FD is able to generate the action sequence that is suitable for realistic control.

3.5.2.2 Tracking Problems

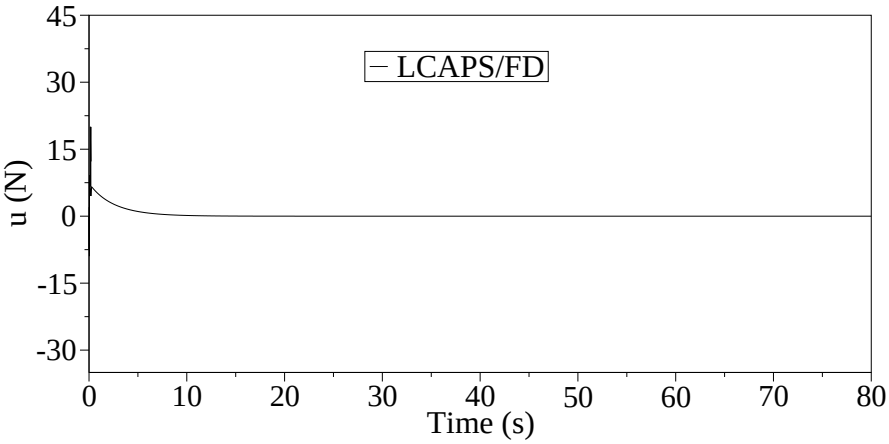
The second set of experiments are concerned with the tracking problem in which the desired θ is changing. In this simulation, we set the trajectory of θ to be tracked by FCAPS as $\Theta(t) = (\pi/30)\sin(t)$ rad. With this trajectory, the pole periodically oscillates around the vertical position with the maximum deviation of $\pi/30$ rad.

Figure 3.5(a) shows the angular responses of both LCAPS's in the tracking problems. The graphs show that LCAPS/FD controlled the pole successfully to follow the desired trajectory, but LCAPS failed.

In the tracking problems, the effectiveness of applying the failure detector in LCAPS/FD can also be shown in the produced action behaviors. Like in the set-point problems, LCAPS/FD (see the graph in Fig. 3.5(b)) produces smooth action behaviors. The actions in Fig. 3.5(b) do not converge to zero but oscillate slightly around zero because the goal state oscillates around zero to follow the desired trajectory $\Theta(t) = (\pi/30)\sin(t)$.



(a)

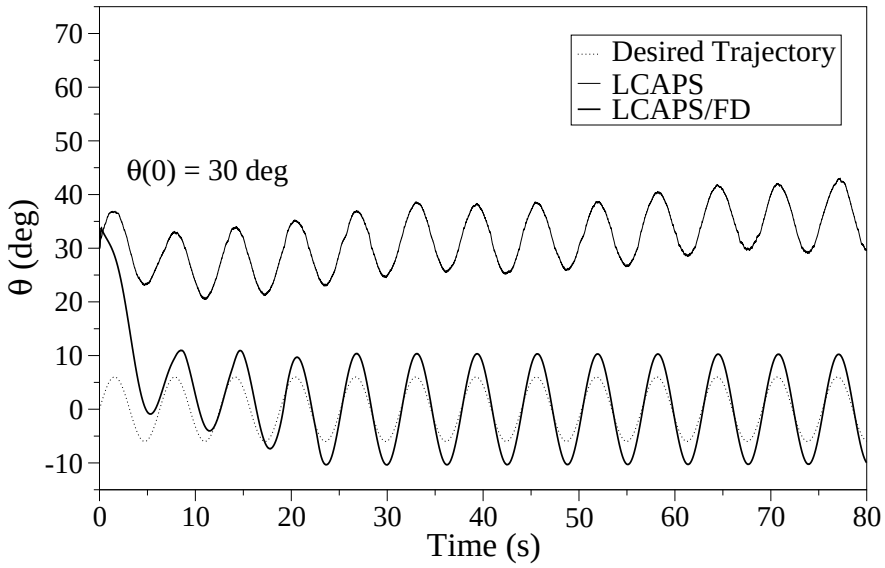


(b)

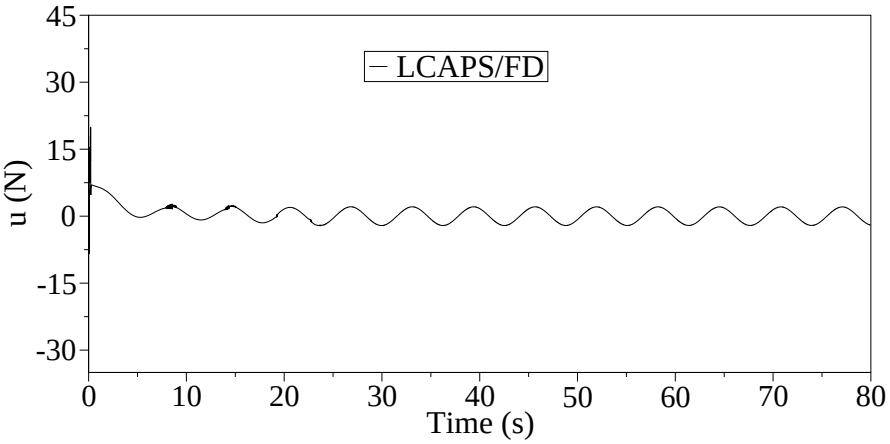
Figure 3.4: LCAPS/FD for solving set-point problems: (a) angular responses, (b) actions

3.5.2.3 Swinging-up Problems

In the swinging-up problems, the pole pointing downward at rest initially (i.e., $\theta_0 = 180^0$ and $\dot{\theta}_0 = 0$ rad/s) must be swung up and then balanced in an upright position or kept following a desired trajectory. This goal is the same as that of the set-point or tracking problems. In this section, since LCAPS failed in both types of the problems, we focus on the experiments with LCAPS/FD only.



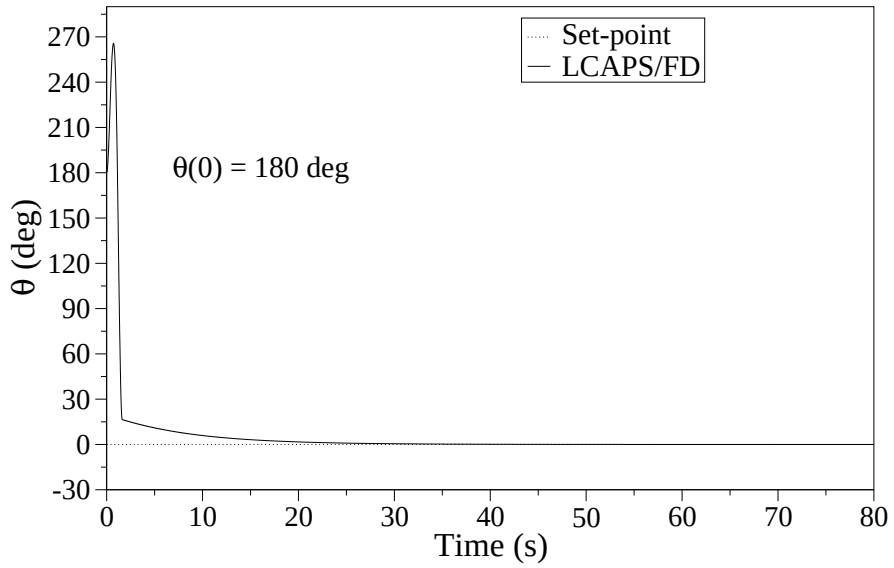
(a)



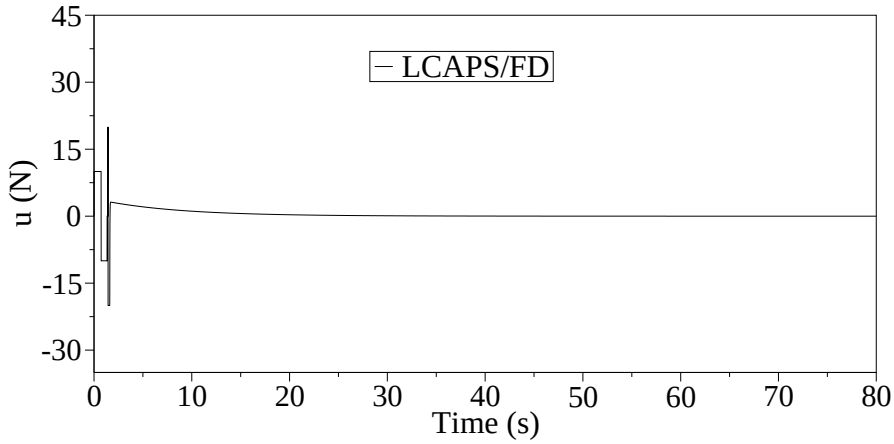
(b)

Figure 3.5: LCAPS/FD for solving tracking problems: (a) angular responses, (b) actions

In the swinging-up problems, it will be very difficult for LCAPS/FD to achieve the same goal of the set-point or tracking problems directly. The simplest way to solve this problem is to swing the pole progressively higher from its rest downward position until it reaches a region where LCAPS/FD can work well. To implement this idea, the pole



(a)



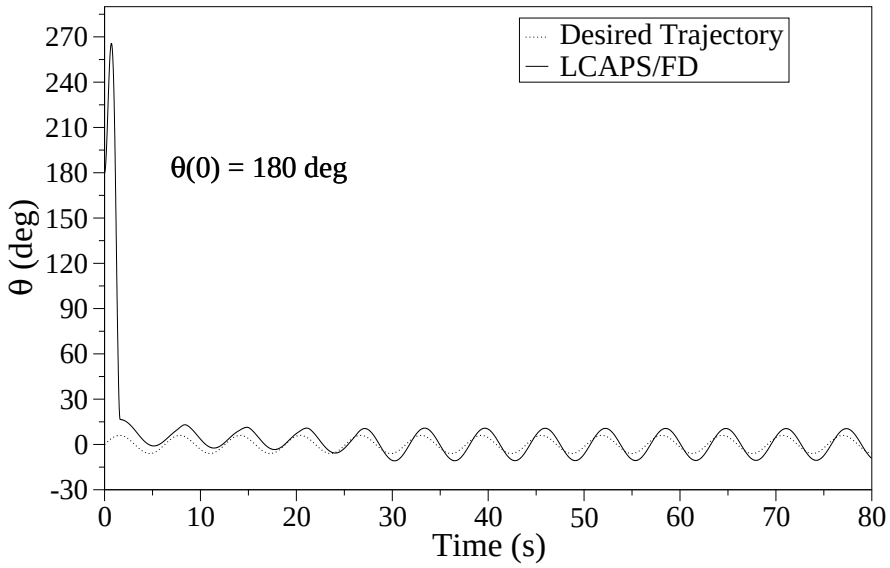
(b)

Figure 3.6: LCAPS/FD for solving swinging-up and set-point problems: (a) angular responses, (b) actions

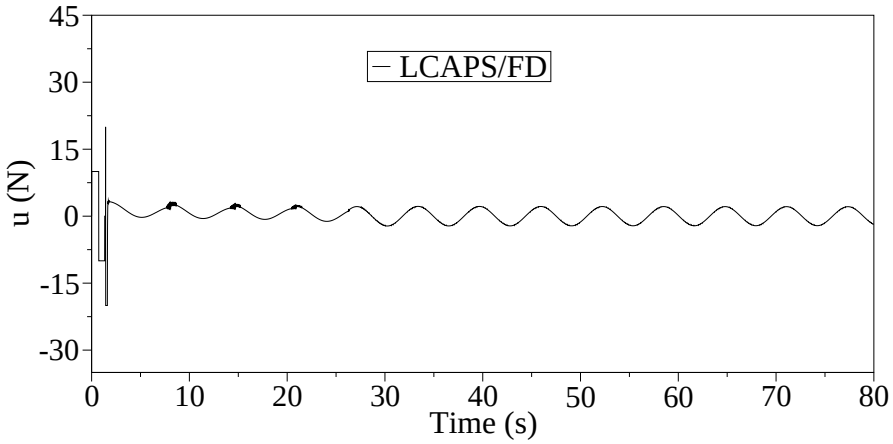
is swung up by applying the following swinging actions:

$$u = \begin{cases} -10 \text{ N}, & \forall \ 90 \text{ deg } < \theta < 270 \text{ deg } \text{ and } \dot{\theta} < 0 \text{ deg /s} \\ 10 \text{ N}, & \forall \ 90 \text{ deg } < \theta < 270 \text{ deg } \text{ and } \dot{\theta} \geq 0 \text{ deg /s}. \end{cases} \tag{3.13}$$

And then, LCAPS/FD is applied only when the angle of the pole is within $|\theta| < 60$ deg in which we assume that LCAPS/FD can work well.



(a)



(b)

Figure 3.7: LCAPS/FD for solving swinging-up and tracking problems: (a) angular responses, (b) actions

When the pole is entering the range $|\theta| < 60$ deg for the first time, LCAPS/FD is given the "initial condition" of angle $|\theta_0| \approx 60$ deg . But, we cannot know *a priori* the "initial condition" of $\dot{\theta}$ at this time, because it depends on the total amount of energy pumped to the pole by the swinging actions in (3.13). This amount of energy can be determined through experiments only. Intuitively, we can say that LCAPS/FD tends to fail to balance the pole, given too high values of $\dot{\theta}$. To avoid this problem, we

must be careful in setting the value of the swinging action for each situation in (3.13), because either too weak or too strong swinging actions may result in too high angle velocity of the pole. If the swinging actions are too weak, the pole takes long time to swing up and enters the region $|\theta| < 60$ deg . And, swinging the pole for too long time will pump very large energy to it. On the contrary, if the swinging actions are too strong, the pole will also receive very large energy in much shorter time. And, in both cases, the angle velocity of the pole $\dot{\theta}$ becomes very high when the pole enters the region $|\theta| < 60$ deg .

Several researchers in the past have proposed the energy control approaches to solve the swinging-up problems (see [53, 54]). The energy control approach regards the acceleration of the cart as the control action to meet the goal. Using such a control action, the pole is moved from its rest downward position to its upright position by controlling its energy instead of controlling its angle and velocity directly. The sequence of actions must result in an accumulated energy that corresponds to the upright position. When the accumulated energy is larger or less than an amount of the energy corresponding to the upright position, it must be decreased or increased by applying the decreased or increased acceleration of the cart accordingly.

Despite its ease to implement, the energy control approach cannot apply its actions to the cart directly. This approach demands a precise dynamics function of the plant to convert its planned control action to the actual driving force that is readily applied to the cart. This convertion is not as simple as multiplying the acceleration with the mass of the cart, since it must include the mass, angle and angle velocity of the pole as well (see [54] for its detail). Whereas, LCAPS/FD produces the driving force directly that is readily applied to the cart without including any dynamics function of the plant.

Figures 3.6(a) and 3.7(a) show the angular responses of LCAPS/FD in the swinging up problems followed by the set-point and tracking problems, respectively. The graphs in both figures show that, by applying the actions in (3.13), the angle of the pole could be moved to the desired region after about 2.0 seconds. Afterward, LCAPS/FD succeeded to balance the pole in the upright position and keep it following the desired trajectory. Figures 3.6(b) and 3.7(b) show the actions for the angular responses in Fig. 3.6(a) and 3.7(a), respectively. The graphs in both figures show smooth action behaviors produced by LCAPS/FD.

3.6 Fuzzy Controller with Approximated Policy Search Approach (FCAPS)

A *fuzzy controller* has been employed in a wide range of control problems. However, tuning fuzzy rules used in the fuzzy controller still remains as a difficult task for its successful application to practical problems. For this reason, developing a self-learning fuzzy controller is of keen interest to many researchers [5, 43–49]. In this section, an alternative sel-tuning fuzzy controller called FCAPS is discussed. The first discussion is regarding the fuzzy controller structure. Simulation results of applying FCAPS for solving the pole-balancing problem are then given.

3.6.1 Fuzzy Controller

There are two alternatives for tuning the fuzzy controller. The first is structural learning in which only the number of rules, which is dependent of the number of fuzzy sets per state variable, is tuned. The second is parametric learning in which only the parameters of the fuzzy controller are tuned. In this paper, the parameters of the fuzzy controller mean the fuzzy set positions of both the input parts (or, IF-parts) and the output parts (THEN-parts) of the fuzzy rule.

Simultaneous application of both learning methods are possible only at the expense of a very large search space and a complex performance evaluation surface. Parametric learning alone has a difficult problem to be solved: when the output of the fuzzy controller is incorrect, it can be corrected by tuning the parameters in either input or output parts of the rules. It is difficult to tell which part contributes the incorrect output and should be updated. To avoid this problem, only the parameters of the output part are tuned in this research and the structure of the fuzzy controller is given as follows.

The input of the fuzzy controller is a state $\mathbf{s}$ of the plant. Let n be a size of the state $\mathbf{s}$. For the i-th state variable s_i $(i = 1, 2, 3, ..., n)$, we define p_i fuzzy sets or membership functions, each of which is denoted by $A_i^{l_i}$ $(l_i = 1, 2, 3, ..., p_i)$. The fuzzy controller is constructed with all possible combinations of the predefined membership functions, i.e., we will have $\prod_{i=1}^{n} p_i$ rules, each of which is:

$$\begin{aligned} &IF\ s_1\ is\ A_1^{l_1}\ and\ ...\ and\ s_n\ is\ A_n^{l_n} \\ &\qquad THEN\ f(\mathbf{s})\ is\ W^{l_1 \ldots l_n}, \end{aligned} \tag{3.14}$$

where $W^{l_1 \ldots l_n}$ denotes a fuzzy set label of the output part. In this research, a membership function $A_i^{l_i}$ of the state variable s_i is represented by a Gaussian function $\mu_{A_i^{l_i}}(s_i)$, and the parameters of the output part are represented by an adjustable vector $\mathbf{w} = \{w_{l_1 \ldots l_n}\}$.

Using a product inference system, singleton fuzzifier, and center of average defuzzifier [5], the fuzzy controller can be written in the following form:

$$u(\mathbf{s}) = \frac{\sum_{l_1=1}^{p_1} \cdots \sum_{l_n=1}^{p_n} w_{l_1 \ldots l_n} (\prod_{i=1}^{n} \mu_{A_i^{l_i}}(s_i))}{\sum_{l_1=1}^{p_1} \cdots \sum_{l_n=1}^{p_n} (\prod_{i=1}^{n} \mu_{A_i^{l_i}}(s_i))}. \tag{3.15}$$

From this equation, $\phi_{w(t)}$ can be easily calculated as follows:

$$\phi_{w(t)} = \frac{\partial u(t)}{\partial w(t)} = \frac{\prod_{i=1}^{n} \mu_{A_i^{l_i}}(s_i)}{\sum_{l_i=1}^{p_1} \cdots \sum_{l_n=1}^{p_n} (\prod_{i=1}^{n} \mu_{A_i^{l_i}}(s_i))}. \tag{3.16}$$

Hence, the update rules for tuning the linear state feedback controller in FCAPS can be obtained by substituting $\phi_{w(t)}$ of (3.16) into the aforementioned generic update rules of CAPS.

3.6.2 Experimental Results

To evaluate performances of FCAPS, the pole-balancing plant as shown in Fig. 3.3 with the dynamics of (3.12) is used as the benchmark problem for the experiments. For the state variables: $s_1 = \theta$ and $s_2 = \dot{\theta}$, we define five Gaussian membership functions (i.e., $p_1 = p_2 = 5$), with the centers at $\{-30, -20, 0, 20, 30\}$ deg and $\{-60, -30, 0, 30, 60\}$ deg/s, respectively, and standard deviations: $\{20, 10, 10, 10, 20\}$ deg and $\{30, 15, 10, 15, 30\}$ deg/s, respectively. Note that beyond the range [-30,30] deg for θ and [-60,60] deg/s for $\dot{\theta}$, the state variables will be assigned with the maximum degree of membership (i.e., 1.0). All the elements of $\mathbf{w}$ in the fuzzy controller are initialized to zero and the controller output is limited within the range of $[-30, 30]$ N. The learning rate is chosen as $\eta = 5000$, while α is set to 1×10^{-5}.

By defining five membership functions for each state variables, the controller parameter vector $\mathbf{w}$ will have the size of 25. Given big number of controller parameters, the surface of the evaluation function becomes very complex where it may have many local optima which make FCAPS difficult to find optimum $\mathbf{w}$. And, since we do not introduce any prior knowledge of the plant to initialize $\mathbf{w}$ (instead, all its elements are simply set to zero), FCAPS will have a heavy burden of adjusting $\mathbf{w}$ initially.

There have been many proposed methods to balance the pole [5, 43]. They include prior knowledge of the plant and reduce the number of parameters of the fuzzy controller to make tuning easier. In addition, they use the normalized values of the state variables to reduce the search space of controller parameters.

But, we are not primarily interested in solving pole-balancing problem using FCAPS. Instead, we simply set the controller parameters as explained above intentionally to make the problem of balancing the pole more difficult. This will give us a fair evaluation of FCAPS againts difficult control problems. While a variety of well-developed adaptive tuning method can be (and has been) successfully applied to the pole-balancing problem, they may not be applicable to the problem with the simple setting explained above.

In the experiments, we consider three types of problems: (1) *Set-point problems* where the goal state is fixed, (2) *Tracking problems* where the goal state is changing, and (3) *Swinging-up problems* where the goal state is fixed or changing, but the pole is in a downward position at rest initially.

In this section, the fuzzy controller system using the update rule (3.7) is referred to as FCAPS, and the system with a failure detector using the update rule (3.9) is referred to as FCAPS/FD.

3.6.2.1 Set-point Problems

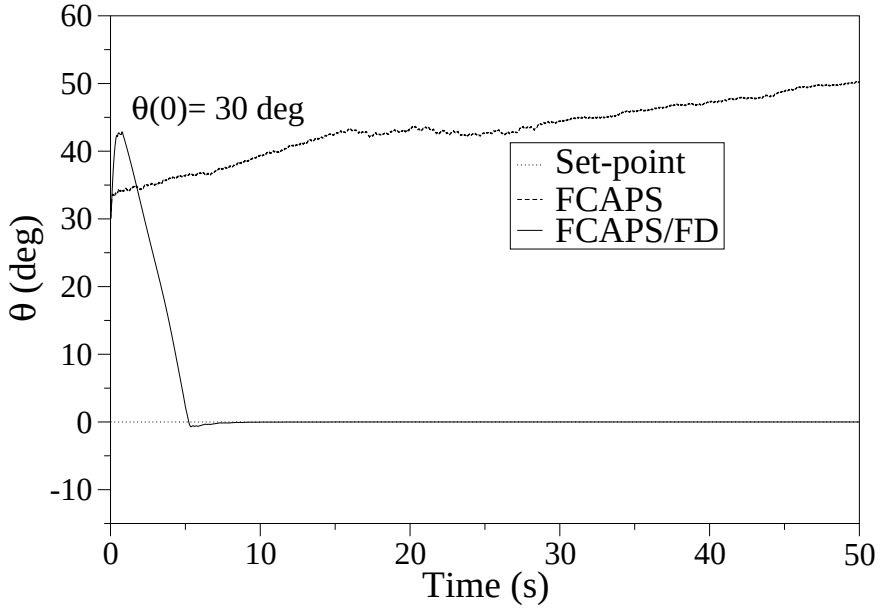
In the set-point problems, the goal is to balance the pole in an upright position (i.e., $\theta_{goal} = 0$ deg and $\dot{\theta}_{goal} = 0$ deg/s).

The angular responses due to the application of both FCAPS and FCAPS/FD to the plant are shown in Fig. 3.8(a). These graphs show that FCAPS/FD successfully balanced the pole initialized at 30 deg, but FCAPS failed.

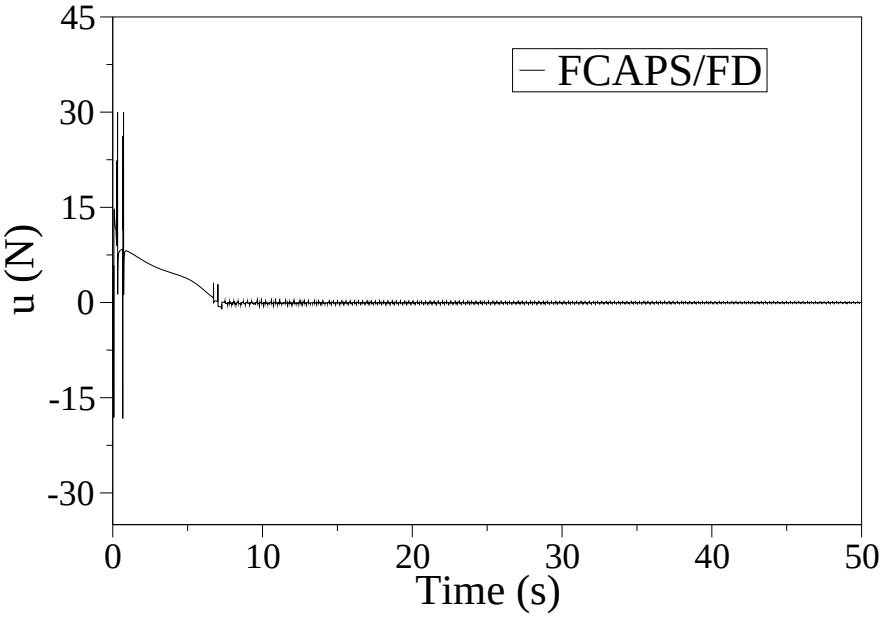
The effectiveness of applying the failure detector in FCAPS/FD can be shown also in the produced action behaviors. FCAPS/FD (see the graph in Fig. 3.8(b)) produces smooth action behaviors, so that FCAPS/FD is able to generate the action sequence that is suitable for realistic control.

3.6.2.2 Tracking Problems

The second set of experiments are concerned with the tracking problem in which the desired θ is changing. In this simulation, we set the trajectory of θ to be tracked by FCAPS as $\Theta(t) = (\pi/30)\sin(t)$ rad. With this trajectory, the pole periodically oscillates around the vertical position with the maximum deviation of $\pi/30$ rad.

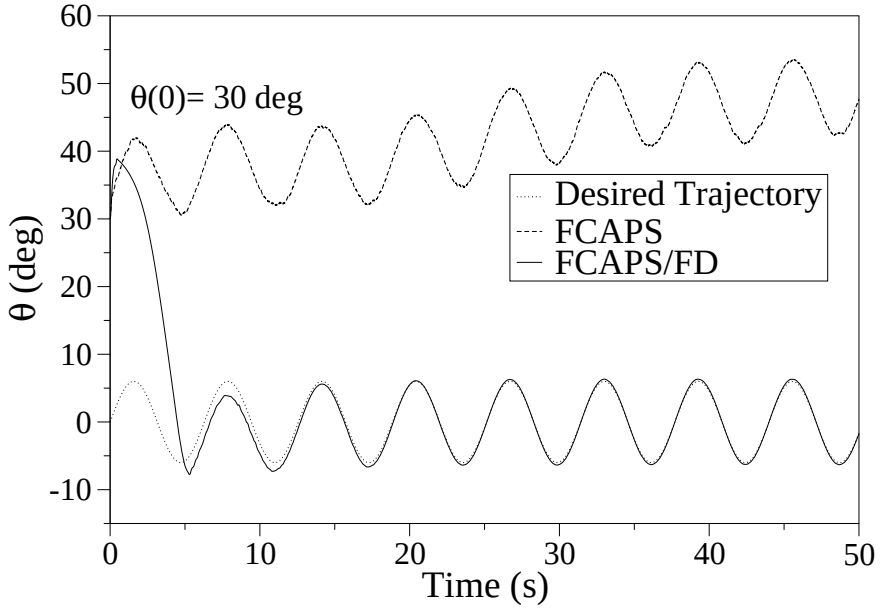


(a)

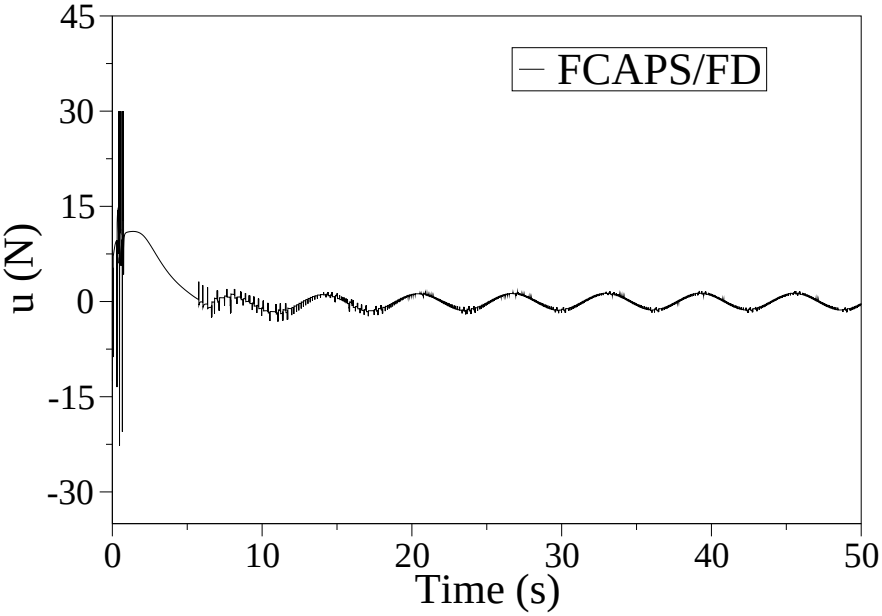


(b)

Figure 3.8: FCAPS/FD for solving set-point problems: (a) angular responses, (b) actions



(a)



(b)

Figure 3.9: FCAPS/FD for solving tracking problems: (a) angular responses, (b) actions

Figure 3.9(a) shows the angular responses of both FCAPS's in the tracking problems. The graphs show that FCAPS/FD controlled the pole successfully to follow the desired trajectory, but FCAPS failed.

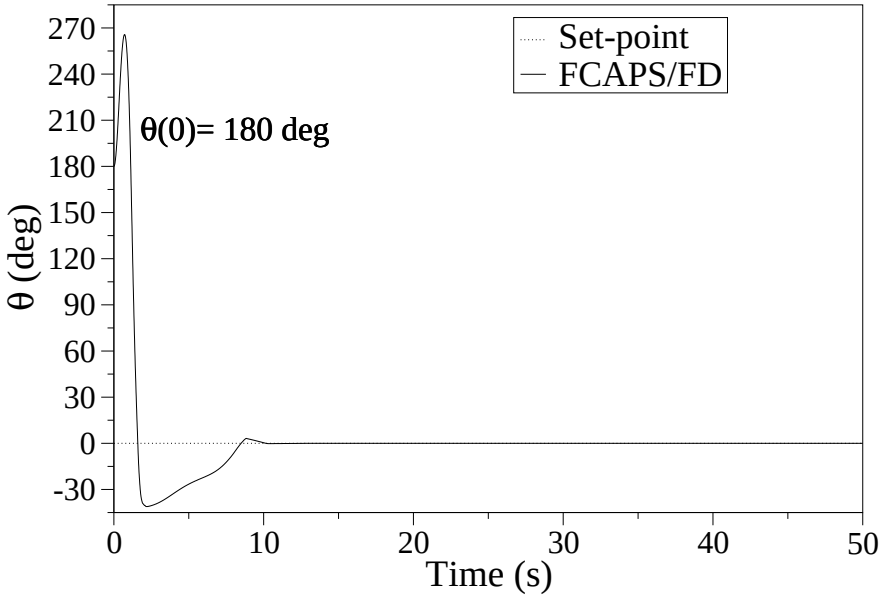
In the tracking problems, the effectiveness of applying the failure detector in FCAPS/FD can also be shown in the produced action behaviors. Like in the set-point problems, FCAPS/FD (see the graph in Fig. 3.9(b)) produces smooth action behaviors. The actions in Fig. 3.9(b) do not converge to zero but oscillate slightly around zero because the goal state oscillates around zero to follow the desired trajectory $\Theta(t) = (\pi/30)\sin(t)$ rad.

3.6.2.3 Swinging-up Problems

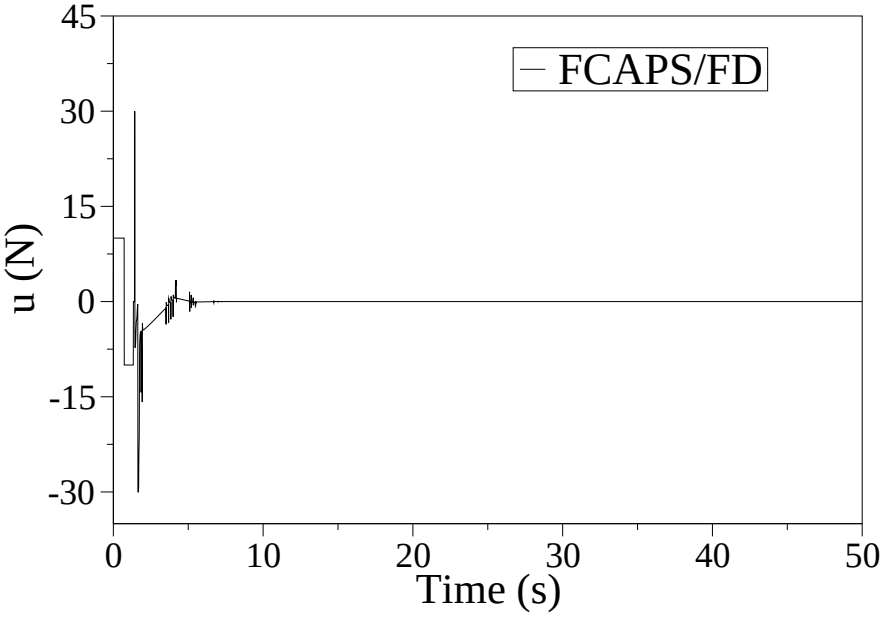
In the swinging-up problems, the pole pointing downward at rest initially (i.e., $\theta_0 = 180$ deg and $\dot{\theta}_0 = 0$ rad/s) must be swung up and then balanced in an upright position or kept following a desired trajectory. This goal is the same as that of the set-point or tracking problems. In this section, since FCAPS failed in both types of the problems, we focus on the experiments with FCAPS/FD only.

The initial angle of the pole pointing downward is far beyond the range of θ defined in FCAPS/FD (i.e., [-30,30] deg). Therefore, in the swinging-up problems, it will be very difficult for FCAPS/FD to achieve the same goal of the set-point or tracking problems directly. The simplest way to solve this problem is to swing the pole progressively higher from its rest downward position until it reaches a region where FCAPS/FD can work well. To implement this idea, the pole is swung up by applying the following swinging actions of (3.13). And then, FCAPS/FD is applied only when the angle of the pole is within $|\theta| < 60$ deg in which we assume that FCAPS/FD can work well.

When the pole is entering the range $|\theta| < 60$ deg for the first time, FCAPS/FD is given the "initial condition" of angle $|\theta_0| \approx 60$ deg. But, we cannot know *a priori* the "initial condition" of $\dot{\theta}$ at this time, because it depends on the total amount of energy pumped to the pole by the swinging actions in (3.13). This amount of energy can be determined through experiments only. Intuitively, we can say that FCAPS/FD tends to fail to balance the pole, given too high values of $\dot{\theta}$. To avoid this problem, we must be careful in setting the value of the swinging action for each situation in (3.13), because either too weak or too strong swinging actions may result in too high angle velocity of the pole. If the swinging actions are too weak, the pole takes long time to swing up and enters the region $|\theta| < 60$ deg. And, swinging the pole for too long time will pump very large energy to it. On the contrary, if the swinging actions are

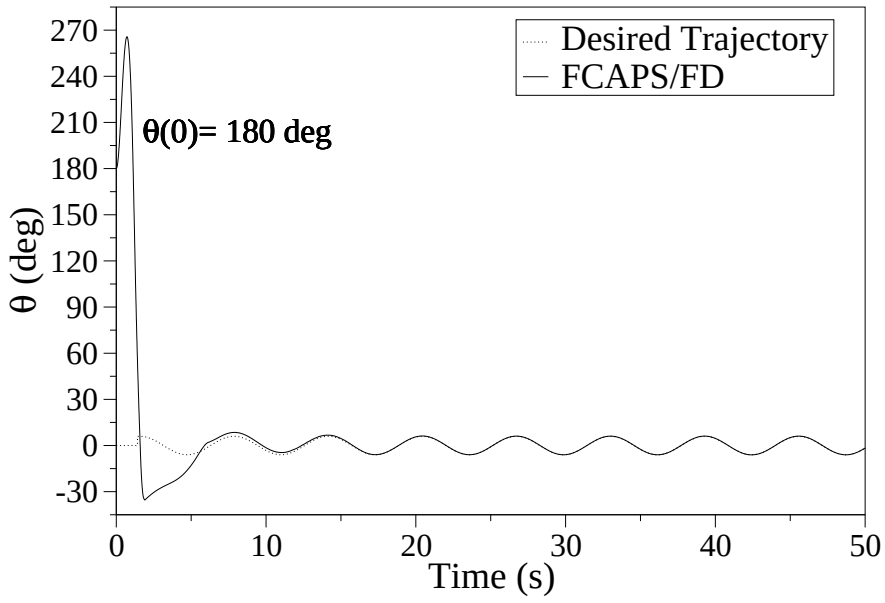


(a)

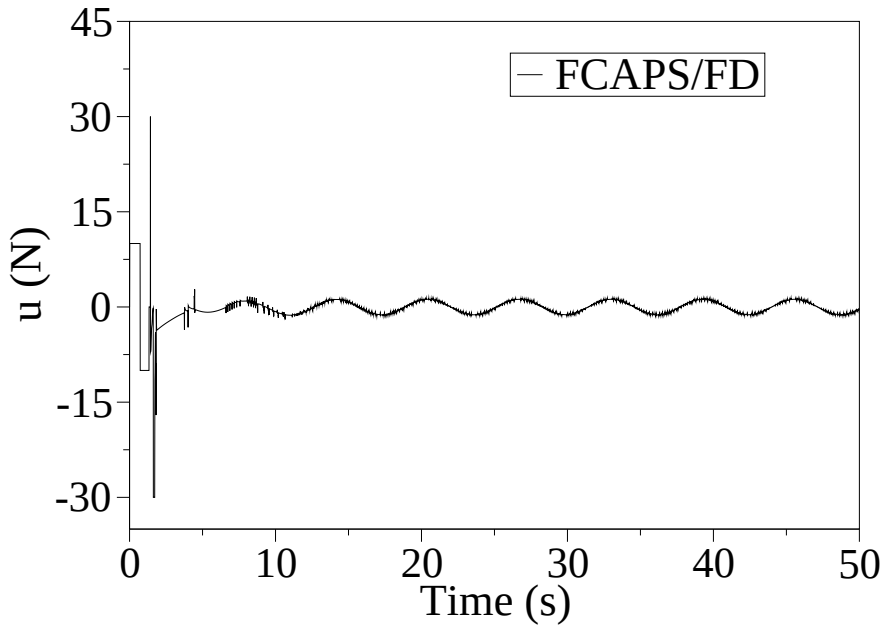


(b)

Figure 3.10: FCAPS/FD for solving swinging-up and set-point problems: (a) angular responses, (b) actions



(a)



(b)

Figure 3.11: FCAPS/FD for solving swinging-up and tracking problems: (a) angular responses, (b) actions

too strong, the pole will also receive very large energy in much shorter time. And, in both cases, the angle velocity of the pole $\dot{\theta}$ becomes very high when the pole enters the region $|\theta| < 60$ deg.

Figures 3.10(a) and 3.11(a) show the angular responses of FCAPS/FD in the swinging up problems followed by the set-point and tracking problems, respectively. The graphs in both figures show that, by applying the actions in (3.13), the angle of the pole could be moved to the desired region after about 1.5 seconds. Afterward, FCAPS/FD succeeded to balance the pole in the upright position and keep it following the desired trajectory. Figures 3.10(b) and 3.11(b) show the actions for the angular responses in Fig. 3.10(a) and 3.11(a), respectively. The graphs in both figures show smooth action behaviors produced by FCAPS/FD.

3.7 Analisys of Results

An accurate evaluation function can represent the dynamics of the system (the plant and the controller). Given any state of the plant, the accurate evaluation function can tell the controller the best action to take because it can predict the result of executing actions precisely.

The accuracy of the evaluation function $P_{\mathbf{w}(t)}$ is not guaranteed. But, using $P_{\mathbf{w}(t)}$, both LCAPS and FCAPS have succeeded to produce the actions that lead to the goal state. We can analyze the experiment results using the Lyapunov's theorem.

In the author's control problem, the vector $\{\mathbf{s}(t), u(t)\}$ corresponds to the state $\mathbf{x}(t)$ of (2.6). And, both the plant and the controller are treated as a single system corresponding to the system described by (2.6). This system has a goal state of $\mathbf{s} = \mathbf{0}$ and $u = 0$.

By its definition in (3.1), it is obvious that $P_{\mathbf{w}(t)}$ is a positive definite function. Now, suppose that its time derivative $\dot{P}_{\mathbf{w}(t)}$ is always negative. Then, as time increases the value of $P_{\mathbf{w}(t)}$ gets smaller and smaller, i.e., $P_{\mathbf{w}(t)} < P_{\mathbf{w}(t-\triangle t)}$. Finally, $P_{\mathbf{w}(t)}$ approaches to zero, and therefore both the state $\mathbf{s}(t)$ and the action $u(t)$ converge to zero. Hence, we can say that $P_{\mathbf{w}(t)}$ is a good evaluation function to find the best trajectory toward the goal state when it can be tuned to satisfy the Lyapunov stability conditions.

Based on the above explanation, it seems that both LCAPS and FCAPS produced good experiment results because they succeeded to make the approximated evaluation function $P_{\mathbf{w}(t)}$ satisfy the Lyapunov stability conditions (i.e., by making the controller parameters appropriate).

In the experiments, FCAPS is given 625 parameters of the fuzzy controller while LCAPS only 2 parameters of the linear state feedback controller. The simulation results presented in this chapter show that FCAPS is better than LCAPS. Thus, those results correspond to the fact that the more number of the controller parameters enables the controller to have more resources and resolution, i.e., it can keep producing appropriate actions, given seemingly same states.

CHAPTER 4

Adaptive Control with On-line Tuning of Evaluation Function

The effectiveness of a fixed approximated value function as an evaluation function has been empirically proved in the previous chapter. However, in general it is difficult or even impossible that we can find an appropriate evaluation function that can be used to measure a control performance of any adaptive control. This chapter presents TAC (Two-stage Adaptive Control), i.e., an adaptive control scheme that tunes a controller with a gradient method using an approximate evaluation function adjusted on-line.

4.1 General Architecture

Figure 4.1 decribes the architecture of TAC. In general, TAC is supposed to work with the assumption that the plant state is continuous everywhere in a state space and imprecise plant knowledge that enables the engineer to make a "good" guess of initial controller parameters is available. Approximation errors of the evaluation function are coped with through online adjustment. The adjustment of the approximate evaluation function need not to be done through many time-consuming trials and errors. Therefore, TAC is expected to save computation time in finding appropriate controller parameters.

General architecture of TAC as shown in Figure 4.1 can be considered as similar to the policy search approach of RL. The difference is that rather than learning the

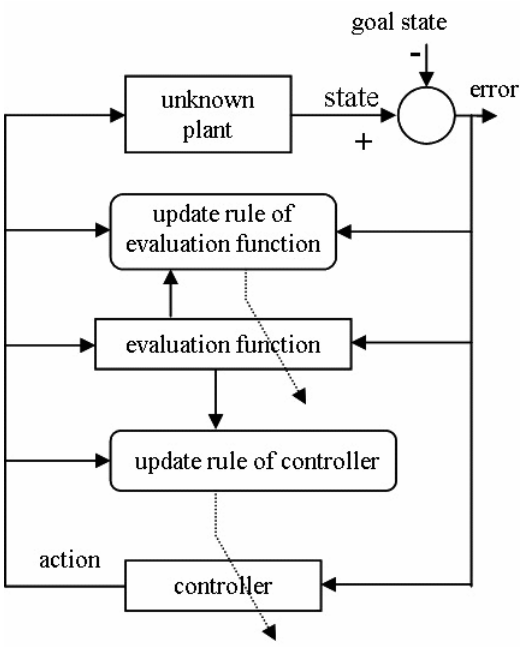


Figure 4.1: Two-stage Adaptive Control

value function the developed TAC simply uses an *approximate* evaluation function for tuning a controller with the gradient-based method.

The author proposes two types of the approximate evaluation function to be adjusted on-line. 1) Partially adjustable Evaluation Function (PEF). 2) Fully adjustable Evaluation Function (FEF). Based on the type of the evaluation function, the author classifies TAC in two types: 1) TAC/PEF that stands for TAC based on PEF. 2) TAF/FEF that stands for TAC based on FEF.

4.2 TAC based on Partially Adjustable Evaluation Function

4.2.1 Problem Statement

Consider an unknown nonlinear plant

$$\dot{\mathbf{s}}(t) = \mathbf{h}(\mathbf{s}(t), u(t)),\ \mathbf{h}(\mathbf{0}, 0) = \mathbf{0}, \tag{4.1}$$

where $\mathbf{s}(t) = [s_1(t)\ ...\ s_n(t)]^T \in R^n$ and $u(t) \in R$ respectively represent a state vector and a scalar input of the plant. R denotes a real number set. $\mathbf{s}(t)$ is assumed completely

available for measurement. Without loss of generality, a goal state to be achieved is defined at $\mathbf{s} = \mathbf{0}$. The plant (4.1) is controlled using a linear state feedback contoller

$$u(t) = \mathbf{w}^T(t)\mathbf{s}(t), \tag{4.2}$$

where $\mathbf{w}(t) = [w_1(t) \ ... \ w_n(t)]^T \in R^n$ is a parameter vector. For the above closed loop system, the author defines a Lyapunov function candidate

$$J(t) = \mathbf{s}^T(t)\mathbf{H}(t)\mathbf{s}(t), \tag{4.3}$$

where $\mathbf{H}(t)$ is a positive-definite matrix. This function J is used to represent the evaluation function for TAC.

4.2.2 Evaluation Function

Define a scalar function

$$\begin{aligned} L(t) = & \ \dot{J}(t) + \mathbf{s}^T(t)\mathbf{Q}\mathbf{s}(t) + \alpha u^2(t) \\ = & \ \mathbf{s}^T(t)\dot{\mathbf{H}}(t)\mathbf{s}(t) + \dot{\mathbf{s}}^T(t)\mathbf{H}(t)\mathbf{s}(t) + \mathbf{s}^T(t)\mathbf{H}(t)\dot{\mathbf{s}}(t) + \\ & \ \mathbf{s}^T(t)\dot{\mathbf{Q}}\mathbf{s}(t) + \alpha u^2(t), \end{aligned} \tag{4.4}$$

where α is a positive constant and $\mathbf{Q}$ is a positive definite constant matrix. Suppose that α, positive definite matrices $\mathbf{Q}$ and $\mathbf{H}(t)$, and a time derivative of $\mathbf{H}(t)$ (i.e., $\dot{\mathbf{H}}(t)$) are determined a priori so that $L(t) = 0$ is obtained, or equivalently,

$$\dot{J}(t) = -(\mathbf{s}^T(t)\mathbf{Q}\mathbf{s}(t) + \alpha u^2(t)). \tag{4.5}$$

When (4.5) is satisfied everywhere in the state space $\mathbf{s}$ for all t, then $J(t)$ is a Lyapunov function, i.e., $J(t) \to 0$ as $t \to \infty$ is guaranteed, and therefore, the controller $u(t)$ is said appropriate.

To achieve (4.5), $\mathbf{Q}$ and $\mathbf{H}(t)$ can not be chosen arbitrarily. This is because stable control of certain plants requires certain appropriate controllers. In addition, state convergences of those certain plants to the goal state do not necessarily imply $L(t) = 0$. In other words, state convergences of those certain plants are not necessarily associated with certain α, $\mathbf{Q}$ and $\mathbf{H}(t)$ that leads $L(t) = 0$. Thus, to force states of a certain plant to follow state trajetories implied by $L(t) = 0$, the author thinks there must exist matrices $\mathbf{Q}$ and $\mathbf{H}(t)$ associated with stable control of the plant (i.e., $\mathbf{Q}$ and $\mathbf{H}(t)$ lead $L(t)) = 0$ for all t).

For TAC/PEF to work for a given plant, it is assumed that the plant state convergence to the goal state can be associated with certain appropriate matrices $\mathbf{Q}$ and $\mathbf{H}(t)$ that lead $L(t) = 0$. Because it is difficult or even impossible to solve $L(t) = 0$ to derive those matrices $\mathbf{Q}$ and $\mathbf{H}(t)$ to be used then to solve the good controller, a goal of TAC/PEF is relaxed to find $u(t)$ that makes $L(t)$ as close to zero as possible.

By its definition in (4.4), $L(t)$ may approach zero by one or both of the following two ways:
1) both $\dot{J}(t)$ and $(\mathbf{s}^T(t)\mathbf{Q}\mathbf{s}(t) + \alpha u^2(t))$ approach zero at the same time,
2) $\dot{J}(t)$ (of negative values) converges to $-(\mathbf{s}^T(t)\mathbf{Q}\mathbf{s}(t) + \alpha u^2(t))$.
By the first way, the condition $(\mathbf{s}^T(t)\mathbf{Q}\mathbf{s}(t) + \alpha u^2(t)) \rightarrow 0$ is sufficient to imply that $\mathbf{s} \rightarrow \mathbf{0}$, regardless of whatever $\dot{J}(t)$. By the second way, as $J(t)$ is positive definite the negative $\dot{J}(t)$ implies that $J(t)$ is a Lyapunov function, i.e., it also implies $\mathbf{s} \rightarrow \mathbf{0}$. Thus, we have

$$\mathbf{s} \rightarrow \mathbf{0} \text{ as } L(t) \rightarrow 0. \tag{4.6}$$

Hence, the author considers $L(t)$ getting closer to zero as *direct* indication for good control performance. Due to a good property of $L(t)$ in 4.6, the author considers

$$P(t) = \frac{1}{4}L^2(t)$$

as an "good" evaluation function to be minimized in TAC/PEF.

4.2.3 Approximate evaluation function

When appropriate $\mathbf{Q}$, $\mathbf{H}(t)$, and $\dot{\mathbf{H}}(t)$ are available and $P(t)$ is computable, the TAC/PEF's main task will be only modifying the controller behavior (i.e., represented by $\mathbf{w}(t)$ and $\dot{\mathbf{w}}(t)$) to improve control performance as measured by the evaluation function $P(t)$. Particularly when the plant model is given, modification of the controller parameters that improves control performance as measured by $P(t)$ can be done off-line. The appropriate $\mathbf{Q}(t)$, $\mathbf{H}(t)$, and $\dot{\mathbf{H}}(t)$ are not known yet, and even if they are available, $P(t)$ is not computable because $\dot{\mathbf{s}}(t)$ is unknown.

To overcome those problems, the author uses an approximate version of $P(t)$ as the evaluation function. To approximate $P(t)$, the auhtor first assumes that $\dot{\mathbf{s}}(t)$ is continuous everywhere in the state space without drastic changes so that it can be estimated by

$$\hat{\dot{\mathbf{s}}}(t) = \frac{\mathbf{s}(t) - \mathbf{s}(t - \Delta t)}{\Delta t}, \tag{4.7}$$

where Δt is an elapsed time between time steps.

Then, the auhtor assumes that the plant can only be controlled stably (i.e., by making $L(t)$ get closer to zero) when appropriate $\mathbf{Q}$ can be determined a priori and a "good" guess of $\mathbf{H}(0)$ is available. The "good" guess of $\mathbf{H}(0)$ is required so that $\mathbf{H}(t)$ need not to be adjusted with drastic changes between time steps. The change rate of $\mathbf{H}(t)$ between time steps is equivalent to $\dot{\mathbf{H}}(t)$. In other words, the better estimation of $\mathbf{H}(0)$ should allow TAC to set $\dot{\mathbf{H}}(t)$ with smaller values that can make $L(t)$ get closer to zero. Based on such assumptions, the author considers that the term $\mathbf{s}^T(t)\dot{\mathbf{H}}(t)\mathbf{s}(t)$ of $L(t)$ (see (4.4)) is so small (i.e., compared to total amount of all other terms of $L(t)$) that it can be ignored. Then, $L(t)$ can be approximated with

$$\hat{L}(t) = \hat{\mathbf{s}}^T(t)\mathbf{H}(t)\mathbf{s}(t) + \mathbf{s}^T(t)\mathbf{H}(t)\hat{\mathbf{s}}(t) + \mathbf{s}^T(t)\mathbf{Q}\mathbf{s}(t) + \alpha u^2(t). \tag{4.8}$$

Using (4.8), the author proposes an approximate version of the "ideal" evaluation function $P(t)$ as follows

$$\hat{P}(t) = \frac{1}{4}\hat{L}^2(t). \tag{4.9}$$

The author calls the approximate evaluation function of (4.9) *Partially adjustable Evaluation Function* (PEF).

4.2.4 Algorithm

Using PEF as the evaluation function, TAC/PEF works according to the following steps:

Algorithm 1:

i) determine $\mathbf{Q}$ and α by trials-and-errors
ii) initialize $\mathbf{s}(0)$, $\mathbf{w}(0)$ and $\mathbf{H}(0)$
iii) set $t = 0$
iv) while $t <$ maximum time do
    1) get $\mathbf{s}(t)$ and execute $u(t) = \mathbf{w}^T(t)\mathbf{s}(t)$
    2) adjust $\mathbf{H}(t)$ using an update rule of $\mathbf{H}$
    3) update PEF using the adjusted $\mathbf{H}(t)$
    4) tune $\mathbf{w}(t)$ using an update rule of $\mathbf{w}$
    5) $t = t + \Delta t$
v) end.

4.2.4.1 Update rule of H

Let elements of $\mathbf{H}(t)$ be represented as $\{h_{ij}(t)\}$, $i, j = 1, ..., n$. The elements of $\mathbf{H}$ are adjusted to minimize (4.9) with the following gradient descent method:

$$\frac{dh_{ij}(t)}{dt} = -\eta_h \frac{\partial \hat{P}(t)}{\partial h_{ij}(t)} \tag{4.10}$$

where η_h is an adaptation rate. Using the chain rule, we can write

$$\frac{\partial \hat{P}(t)}{\partial h_{ij}(t)} = \frac{\partial \hat{P}(t)}{\partial \hat{L}(t)} \frac{\partial \hat{L}(t)}{\partial h_{ij}(t)}. \tag{4.11}$$

Using $\hat{L}(t)$ of (4.8), it can be shown that

$$\frac{\partial \hat{L}(t)}{\partial h_{ij}(t)} = \hat{\dot{s}}_i(t) s_j(t) + s_i(t) \hat{\dot{s}}_j(t). \tag{4.12}$$

From (4.12) and (4.11), (4.10) can be solved to obtain an adaptation law for tuning $h_{ij}(t)$ as follows:

$$h_{ij}(t + \Delta t) = h_{ij}(t) - \frac{\eta_h}{2} \hat{L}(t) (\hat{\dot{s}}_i(t) s_j(t) + s_i(t) \hat{\dot{s}}_j(t)) \Delta t. \tag{4.13}$$

The adaptation law (4.13) is required to keep $\mathbf{H}(t)$ positive definite. No update is done when $\hat{L}(t) = 0$.

Tuning $\mathbf{H}$ with (4.13) only might make $\mathbf{H}$ non-positive definite. This undesired condition will make $\hat{L}(t)$ step away from zero, and therefore, the control objective becomes more difficult to achieve. To overcome this problem, the author proposes a heuristic approach for tuning $\mathbf{H}(t)$ as follows:

Algorithm 2:

i) $\mathbf{T} \leftarrow \mathbf{H}(t)$; $\mathbf{H}(t)$ is copied to $\mathbf{T} = \{t_{ij}\}$

ii) FOR $i, j = 1, .., n$:

$$t_{ij} \leftarrow t_{ij} - \frac{\eta_h}{2} \hat{L}(t) (\hat{\dot{s}}_i(t) s_j(t) + s_i(t) \hat{\dot{s}}_j(t)) \Delta t$$

iii) IF $\mathbf{T}$ is positive definite:
THEN $\mathbf{H}(t + \Delta t) \leftarrow \mathbf{T}$
ELSE $\mathbf{H}(t + \Delta t) \leftarrow \mathbf{H}(t)$.

Given an initial positive definite $\mathbf{H}(0)$, **Algorithm 2** guarantees that a resultant $\mathbf{H}(t)$ is positive definite. The author uses Sylvester's criterion [38] to determine whether $\mathbf{H}$ is positive definite.

4.2.4.2 Update Rule of w

The controller parameter vector $\mathbf{w}$ is adjusted with the following gradient descent method:

$$\frac{dw_i(t)}{dt} = -\eta_w \frac{\partial \hat{P}(t)}{\partial w_i(t)} \tag{4.14}$$

where η_w is an adaptation rate. In (4.14), $\hat{P}(t)$ is computed using $\mathbf{H}(t)$ adjusted with **Algorithm 2**.

Using the chain rule, we obtain

$$\frac{\partial \hat{P}(t)}{\partial w_i(t)} = \frac{1}{2}\hat{L}(t)\frac{\partial \hat{L}(t)}{\partial u(t)}\frac{\partial u(t)}{\partial w_i(t)}. \tag{4.15}$$

We have that

$$\frac{\partial \mathbf{s}(t)}{\partial u(t)} = \mathbf{0},\ \frac{\partial \mathbf{H}(t)}{\partial u(t)} = \mathbf{0},\ \text{and}\ \frac{\partial \mathbf{Q}}{\partial u(t)} = \mathbf{0}$$

that reduces a computation of $\partial \hat{L}(t)/\partial u(t)$ to

$$\frac{\partial \hat{L}(t)}{\partial u(t)} = \Big(\frac{\partial \hat{\dot{\mathbf{s}}}(t)}{\partial u(t)}\Big)^T \mathbf{H}(t)\mathbf{s}(t) + \mathbf{s}^T(t)\mathbf{H}(t)\frac{\partial \hat{\dot{\mathbf{s}}}(t)}{\partial u(t)} + 2\alpha u(t). \tag{4.16}$$

In (4.16), $\partial \hat{\dot{\mathbf{s}}}(t)/\partial u(t)$ is an estimation of $\partial \dot{\mathbf{s}}(t)/\partial u(t)$, i.e., how large the change rate of $\dot{\mathbf{s}}(t)$ (*not* $\mathbf{s}(t)$) due to the change of $u(t)$ between time steps. $\partial \dot{\mathbf{s}}(t)/\partial u(t)$ is not necessarily proportional to the change rate of $\mathbf{s}(t)$. This implies that drastic changes of $\mathbf{s}(t)$ do not necessarily imply drastic changes on $\partial \dot{\mathbf{s}}(t)/\partial u(t)$. The changes of $\partial \dot{\mathbf{s}}(t)/\partial u(t)$ depend on the plant dynamics.

However, as the plant dynamics is assumed unknown, the approximation of $\partial \dot{\mathbf{s}}(t)/\partial u(t)$ remains difficult to solve. To cope with this problem, the author imposes an additional restriction upon TAC/PEF to work. TAC/PEF is supposed to work for the control problem in which any change of $u(t)$ will not cause drastic changes of $\dot{\mathbf{s}}(t)$, i.e., the ratio between $\Delta \dot{\mathbf{s}}(t)$ and $\Delta u(t)$ is so small that it allows us to ignore

$$\Big(\frac{\partial \hat{\dot{\mathbf{s}}}(t)}{\partial u(t)}\Big)^T \mathbf{H}(t)\mathbf{s}(t)\ \text{ and }\ \mathbf{s}^T(t)\mathbf{H}(t)\frac{\partial \hat{\dot{\mathbf{s}}}(t)}{\partial u(t)}$$

in (4.16). When this assumptions holds, we obtain

$$\frac{\partial \hat{L}(t)}{\partial u(t)} \approx 2\alpha u(t). \tag{4.17}$$

From (4.14)-(4.17), an update rule the controller parameters is obtained as follows:

$$w_i(t + \Delta t) = w_i(t) - \eta_w \alpha \hat{L}(t) u(t) \frac{\partial u(t)}{\partial w_i(t)} \Delta t. \tag{4.18}$$

The update rule of (4.18) updates w_i as long as $\hat{L}$, u, and $\partial u/\partial w_i$ have non-zero values. On the contrary, although u and $\partial u/\partial w_i$ have non-zero values, the update rule of (4.14) updates nothing when $\hat{L}(t) = 0$. In general, (4.14) is of a generic form for a class of adaptation laws which can be exactly derived as long as a derivative of u with respect to its parameters is given. For the linear state feedback controller considered in TAC/PEF, $\partial u(t)/\partial w_i(t) = s_i(t)$.

4.2.5 Experimental Results

To evaluate its effectiveness, TAC/PEF is applied to a cart-pole balancing problem. A physical description of the cart-pole plant is described in Fig. 3.3. Unlike the pole balancing problem considered in the previous chapter, The cart-pole balancing problem is similar to the pole balancing problem except that the cart must also be maintained at a center or a desired position. Simulations in this section use the same dynamic equation of the cart-pole plant as given in (3.12).

In the experiments, the author sets the cart-pole plant parameters: $g = 9.81$ ms^{-2}, $m_c = 1.0$ kg, $m = 0.1$ kg, and $l = 0.5$ m. The above cart-pole plant dynamics were then simulated using the 4th-order Runge-Kutta method with a time step of $\triangle t = 10$ ms where the controller output is limited within the range of $[-20, 20]$ N. The simulation was stopped when $|\theta| > 90^o$.

To control the above cart-pole plant using TAC/PEF, the state vector of the plant is chosen as $\mathbf{s}(t) = [s_1(t)\ s_2(t)\ s_3(t)\ s_4(t)]^T$ and assumed completely observable. In contrast, two elements of $\dot{\mathbf{s}}(t)$, i.e., $\dot{s}_2(t)$ and $\dot{s}_4(t)$, are assumed to be unobservable. Both unknown elements are estimated as $\dot{s}_2(t) \approx \hat{\dot{s}}_2(t) = (s_2(t) - s_2(t - \triangle t))/\triangle t$ and $\dot{s}_4(t) \approx \hat{\dot{s}}_4(t) = (s_4(t) - s_4(t - \triangle t))/\triangle t$. Together with $\dot{s}_1(t) = s_2(t)$ and $\dot{s}_3(t) = s_4(t)$, both estimated elements are then collected into an estimated vector of $\dot{\mathbf{s}}(t)$, i.e., $\hat{\dot{\mathbf{s}}}(t) =$

$[s_2(t)\ \hat{\dot{s}}_2(t)\ s_4(t)\ \hat{\dot{s}}_4(t)]^T$. Note that, hence, we have

$$\frac{\partial \hat{\dot{\mathbf{s}}}(t)}{\partial u(t)} = \left[0\ \ \frac{\partial \hat{\dot{s}}_2(t)}{\partial u(t)}\ \ 0\ \ \frac{\partial \hat{\dot{s}}_4(t)}{\partial u(t)}\right]^T,$$

which supports the aforementioned restriction, i.e., $\partial \hat{\dot{\mathbf{s}}}(t)/\partial u(t)$ is ignored in TAC/PEF.

The author carried out experiments comparing three types of control. 1) *Non-Adaptive Control* (NAC) where $\mathbf{w}$ is kept constant. 2) TAC/H where $\mathbf{w}$ is tuned on-line based on PEF but $\mathbf{H}$ of PEF is kept constant. 3) TAC/PEF. The adaptation rates are chosen as $\eta_w = 2.0$ and $\eta_h = 0.0004$, an initial time derivative of the plant state as $\hat{\dot{\mathbf{s}}}(0) = \mathbf{0}$, and an initial matrix $\mathbf{H}$ as $\mathbf{H}(0) = diag(1, 1, 1, 1)$. After some trial-and-error, the author set $\alpha = 0.001$ and $\mathbf{Q} = diag(1, 1, 1, 1)$.

4.2.5.1 Initialization of Controller Parameters

Despite unknown nature of the cart-pole plant model, it is possible for TAC to use imprecise knowledge of the plant behavior for initialization of the controller parameters.

Based on Fig. 3.3, positive (negative) pole angle is clockwise (counterclockwise), the positive (negative) cart position is in the right (left) side of the origin, and the positive (negative) force u is toward the right (left). Intuitively, we can assume that when the pole is not balanced and the cart is not centered, the control of the pole angle should be prioritized over that of the cart position.

When the angle and angular velocity of the pole are positive, a positive force should be applied to the cart. Thereby, the pole rotates counterclockwise and the cart moves toward the right. When the angle and angular velocity of the pole are negative, a negative force should be applied to the cart. Consequently, the pole rotates clockwise and the cart moves toward the left.

After the pole is nearly upright while its angular velocity decreases, the control of the cart position should be strengthened. When the position and velocity of the cart are positive, the cart should be moved slightly rightward by applying small positive force, causing the pole to rotate counterclockwise. In such a situation, the pole rotation must reach a negative angle such that it can trigger the control of the pole angle to produce a larger negative force, causing the cart to move leftward toward the origin while the pole rotates clockwise toward the upright position. In contrast, when the position and velocity of the cart are negative, the cart should be moved slightly leftward by applying a small negative force, thereby causing the pole to rotate clockwise. In such a situation, the pole rotation must reach a positive angle such that it can trigger the control of the

pole angle to produce a larger positive force, causing the cart to move rightward toward the origin, while the pole rotates counterclockwise toward the upright position.

The above description suggests that the parameters of the controller should be set to positive values according to importance degrees of the state variables. Unfortunately, precise importance degrees of the state variables are unknown. Nevertheless, we can infer that the pole angle has the highest importance degree among other state variables. Based on such imprecise knowledge, the controller parameters are initialized as $\mathbf{w}(0) = [50\ 5\ 5\ 5]^T$.

4.2.5.2 Simulation Results

Figures 4.2 and 4.3 show the simulation results for the initial pole angle $\theta(0) = 25$ deg at two opposite initial cart positions, i.e., $x(0) = -0.8$ m and $x(0) = 0.8$ m. Intuitively, when both initial pole angle and cart position have the same sign, the control of the cart-pole plant is more difficult than when both have opposite signs. That intuition is proven by the fact that NAC succeeded to balance the pole and center the cart when $x(0) = -0.8$ m but failed when $x(0) = 0.8$ m despite its actions reach the lower and upper limits of the actions (i.e., [-20,20] N). However, given the same initial conditions as in NAC, both TAC/H and TAC/PEF succeeded to balance the pole and center the cart. For the successful cases of all the controllers as shown in Fig. 4.2, NAC has much worse performance than both TAC/H and TAC/PEF in all terms: the pole angle, the cart position, and the action. It seems that there is no significant difference of performances of TAC/H and TAC/PEF.

Figures 4.4 and 4.5 show the simulation results for the initial pole angles larger than $\theta(0) = 25$ deg. With the initial pole angles of 40 deg and 45 deg, the cart-pole plant behaves highly nonlinear where the pole tends to fall down easily and quickly if no enough action is applied. To solve such initial condition problems, NAC produced the maximum actions that forced the pole to become upright very quickly. But, it seems NAC kept the maximum actions too long which made the pole continue to rotate fast and therefore it felt down counterclockwise eventually.

Given the same initial control problems, TAC/H also produces the maximum positive actions. The pole then rotates counterclockwise quickly. When the pole is near upright (i.e., around 0 deg), the control of the pole angle reduces around "zero". But, at the same time the controls of other state variables (i.e., the pole angular velocity, the cart position and its velocity), which produce negative actions, cause the pole to slowly rotate counterclockwise until around -20 deg. The TAC/H's actions that cause

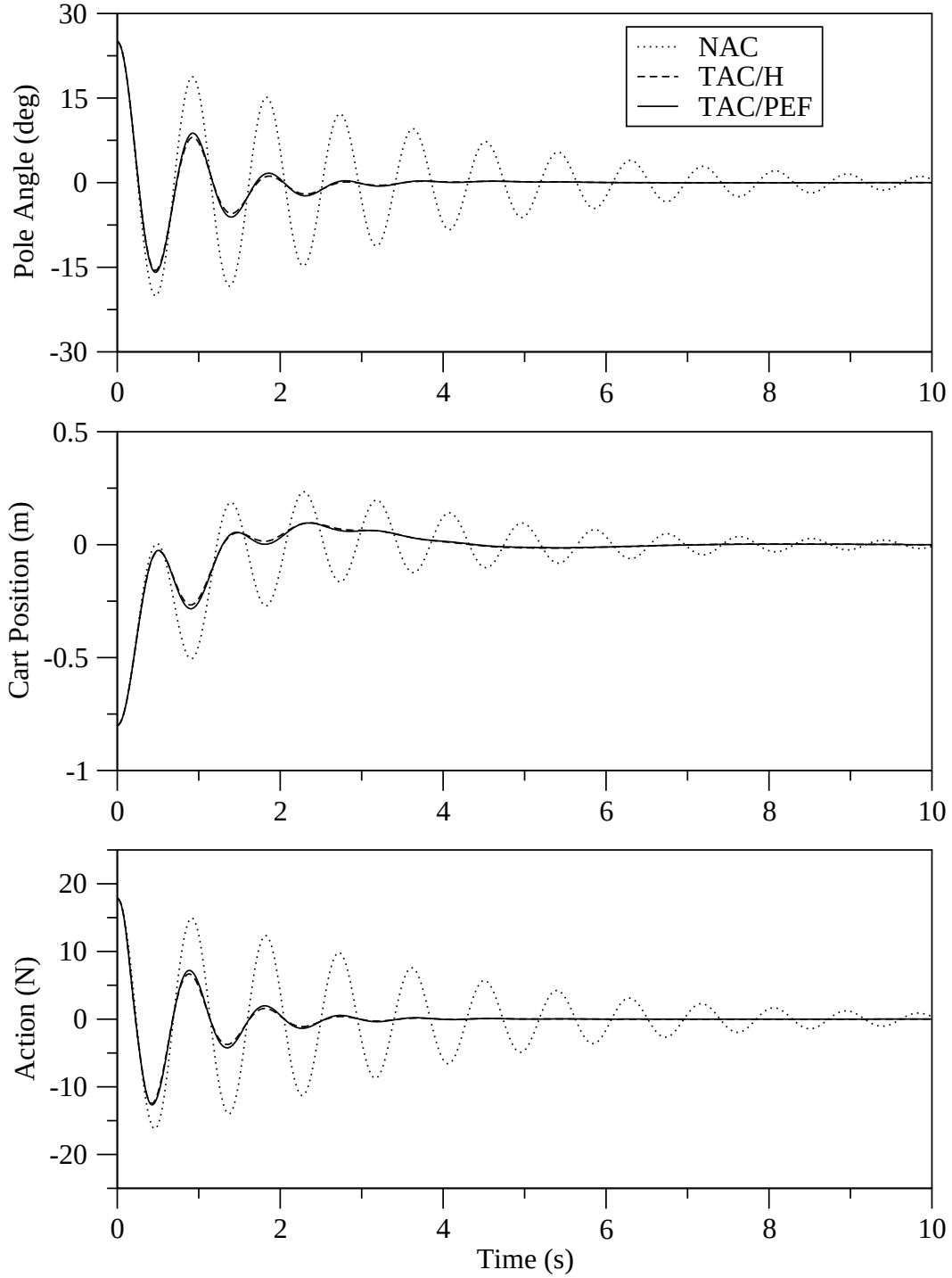


Figure 4.2: TAC based on PEF used to solve cart-pole balancing problem with $\theta(0) = 25$ deg and $x(0) = -0.8$ m.

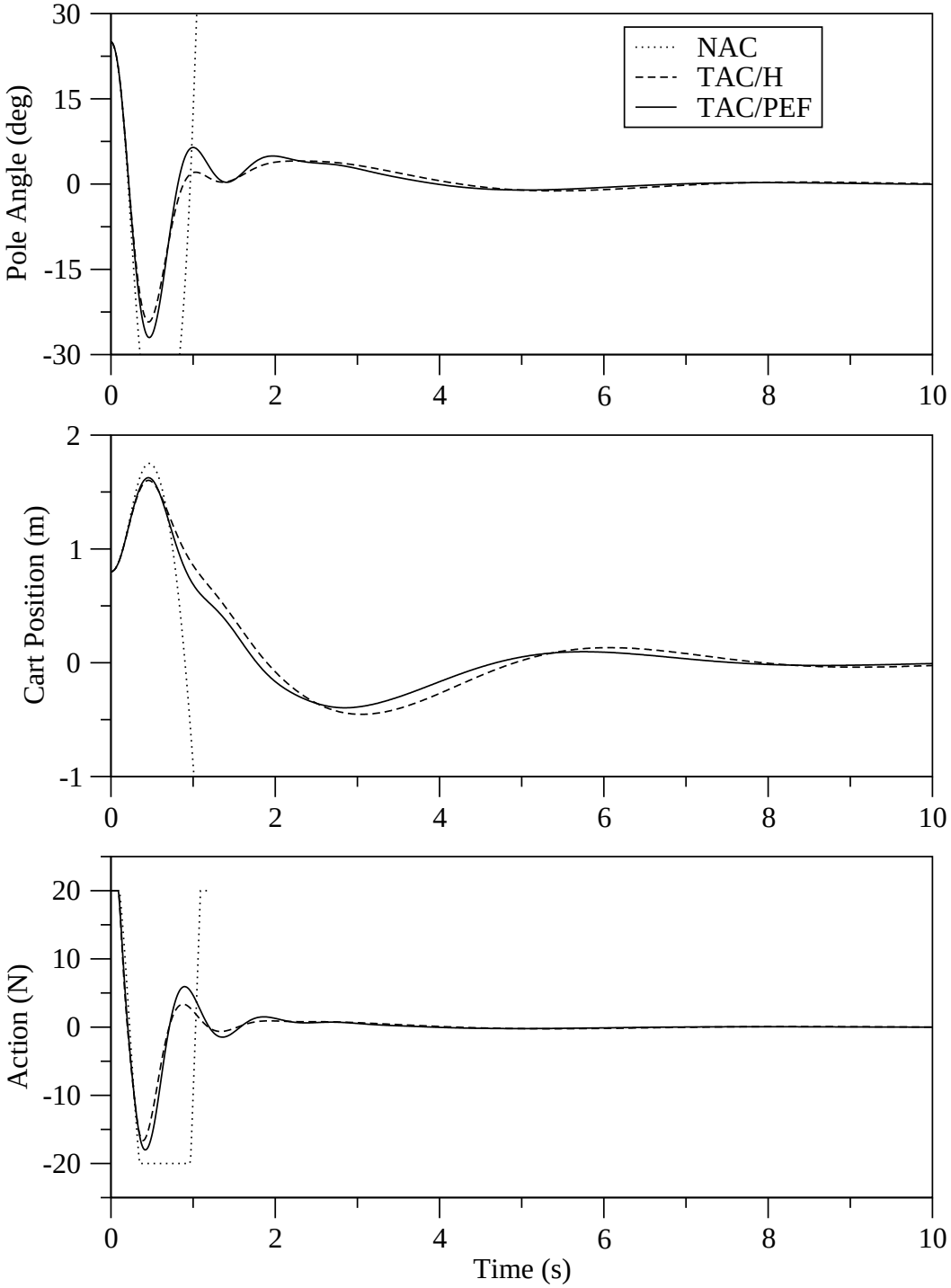


Figure 4.3: TAC based on PEF used to solve cart-pole balancing problem with $\theta(0) = 25$ deg and $x(0) = 0.8$ m.

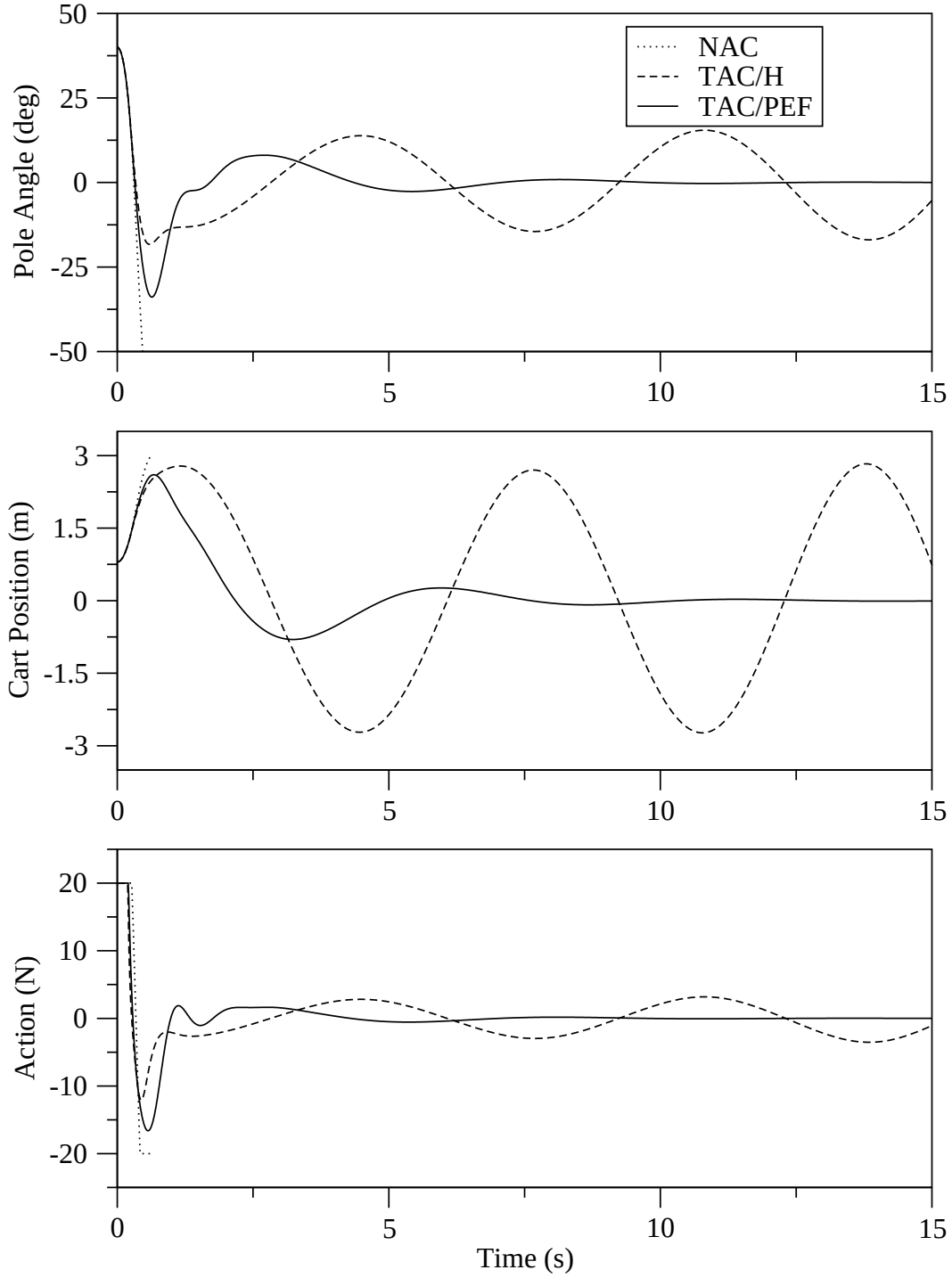


Figure 4.4: TAC based on PEF used to solve cart-pole balancing problem with $\theta(0) = 40$ deg and $x(0) = 0.8$ m.

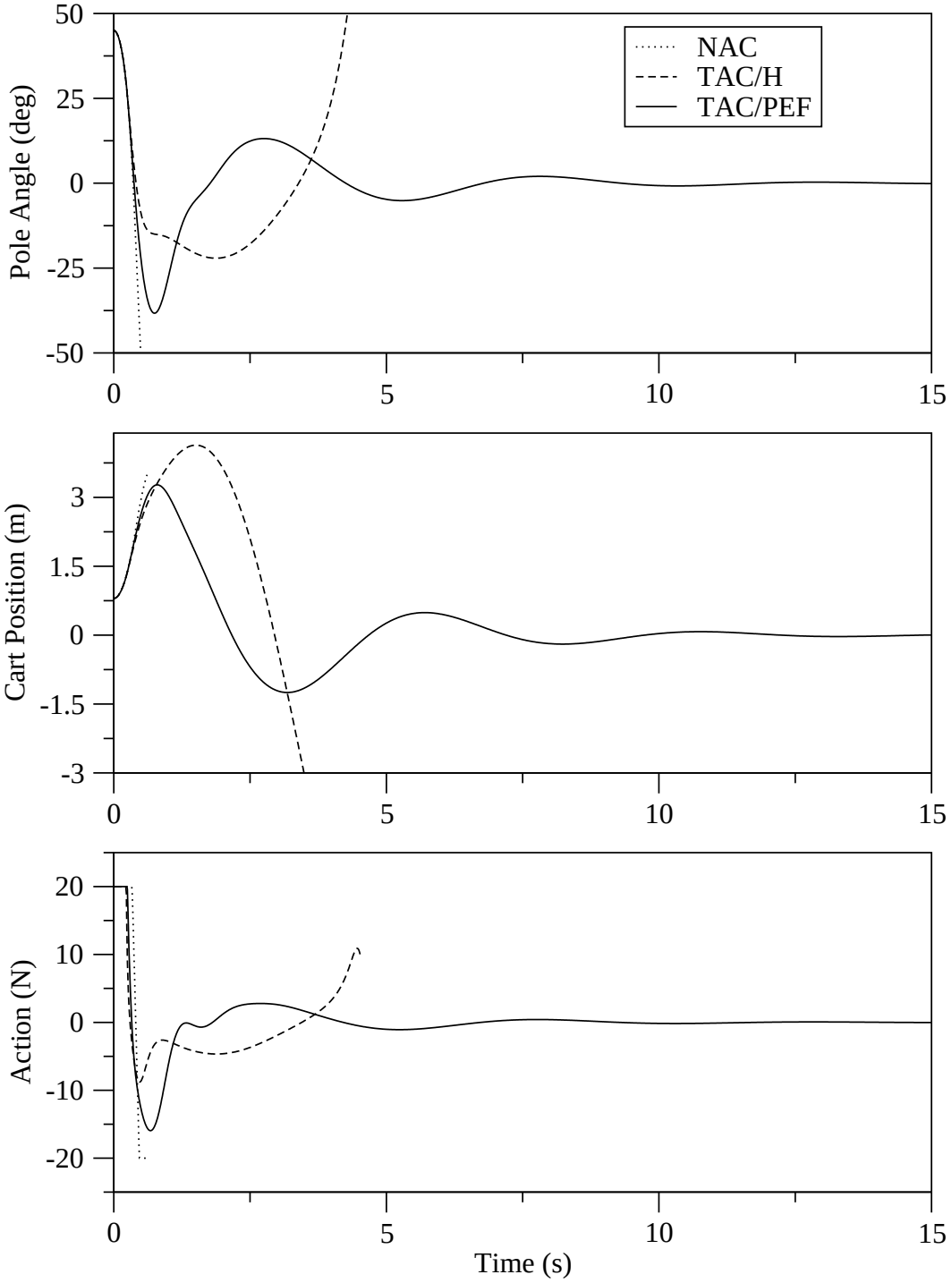


Figure 4.5: TAC based on PEF used to solve cart-pole balancing problem with $\theta(0) = 45$ deg and $x(0) = 0.8$ m.

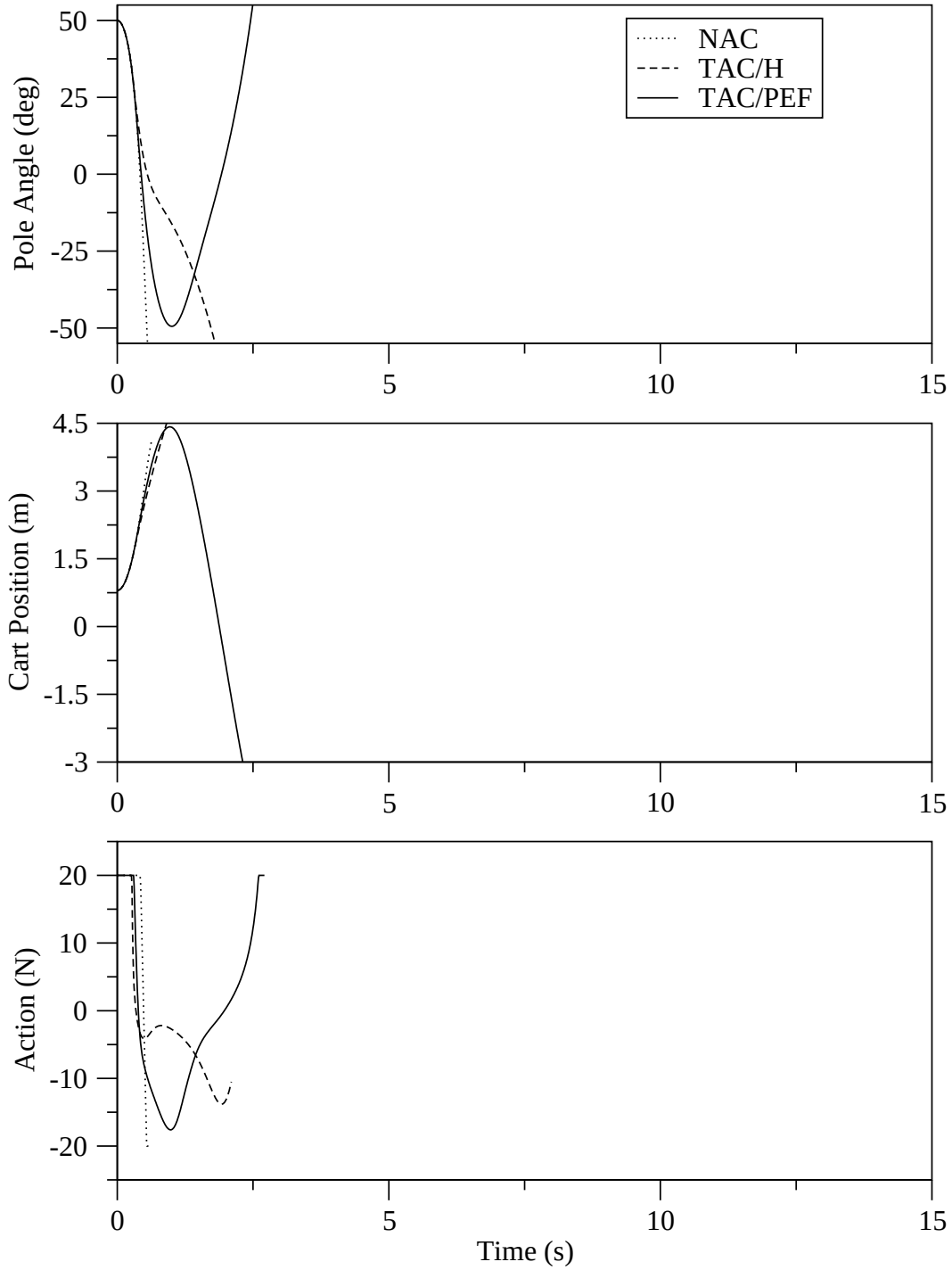


Figure 4.6: TAC based on PEF used to solve cart-pole balancing problem with $\theta(0) = 50$ deg and $x(0) = 0.8$ m.

all these occur change from maximum positive to negative (i.e., until around -12 N, the lowest TAC/H's action) and then approach zero. After applications of these actions, the pole turns back to rotate clockwise to approach 0 deg. Afterward, however, in Figure 4.4 the TAC/H's actions make the pole keep rotating clockwise and counterclockwise with its angle being larger and larger, while the cart keeps moving to the left and the right with the distance being farther and farther from the center. For a larger initial pole angle as shown in Figure 4.5 the TAC/H's actions failed soon at control of the cart-pole plant.

Comparing their actions, it is clearly seen that NAC and TAC/H behave "oppositely" but either of both leads bad performances. Both figures show that the TAC/PEF's actions behave "in between" those of NAC and TAC/H. Given the initial pole angles of 40 and 45, TAC/PEF also produces the maximum positive actions. The pole then rotates counterclockwise quickly. But, the TAC/PEF's actions are not too large as those of NAC and not too small as those of TAC/H, which make the pole angle gets the lowest value of around -40 deg. Thus, TAC/PEF finally succeeded to balance the pole while maintaining the cart at the center. However, TAC/PEF failed at control for the larger initial pole angle 45 deg as shown in Figure 4.6.

4.2.5.3 Defects of TAC based on PEF

Despite its successful application to solve the cart-pole balancing problem, the author notices some limitations of TAC based on PEF as follows:

1) The appropriateness of PEF is heavily dependent on many parameters, i.e., $\mathbf{Q}$ and $\mathbf{H}$, which are seem. In other words, it implies a heavy burden of parameter initialization to enable TAC/PEF to work well.
2) It is unclear what the physical meaning of $\mathbf{Q}$ and $\mathbf{H}$. This leads no clear way of how to better set $\mathbf{Q}$ and $\mathbf{H}$.
3) An appropriate PEF requires a positive definite J that converges to zero when the plant state converges to the goal state. The appropriate J requires an appropriate positive definite $\mathbf{H}$. This is exactly a problem of finding a Lyapunov function J, which is difficul to solve for almost all complex control problems.

4.3 TAC based on Fully Adjustable Evaluation Function

Those limitations of PEF motivate the author to develope another type of the evaluation function for TAC which is discussed in this section. The author considers the

same unknown nonlinear plant of (4.1) to be controlled with the state linear feedback controller (4.2).

4.3.1 Evaluation Function

Define a scalar function

$$J(t) = u^2(t) + \mathbf{s}^T(t)\mathbf{Q}(t)\mathbf{s}(t), \tag{4.19}$$

where $\mathbf{Q}(t) \in R^{n\times n}$ is a $n \times n$ matrix whose i,j-th element is denoted with $q_{ij}(t),\ i,j = 1,...,n$. Or equivalently,

$$J(t) = \quad \mathbf{s}^T(t)\mathbf{H}(t)\mathbf{s}(t),$$

where

$$\mathbf{H}(t) = \mathbf{w}(t)\mathbf{w}^T(t) + \mathbf{Q}(t). \tag{4.20}$$

An i,j-th element of $\mathbf{H}(t)$ is denoted with

$$h_{ij}(t) = w_i(t)w_j(t) + q_{ij}(t),\ i,j = 1,...,n.$$

Both matrices $\mathbf{Q}(t)$ and $\mathbf{H}(t)$ are non-singular but need not to be positive definite.

The author assumes that an approximated value function in TAC can be defined as follows

$$P(t) = \frac{1}{2}L(t)^2, \tag{4.21}$$

where

$$\begin{aligned} L(t) = &\quad \dot{J}(t) + kJ(t) \\ = &\quad \mathbf{s}^T(t)\dot{\mathbf{H}}(t)\mathbf{s}(t) + \dot{\mathbf{s}}^T(t)\mathbf{H}(t)\mathbf{s}(t) + \mathbf{s}^T(t)\mathbf{H}(t)\dot{\mathbf{s}}(t) + k\mathbf{s}^T(t)\mathbf{H}(t)\mathbf{s}(t), \end{aligned} \tag{4.22}$$

$$\dot{\mathbf{H}}(t) = 2\mathbf{w}(t)\dot{\mathbf{w}}^T(t) + \dot{\mathbf{Q}}(t),$$

and $k > 0$. When $P(t) = 0$ is obtained we obtain $L(t) = 0$, or equivalently,

$$\dot{J}(t) = -kJ(t), \tag{4.23}$$

which implies $J(t) \to 0$ as $t \to \infty$ with a time constant of $1/k$. When $J(t)$ is positive definite, then $J(t)$ is a Lyapunov function. Otherwise, $J(t)$ is an oscillating but approaching-to-zero function. The author supposes k as a design parameter that represents a desired speed of convergence of $J(t)$ to zero, i.e., setting larger k means we want $J(t)$ to converge faster, and vice versa. Lower and upper bounds of k that can

be chosen depend on the plant behavior, i.e., how quickly the plant state can be made converge to the goal state at which the action should be zero, corresponding to how quickly $J(t)$ can be made converge to zero.

In TAC, for $P(t)$ of (4.21) to be a good approximation of the value function, $P(t)$ is required to have a property as follows: $P(t) \to 0$ implies $\mathbf{s}(t) = \{s_i(t)\}$ gets closer to the goal state along the trajectories implied by the following dynamics:

$$\dot{s}_i(t) = -k_i s_i(t) \; ; k_i > 0, \; i = 1, \cdots, n. \tag{4.24}$$

From (4.24), it follows that $\mathbf{s} \to \mathbf{0}$ as $t \to \infty$. To obtain such a property of $P(t)$, $\mathbf{H}(t)$ of (4.20) can not be chosen arbitrarily. This is because $P(t) \to 0$ (or, $L(t) \to 0$) does not always indicate $\mathbf{s}(t) \to \mathbf{0}$ when (4.24) does not apply. Thus, the dynamics of (4.24) has a role as an *evaluator* for the appropriateness of $P(t)$.

The author supposes $P(t)$ with the above property has a role similar to the role of the state-action-value function of RL. A smaller value of $P(t)$ represents a better performance of an action $u(t)$ executed at $\mathbf{s}(t)$, corresponding to a higher value of state-action $\{\mathbf{s}(t), u(t)\}$ in RL. $P(t)$ is not known until the result of applying $u(t)$ to the plant (i.e., the next state), is obtained. In $P(t)$, the next state is included in $\dot{\mathbf{s}}(t)$.

In RL, an optimal state-action-value function tells the controller the best action to take at any given state so that a maximum reward will be received in the long run. An optimal value of the state-action itself is equal to the reward received at the next time step plus a properly discounted value of the next state-optimal actions. Therefore, taking actions deduced from the optimal state-action value function will lead to the goal state at which the maximum total amount of rewards will be obtained [7, 42]. In TACF, $P(t)$ is used as the evaluation function at the next time step to evaluate the action $u(t)$ executed at $\mathbf{s}(t)$ and then update the controler parameters. Thus, an action in TAC/FEF that should be taken at a given state $\mathbf{s}(t)$ is the action $u(t)$ by which $P(t)$ gets smallest.

4.3.2 Approximate Evaluation Function

When appropriate $\mathbf{Q}(t)$ and $\dot{\mathbf{Q}}(t)$ are available and $P(t)$ is computable, the TAC's main task will be only modifying the controller behavior (i.e., represented by $\mathbf{w}(t)$ and $\dot{\mathbf{w}}(t)$) to improve control performance as measured by the evaluation function $P(t)$. Particularly when the plant model is given, modification of the controller parameters that improves control performance as measured by $P(t)$ can be done off-line. However,

the appropriate $\mathbf{Q}(t)$ and $\dot{\mathbf{Q}}(t)$ are not available yet, and even if they are available, $P(t)$ is not computable because $\dot{\mathbf{s}}(t)$ is unknown.

For TAC, the author defines an approximated version of $P(t)$. To approximate $P(t)$, the author first assumes that $\dot{\mathbf{s}}(t)$ is continuous everywhere in the state space so that it can be estimated by the following forward difference equation

$$\hat{\dot{\mathbf{s}}}(t) = \frac{\mathbf{s}(t+\Delta t) - \mathbf{s}(t)}{\Delta t}. \tag{4.25}$$

Estimation of $\dot{\mathbf{s}}(t)$ in (4.25) can be done at time $t + \Delta t$ at which $\mathbf{s}(t + \Delta t)$ is known. Then, the author assumes that the plant can only be controlled stably when we can make "good" guess of $\mathbf{H}(0)$ so that TAC does not need to adjust $\mathbf{H}(t)$ with drastic changes between time steps. The change rate of $\mathbf{H}(t)$ between time steps is equivalent to $\dot{\mathbf{H}}(t)$. In other words, the author assumes that the better estimation of $\mathbf{H}(0)$ should allow TAC to set $\dot{\mathbf{H}}(t)$ with smaller values that can make $L(t)$ get closer to zero. Based on such assumptions, the author considers that the term $\mathbf{s}^T(t)\dot{\mathbf{H}}(t)\mathbf{s}(t)$ of $L(t)$ (see (4.22)) is so small (i.e., compared to total amount of all other terms of $L(t)$)) that it can be ignored. Then, the author approximates $L(t)$ with

$$\hat{L}(t) = \quad \hat{\dot{\mathbf{s}}}^T(t)\mathbf{H}(t)\mathbf{s}(t) + \mathbf{s}^T(t)\mathbf{H}(t)\hat{\dot{\mathbf{s}}}(t) + k\mathbf{s}^T(t)\mathbf{H}(t)\mathbf{s}(t). \tag{4.26}$$

Using (4.26), the author approximates $P(t)$ with

$$\hat{P}(t) = \frac{1}{2}\hat{L}(t)^2. \tag{4.27}$$

and calls it *Fully adjustable Evaluation Function* (FEF). Using that FEF, TAC is referred to as TAC/FEF.

4.3.3 Algorithm

TAC/FEF works according to the following steps:

Algorithm 3:

i) set k and initialize $\mathbf{w}(0)$, $\mathbf{Q}(0)$, and $\mathbf{s}(0)$.
ii) execute $u(0) = \mathbf{w}(0)^T\mathbf{s}(0)$
iii) set $t = \Delta t$
iv) while $t <$ maximum time do
    1) get $\mathbf{s}(t)$

2) adjust $\mathbf{Q}(t)$ using update rule of $\mathbf{Q}$
3) update FEF using the adjusted $\mathbf{Q}(t)$
4) tune $\mathbf{w}(t)$ using update rule of $\mathbf{w}$
5) execute $u(t) = \mathbf{w}^T(t)\mathbf{s}(t)$ determined using the tuned $\mathbf{w}(t)$
6) $t = t + \Delta t$

v) end.

4.3.3.1 Update rule of Q

Let denote $\tau = t - \Delta t$. An update rule of the elements of $\mathbf{Q}$ is derived using a gradient-descend method as follows:

$$\begin{aligned} q_{ij}(t) = & \quad q_{ij}(\tau) - \eta_q \frac{\partial \hat{P}(t)}{\partial q_{ij}(t)}\bigg|_{t=\tau} \Delta t \\ = & \quad q_{ij}(\tau) - \eta_q \hat{L}(\tau) \frac{\partial \hat{L}(t)}{\partial q_{ij}(t)}\bigg|_{t=\tau} \Delta t, \end{aligned} \tag{4.28}$$

where $\eta_q > 0$ is an adaptation rate. Because

$$\frac{\partial \mathbf{H}(t)}{\partial q_{ij}(t)} = \frac{\partial (\mathbf{w}(t)\mathbf{w}^T(t) + \mathbf{Q}(t))}{\partial q_{ij}(t)} = \frac{\partial \mathbf{Q}(t)}{\partial q_{ij}(t)},$$

we obtain

$$\begin{aligned} \frac{\partial \hat{L}(t)}{\partial q_{ij}(t)} = & \quad \dot{\hat{\mathbf{s}}}(t) \frac{\partial \mathbf{H}(t)}{\partial q_{ij}(t)} \mathbf{s}(t) + \mathbf{s}^T(t) \frac{\partial \mathbf{H}(t)}{\partial q_{ij}(t)} \dot{\hat{\mathbf{s}}}(t) + k\mathbf{s}^T(t) \frac{\partial \mathbf{H}(t)}{\partial q_{ij}(t)} \mathbf{s}(t) \\ = & \quad \dot{\hat{\mathbf{s}}}(t) \frac{\partial \mathbf{Q}(t)}{\partial q_{ij}(t)} \mathbf{s}(t) + \mathbf{s}^T(t) \frac{\partial \mathbf{Q}(t)}{\partial q_{ij}(t)} \dot{\hat{\mathbf{s}}}(t) + k\mathbf{s}^T(t) \frac{\partial \mathbf{Q}(t)}{\partial q_{ij}(t)} \mathbf{s}(t) \\ = & \quad \sum_i^n \sum_j^n \left(\dot{\hat{s}}_i(t) s_j(t) + s_i(t) \dot{\hat{s}}_j(t) + k s_i(t) s_j(t) \right) \end{aligned}$$

so that

$$\frac{\partial \hat{L}(t)}{\partial q_{ij}(t)}\bigg|_{t=\tau} = \sum_i^n \sum_j^n \left(\dot{\hat{s}}_i(\tau) s_j(\tau) + s_i(\tau) \dot{\hat{s}}_j(\tau) + k s_i(\tau) s_j(\tau) \right).$$

Updated by the adaptation law of (4.28), $\mathbf{Q}(t)$ might make $\mathbf{H}(t) = \mathbf{w}(t)\mathbf{w}^T(t) + \mathbf{Q}(t)$ singular. On the other hand, the tuning of $\mathbf{Q}(t)$ is required to make $P(t)$ get improved while implying that (4.24) applies al the times. To solve these problems, a heuristic approach for tuning $\mathbf{Q}(t)$ is proposed as follows:

$$q_{ij}(t) \leftarrow q_{ij}(\tau) - l(t)\eta_q S(t)\left(\sum_{i}^{n}\sum_{j}^{n}\left(\hat{\dot{s}}_i(\tau)s_j(\tau) + s_i(\tau)\hat{\dot{s}}_j(\tau) + ks_i(\tau)s_j(\tau)\right)\right)\Delta t, \quad (4.29)$$

where $l(t)$ is a *"lie" detector* defined as follows:

$$l(t) = \begin{cases} 0, & \text{if } |\det(\mathbf{H}(\tau))| > \epsilon_H \text{ and } \operatorname{sign}(\hat{\dot{J}}(\tau)) \neq \operatorname{sign}(J(\tau)) \\ & \text{and } \operatorname{sign}(\hat{\dot{s}}_i(\tau)) \neq \operatorname{sign}(s_i(\tau)),\ (i = 1, \cdots, n), \\ 1, & \text{otherwise,} \end{cases} \quad (4.30)$$

and $S(t)$ is a *"stuck" detector* defined as follows:

$$S(t) = \begin{cases} \operatorname{sign}(L(\tau)), & \text{if } l(t) = 1 \text{ and } |L(\tau)| \leq \epsilon_L \\ L(\tau), & \text{otherwise,} \end{cases} \quad (4.31)$$

where ϵ_H and ϵ_L are small positive constants, and

$$\hat{\dot{J}}(\tau) = \hat{\dot{\mathbf{s}}}^T(\tau)\mathbf{H}(\tau)\mathbf{s}(\tau) + \mathbf{s}^T(\tau)\mathbf{H}(\tau)\hat{\dot{\mathbf{s}}}(\tau).$$

Given an initial nonsingular $\mathbf{H}(0) = \mathbf{w}(0)\mathbf{w}^T(0) + \mathbf{Q}(0)$, the heuristic update rule of (4.29) guarantees that the tuned $\mathbf{H}(t) = \mathbf{w}(t)\mathbf{w}^T(t) + \mathbf{Q}(t)$ is nonsingular. The singularity of $\mathbf{H}(t)$ is determined from its determinant, i.e., $\mathbf{H}(t)$ is considered as nonsingular if $\det(\mathbf{H}(t)) > \epsilon_H$, otherwise singular. The heuristic update rule (4.29) keeps $\mathbf{Q}(t)$ unchanged (i.e., by setting $l(t) = 0$) when $\mathbf{H}(\tau) = \mathbf{w}(\tau)\mathbf{w}^T(\tau) + \mathbf{Q}(\tau)$ is nonsingular and

$$\operatorname{sign}(\hat{\dot{J}}(\tau)) \neq \operatorname{sign}(J(\tau)) \quad (4.32)$$

and

$$\operatorname{sign}(\hat{\dot{s}}_i(\tau)) \neq \operatorname{sign}(s_i(\tau))\ (i = 1, \cdots, n). \quad (4.33)$$

The condition of (4.32) is derived from (4.24). Note that without loss of generality, whatever $k_i > 0$ in (4.24), the oppposite signs of $\dot{s}_i(t)$ and $s_i(t)$ for all the times are sufficient to imply that $\mathbf{s}(t)$ be lying on a stable trajectory implied by (4.24). Thus, when the condition of (4.32) applies for all the times, $\mathbf{s}(t)$ converges to the goal state. Similarly, the opposite signs of $J(t)$ and $\dot{J}(t)$ are sufficient to imply that $J(t)$ converges to zero (or equivalently, FEF converges to zero). In the following explanation, by the stable trajectory the author means the trajectory implied by the condition of (4.33).

The intuitive explanation for the role of the "lie" detector of (4.30) is as follows: if FEF is decreasing but a current state is not on the stable trajectory, meaning that FEF is telling a "lie" that a current control performance is good, but actually it is bad. Similarly, if FEF is increasing but the current state is on the stable stable trajectory, meaning FEF is also telling the "lie" that a current control performance is "bad", but actually it is good. When either of these occurs, $\mathbf{Q}$ of FEF is considered as "bad" and needs to be tuned (i.e., $l(t)$ should be set to 1). Thus, $l(t) = 1$ corresponds to FEF telling the "lie". In contrast, if FEF is decreasing and the current state is on the stable trajectory, meaning that FEF is telling a "truth" that the current control performance is really good. Similarly, if FEF is increasing and the current state is not on the stable trajectory, meaning that FEF is also telling the "truth" that the current control performance is really bad. When either of these occurs, $\mathbf{Q}$ of FEF is considered as "good" and it is reasonable to keep it unchanged (i.e., $l(t)$ should be set to 0), except when $\mathbf{Q}$ is singular. Thus, $l(t) = 0$ corresponds to FEF telling the "truth". In the heuristic update rule (4.29), $\mathbf{Q}$ is kept unchanged (i.e., by setting $l(t) = 0$) only when $\mathbf{Q}$ is nonsingular, FEF is decreasing and the current state is on the stable trajectory. Note that although FEF is increasing and the current state is not on the stable trajectory, we cannot assure that the current state is really a bad state that never leads to better next states later. In other words, although the current state is not on the stable trajectory, it might ease the adaptive control to find much better states later. For this reason, although $\mathbf{Q}$ is nonsingular, FEF is increasing and the current state is not on the stable trajectory, the heuristic update rule of (4.29) tunes $\mathbf{Q}$ (by setting $l(t) = 1$).

As explained above, when $l(t) = 1$, then $\mathbf{Q}(t)$ needs be adjusted. However, there might be a condition at which even if $l(t) = 1$ but $L(t)$ is already zero. Using the original adaptation law of (4.28) for tuning $\mathbf{Q}(t)$, that condition implies that the tuning of $\mathbf{Q}(t)$ gets stuck, while $\mathbf{s}(t)$ diverges from the goal state. To prevent such a condition from happening, the author introduces the "stuck" detector $S(t)$ (see (4.31)) in the heuristic update rule of (4.29). Using the "stuck" detector, the update rule of (4.29) keeps on adjusting $\mathbf{Q}(t)$ when $|L(t)| < \epsilon_L$.

4.3.3.2 Update rule of w

An update rule of the elements of $\mathbf{w}(t)$ is derived using the gradient-descend method as follows:

$$\begin{aligned} w_i(t) = &\ w_i(\tau) - \eta_w \frac{\partial \hat{P}(t)}{\partial w_i(t)}\bigg|_{t=\tau} \Delta t \\ = &\ w_i(\tau) - \eta_w \hat{L}(\tau) \frac{\partial \hat{L}(t)}{\partial w_i(t)}\bigg|_{t=\tau} \Delta t, \end{aligned} \tag{4.34}$$

where $\eta_w > 0$ is an adaptation rate. In (4.34), $\hat{P}(t)$ is computed using $\mathbf{Q}(t)$ adjusted with the heuristic update rule of (4.29).

To compute $\partial \hat{L}(t)/\partial w_i(t)$ in (4.34), $\hat{L}(t)$ in (4.26) is first rewritten as

$$\begin{aligned} \hat{L}(t) = &\ \hat{\mathbf{s}}^T(t)\mathbf{H}(t)\mathbf{s}(t) + \mathbf{s}^T(t)\mathbf{H}(t)\hat{\mathbf{s}}(t) + k\mathbf{s}^T(t)\big(\mathbf{w}(t)\mathbf{w}^T(t) + \mathbf{Q}(t)\big)\mathbf{s}(t) \\ = &\ \hat{\mathbf{s}}^T(t)\mathbf{H}(t)\mathbf{s}(t) + \mathbf{s}^T(t)\mathbf{H}(t)\hat{\mathbf{s}}(t) + k\big(u^2(t) + \mathbf{s}^T(t)\mathbf{Q}(t)\mathbf{s}(t)\big). \end{aligned} \tag{4.35}$$

Using $\hat{L}(t)$ in (4.35), we can compute

$$\begin{aligned} \frac{\partial \hat{L}(t)}{\partial u(t)} = &\ \frac{\partial \hat{\mathbf{s}}^T(t)}{\partial u(t)}\mathbf{H}(t)\mathbf{s}(t) + \hat{\mathbf{s}}(t)\frac{\partial \mathbf{H}(t)}{\partial u(t)}\mathbf{s}(t) + \hat{\mathbf{s}}^T(t)\mathbf{H}(t)\frac{\partial \mathbf{s}(t)}{\partial u(t)} + \\ &\ \frac{\partial \mathbf{s}^T(t)}{\partial u(t)}\mathbf{H}(t)\hat{\mathbf{s}}(t) + \mathbf{s}^T(t)\frac{\partial \mathbf{H}(t)}{\partial u(t)}\hat{\mathbf{s}}(t) + \mathbf{s}^T(t)\mathbf{H}(t)\frac{\partial \hat{\mathbf{s}}(t)}{\partial u(t)} + \\ &\ k\left(2u(t) + \frac{\partial \mathbf{s}^T(t)}{\partial u(t)}\mathbf{Q}(t)\mathbf{s}(t) + \mathbf{s}^T(t)\frac{\partial \mathbf{Q}(t)}{\partial u(t)}\hat{\mathbf{s}}(t) + \mathbf{s}^T(t)\mathbf{Q}(t)\frac{\partial \mathbf{s}(t)}{\partial u(t)}\right). \end{aligned} \tag{4.36}$$

Because $\mathbf{s}(t)$ and $\mathbf{Q}(t)$ are independent of $u(t)$, we have that

$$\frac{\partial \mathbf{s}(t)}{\partial u(t)} = \mathbf{0},\ \frac{\partial \mathbf{Q}(t)}{\partial u(t)} = \mathbf{0},$$

which make $\partial \hat{L}(t)/\partial u(t)$ of (4.36) reduces to

$$\begin{aligned} \frac{\partial \hat{L}(t)}{\partial u(t)} = &\ \frac{\partial \hat{\mathbf{s}}^T(t)}{\partial u(t)}\mathbf{H}(t)\mathbf{s}(t) + \mathbf{s}^T(t)\mathbf{H}(t)\frac{\partial \hat{\mathbf{s}}(t)}{\partial u(t)} + \hat{\mathbf{s}}(t)\frac{\partial \mathbf{H}(t)}{\partial u(t)}\mathbf{s}(t) + \\ &\ \mathbf{s}^T(t)\frac{\partial \mathbf{H}(t)}{\partial u(t)}\hat{\mathbf{s}}(t) + 2ku(t). \end{aligned} \tag{4.37}$$

From (4.2), (4.37) and applying the chain rule, $\partial\hat{L}(t)/\partial w_i(t)$ in (4.34) is computed as follows

$$\begin{aligned}\frac{\partial\hat{L}(t)}{\partial w_i(t)} &= \frac{\partial\hat{L}(t)}{\partial u(t)}\frac{\partial u(t)}{\partial w_i(t)} \\ &= \left[\frac{\partial\hat{\mathbf{s}}^T(t)}{\partial u(t)}\mathbf{H}(t)\mathbf{s}(t) + \mathbf{s}^T(t)\mathbf{H}(t)\frac{\partial\hat{\mathbf{s}}(t)}{\partial u(t)} + 2ku(t)\right]\frac{\partial u(t)}{\partial w_i(t)} + \\ &\quad \hat{\mathbf{s}}(t)\frac{\partial\mathbf{H}(t)}{\partial w_i(t)}\mathbf{s}(t) + \mathbf{s}^T(t)\frac{\partial\mathbf{H}(t)}{\partial w_i(t)}\hat{\mathbf{s}}(t).\end{aligned} \tag{4.38}$$

From (4.2), $\partial u(t)/\partial w_i(t) = s_i(t)$.

$\partial\hat{\mathbf{s}}(t)/\partial u(t)$ in (4.38) is approximated as follows

$$\frac{\partial\hat{\mathbf{s}}(t)}{\partial u(t)} \approx \frac{(\hat{\mathbf{s}}(t) - \hat{\mathbf{s}}(t-\Delta t))}{\Delta_u(t)}, \tag{4.39}$$

where $\hat{\mathbf{s}}(.)$ is computed using (4.25) and

$$\Delta_u(t) = \begin{cases} \epsilon_u & \text{, if } |\Delta u(t)| \leq \epsilon_u \text{ and } \Delta u(t) \geq 0 \\ -\epsilon_u & \text{, if } |\Delta u(t)| \leq \epsilon_u \text{ and } \Delta u(t) < 0 \\ \Delta u(t) & \text{, otherwise,} \end{cases} \tag{4.40}$$

where ϵ_u is a small positive constant and $\Delta u(t) = u(t) - u(t-\Delta t)$. The constant ϵ_u is introduced to avoid a division by zero in (4.39), i.e., when $\Delta u(t) = 0$. The author uses the approximation of $\partial\hat{\mathbf{s}}(t)/\partial u(t)$ in (4.39) with the assumption that $\dot{\mathbf{s}}(t)$ is continuous everywhere in the state space without drastic changes even if $u(t)$ is changed drastically. This is actually the aforementioned assumption required to estimate $\dot{\mathbf{s}}$ except the change of $u(t)$ is included.

$\partial \mathbf{H}(t)/\partial w_i(t)$ in (4.38) is solved as follows. First of all, write

$$\mathbf{w}(t)\mathbf{w}^T(t) = \begin{bmatrix} . & \cdots & . & w_1(t)w_i(t) & . & \cdots & . \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ . & & . & w_{i-1}(t)w_i(t) & . & & . \\ w_i(t)w_1(t) & \cdots & w_i(t)w_{i-1}(t) & w_i^2(t) & w_i(t)w_{i+1}(t) & \cdots & w_i(t)w_n(t) \\ . & & . & w_{i+1}(t)w_i(t) & . & & . \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ . & \cdots & . & w_n(t)w_i(t) & . & \cdots & . \end{bmatrix}, \tag{4.41}$$

where only elements of the i-th row and i-th column of $\mathbf{w}(t)\mathbf{w}^T(t)$ are shown. Note that elements that are not shown in (4.41) do not contain w_i.

Using (4.41) and the fact that

$$\frac{\partial \mathbf{Q}(t)}{\partial w_i(t)} = \mathbf{0}, \tag{4.42}$$

we can compute

$$\begin{aligned} \frac{\partial \mathbf{H}(t)}{\partial w_i(t)} &= \frac{\partial(\mathbf{w}(t)\mathbf{w}^T(t) + \mathbf{Q}(t))}{\partial w_i(t)} \\ &= \frac{\partial(\mathbf{w}(t)\mathbf{w}^T(t))}{\partial w_i(t)} \\ &= \begin{bmatrix} 0 & \cdots & 0 & w_1(t) & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & & 0 & w_{i-1}(t) & 0 & & 0 \\ w_1(t) & \cdots & w_{i-1}(t) & 2w_i(t) & w_{i+1}(t) & \cdots & w_n(t) \\ 0 & & 0 & w_{i+1}(t) & 0 & & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & w_n(t) & 0 & \cdots & 0 \end{bmatrix}. \end{aligned} \tag{4.43}$$

We obtain

$$\hat{\mathbf{s}}^T(t)\frac{\partial \mathbf{H}(t)}{\partial w_i(t)}\mathbf{s}(t) = s_i(t)\sum_{j\neq i}^{n} w_j(t)\hat{s}_j(t) + \hat{s}_i(t)\left(2w_i(t)s_i(t) + \sum_{j\neq i}^{n} w_j(t)s_j(t)\right). \tag{4.44}$$

Similarly,

$$\mathbf{s}^T(t)\frac{\partial \mathbf{H}(t)}{\partial w_i(t)}\hat{\dot{\mathbf{s}}}(t) = \hat{\dot{s}}_i(t)\sum_{j\neq i}^{n} w_j(t)s_j(t) + s_i(t)\left(2w_i(t)\hat{\dot{s}}_i(t) + \sum_{j\neq i}^{n} w_j(t)\hat{\dot{s}}_j(t)\right). \quad (4.45)$$

Adding (4.45) to (4.44), we obtain

$$\begin{aligned}\hat{\dot{\mathbf{s}}}^T(t)\frac{\partial \mathbf{H}(t)}{\partial w_i(t)}\mathbf{s}(t) + \mathbf{s}^T(t)\frac{\partial \mathbf{H}(t)}{\partial w_i(t)}\hat{\dot{\mathbf{s}}}(t) = \; & 2\Big(w_i(t)s_i(t)\hat{\dot{s}}_i(t) + \\ & \textstyle\sum_{j\neq i}^{n} w_j(t)\big(s_i(t)\hat{\dot{s}}_j(t) + \hat{\dot{s}}_i(t)s_j(t)\big)\Big).\end{aligned} \quad (4.46)$$

Hence, the author approximates $\partial\hat{L}(t)/\partial w_i(t)\big|_{t=\tau}$ in (4.34) by substituting (4.40) and (4.46) into (4.38) and then replacing the argument t with τ.

4.3.4 Experimental Results

This subsection presents the experiment results of controlling the cart-pole plant with three types of control methods. (1) *Non-Adaptive Control* (NAC), where $\mathbf{w}$ is kept constant. (2) TAC/Q, where $\mathbf{w}$ is tuned on-line based on FEF but $\mathbf{Q}$ of FEF is kept constant. (3) TAC/FEF. In the experiments, three types of control problems are considered. (1) *Stabilization Problem*, e.g., the problem of balancing the pole and keeping it upright and at the same time maintaining the cart position at a center (i.e., the same cart-pole balancing problem as in the previous experiments). (2) *Swinging-up Problem*, e.g., the same as the stabilization problem except that the pole is pointing downward initially. (3) *Control of Virtual Cart-Pole Plant* in which the cart-pole plant is a virtual plant developed using *ODE* (Open Dynamics Engine) Software. The last control problem involves both stabilization and swinging-up problem.

The same as in TAC/FEF, in TAC/FEF the state vector of the plant is chosen as $\mathbf{s}(t) = [s_1(t)\ s_2(t)\ s_3(t)\ s_4(t)]^T$ and assumed completely observable. Two elements of $\dot{\mathbf{s}}(t)$, i.e., $\dot{s}_2(t)$ and $\dot{s}_4(t)$, are assumed to be unobservable but estimated as $\dot{s}_2(t) \approx \hat{\dot{s}}_2(t) = (s_2(t+\triangle t) - s_2(t))/\triangle t$ and $\dot{s}_4(t) \approx \hat{\dot{s}}_4(t) = (s_4(t+\triangle t) - s_4(t))/\triangle t$. Together with $\dot{s}_1(t) = s_2(t)$ and $\dot{s}_3(t) = s_4(t)$, both estimated elements are then collected into an estimated vector of $\dot{\mathbf{s}}(t)$, i.e., $\hat{\dot{\mathbf{s}}}(t) = [s_2(t)\ \hat{\dot{s}}_2(t)\ s_4(t)\ \hat{\dot{s}}_4(t)]^T$. Hence, we also have

$$\frac{\partial\hat{\dot{\mathbf{s}}}(t)}{\partial u(t)} = \Big[0\ \ \frac{\partial\hat{\dot{s}}_2(t)}{\partial u(t)}\ \ 0\ \ \frac{\partial\hat{\dot{s}}_4(t)}{\partial u(t)}\Big].$$

Non-zero elements of $\partial\hat{\dot{\mathbf{s}}}(t)/\partial u(t)$ are computed using (4.39).

The design parameters are chosen as $\eta_w = 3.0 \times 10^{-8}$, $\eta_q = 1.2 \times 10^{-6}$, $k = 100$, $\epsilon_u = 0.2$, $\epsilon_H = 1.0$, and $\epsilon_L = 1.0$. An initial time derivative of the plant state is set to $\hat{\dot{\mathbf{s}}}(0) = \mathbf{0}$. The controller parameters are initialized the same as in the experiments with TAC/PEF, i.e., $\mathbf{w}(0) = [50\ 5\ 5\ 5]^T$. To keep $\mathbf{w}(t)$ consistent (i.e., all of its elements are positive) and not to become infinite due to badly tuned, elements of $\mathbf{w}(t)$ are kept within the range $[1, 10]$ except for $w_1(t)$ within the range $[1, 60]$.

4.3.4.1 Stabilization Problem

The experiments comparing the performances of TAC based on FEF for various $\mathbf{Q}(0)$ were first carried out to find the "best" initial matrix $\mathbf{Q}(0)$. Figures 4.7 and 4.8 show the experiment results of the responses of the cart-pole plant controlled with TAC/Q and TAC/FEF using the initial matrices of $\mathbf{Q}(0)$ set to $\mathbf{0}$, $10\mathbf{I}$, and $70\mathbf{I}$, where $\mathbf{I}$ denotes an identity matrix. The initial pole angle is 35 deg and the initial cart position is 0.8 m. TAC/Q succeeded for all those initial matrices $\mathbf{Q}(0)$ except for $\mathbf{Q}(0) = 70\mathbf{I}$. In contrast, TAC/FEF succeeded for all those initial matrices. Good performances obtained either using TAC/Q or TAC/FEF seem due to the cart-pole balancing problem for the above initial conditions is not difficult to solve. The best performance of both TAC/Q and TAC/FEF is obtained when $\mathbf{Q}(0) = 10\mathbf{I}$. These experiments results obviously imply that $\mathbf{Q}(0)$ cannot be chosen arbitrarily, rather it must be within a certain range. In the following explanations, only the simulation results using $\mathbf{Q}(0) = 10\mathbf{I}$ and the initial cart position is 0.8 m are given.

Figures 4.9 and 4.10 show the simulation results for the initial pole angles of $\theta(0) =$ 40 deg and $\theta(0) = 45$. TAC/FEF keeps good performance for all those initial conditions. Compared to the simulation results of TAC based on PEF shown in Figures 4.4 and 4.5 where the initial conditions are the same, the performances of TAC/PEF and TAC/FEF seem to be similar. The differences are seen in performances when the parameters of the evaluation function (i.e., PEF and FEF) are kept unchanged. TAC/H only succeeded for $\theta(0) = 40$ deg (see Figure 4.4), otherwise failed. Despite TAC/H succeeded, its performance is much worse than TAC/Q (see Figure 4.9). These results seem to imply that initial FEF is better than initial PEF. The goodness of the initial PEF is dependent on $\mathbf{Q}$ and $\mathbf{H}(0)$ of $\hat{L}(t)$ in (4.8). While, the goodness of FEF is determined by $\mathbf{H}(0) = \mathbf{w}(0)\mathbf{w}^T(0) + \mathbf{Q}(0)$. Note that the definition of $\mathbf{H}$ in FEF is totally different from that in PEF.

Figure 4.11 and 4.12 shows the good performance of TAC/FEF even for the large initial pole angles, i.e., $\theta(0) = 50$ and 55 deg, respectively. As shown in Figure 4.6,

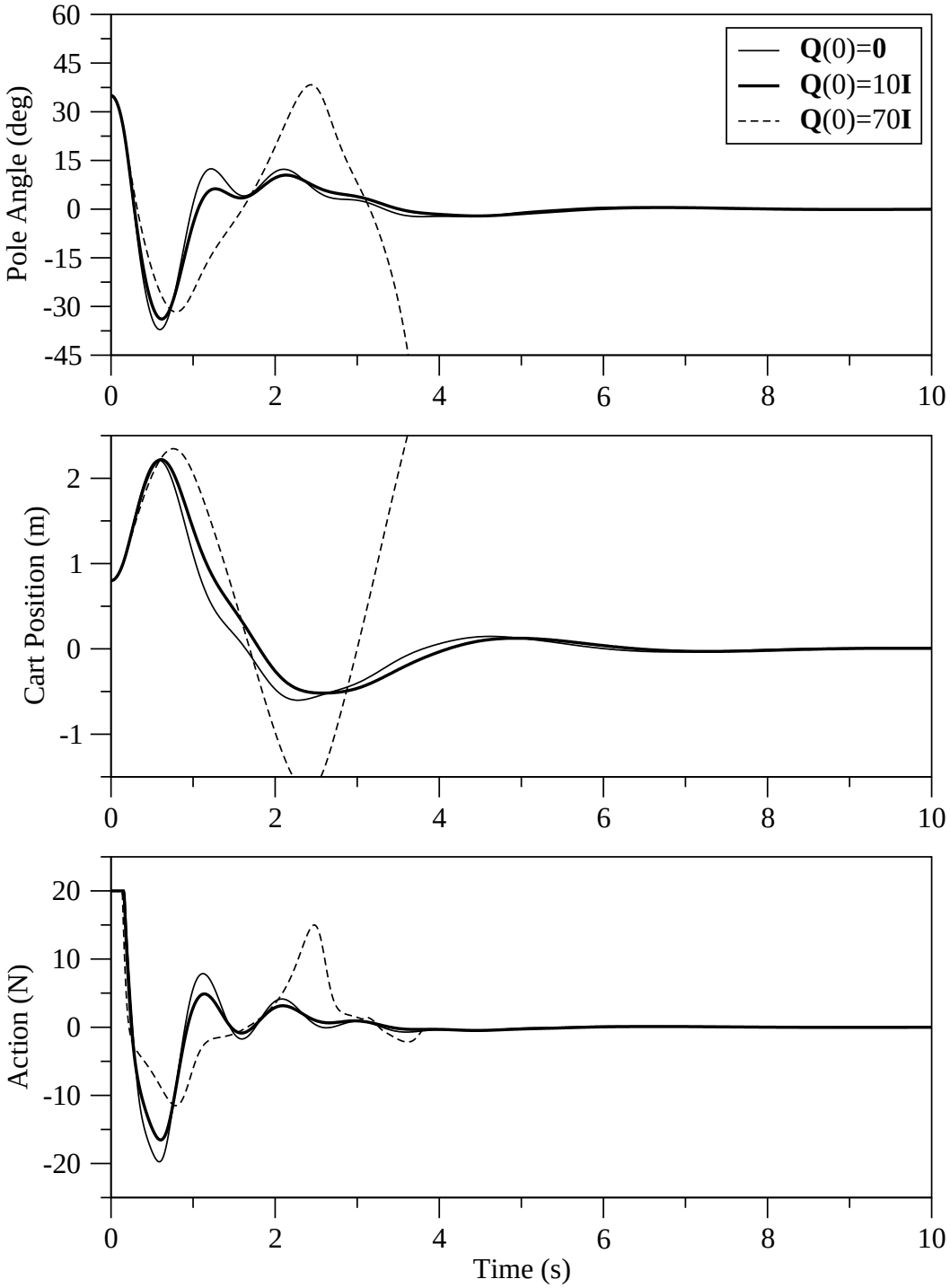


Figure 4.7: TAC/Q for various $\mathbf{Q}(0)$ used to solve cart-pole balancing problem with initial pole angle of 35 deg and initial cart position of 0.8 m.

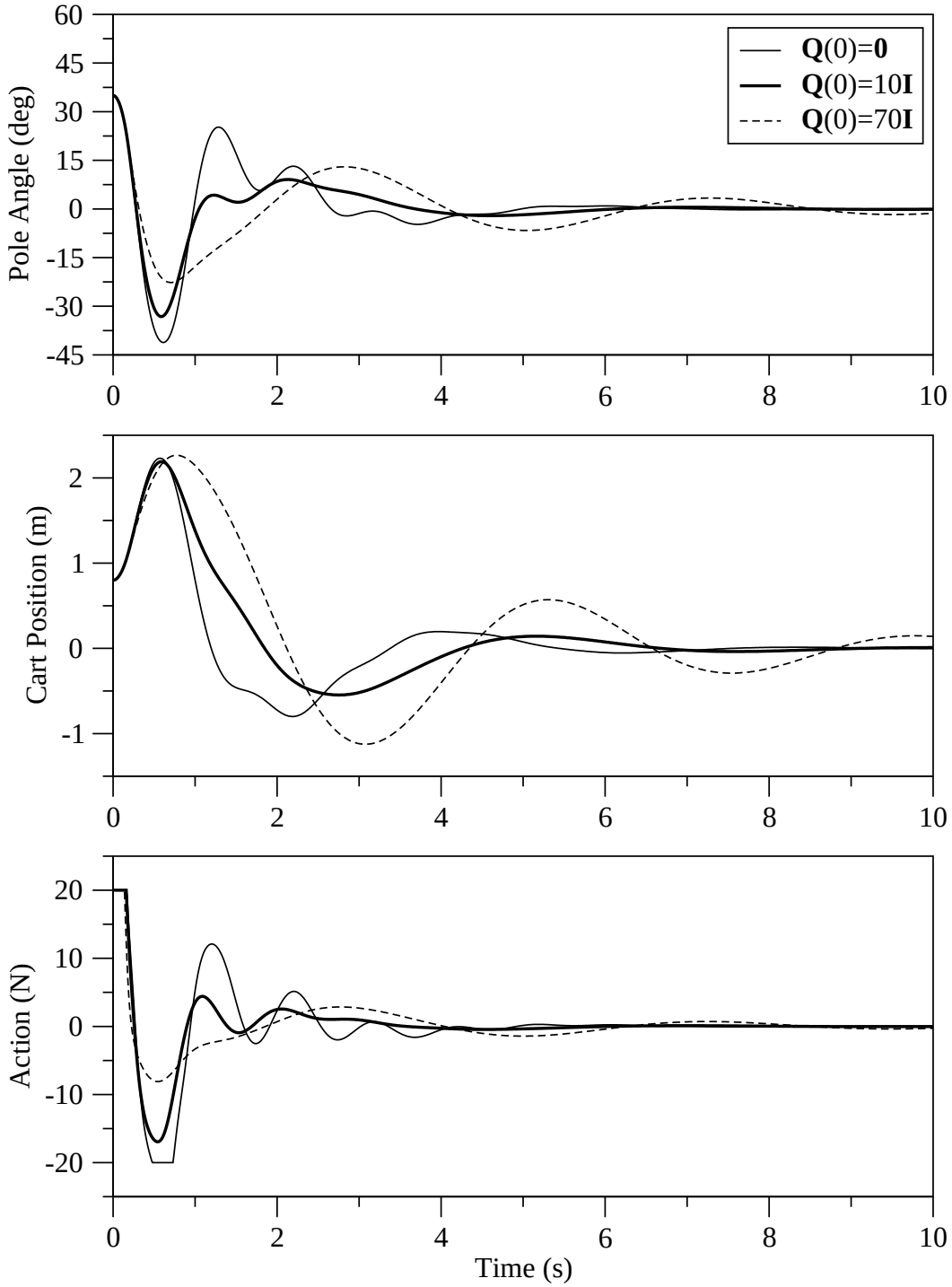


Figure 4.8: TAC/FEF for various $\mathbf{Q}(0)$ used to solve cart-pole balancing problem with initial pole angle of 35 deg and initial cart position of 0.8 m.

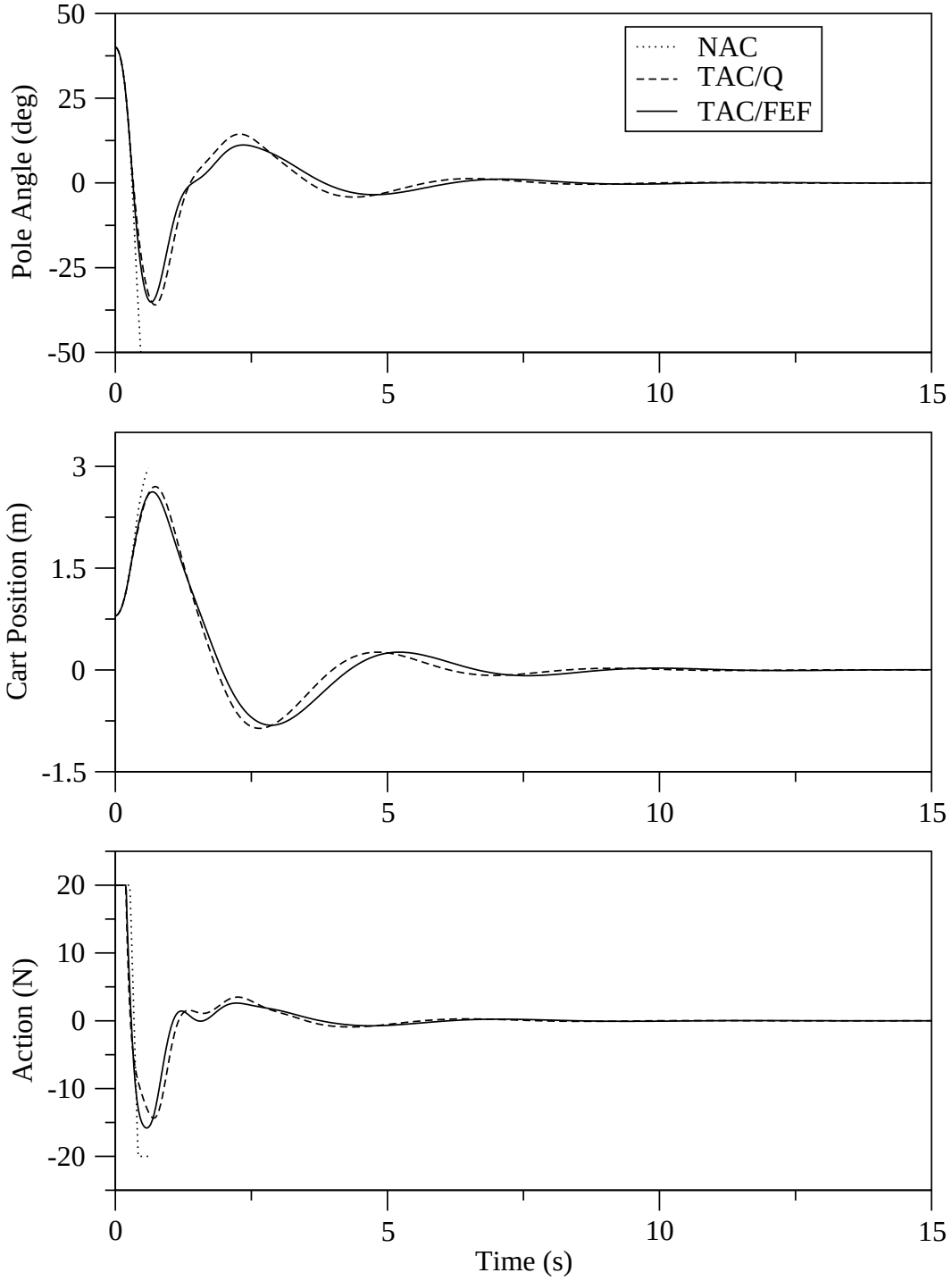


Figure 4.9: TAC based on FEF used to solve cart-pole balancing problem with $\theta(0) = 40$ deg and $x(0) = 0.8$ m.

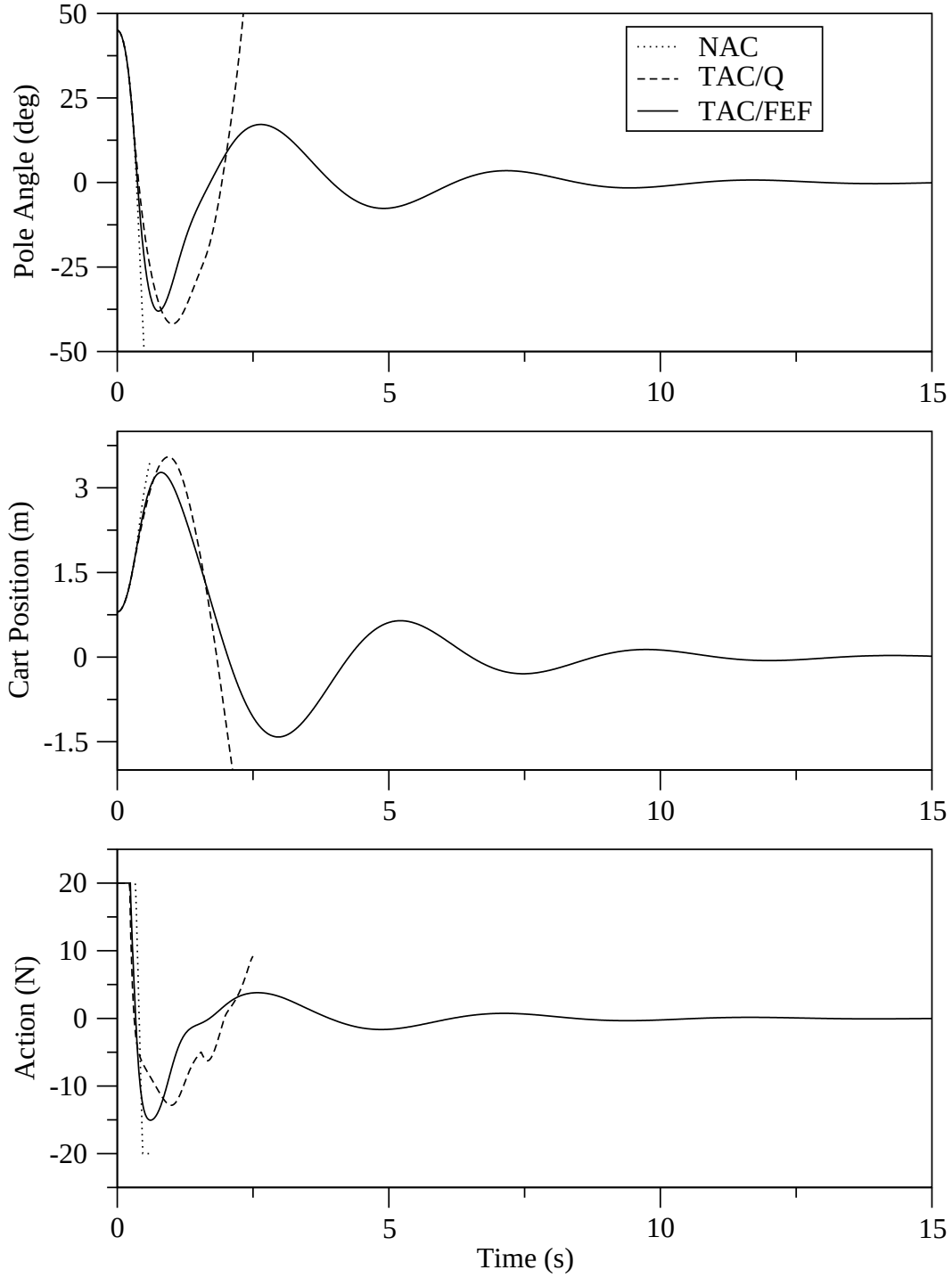


Figure 4.10: TAC based on FEF used to solve cart-pole balancing problem with $\theta(0) = 45$ deg and $x(0) = 0.8$ m.

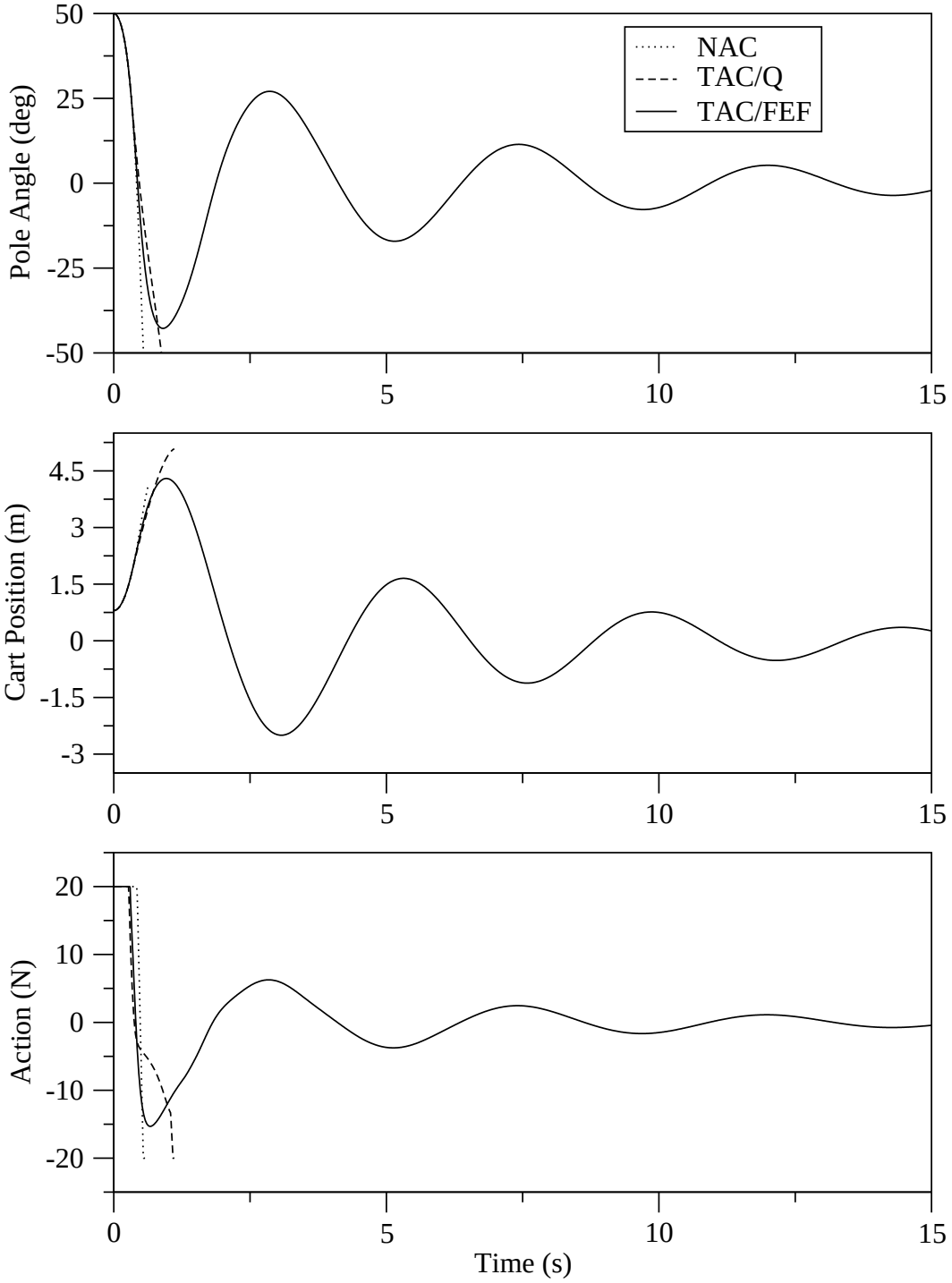


Figure 4.11: TAC based on FEF used to solve cart-pole balancing problem with $\theta(0) = 50$ deg and $x(0) = 0.8$ m.

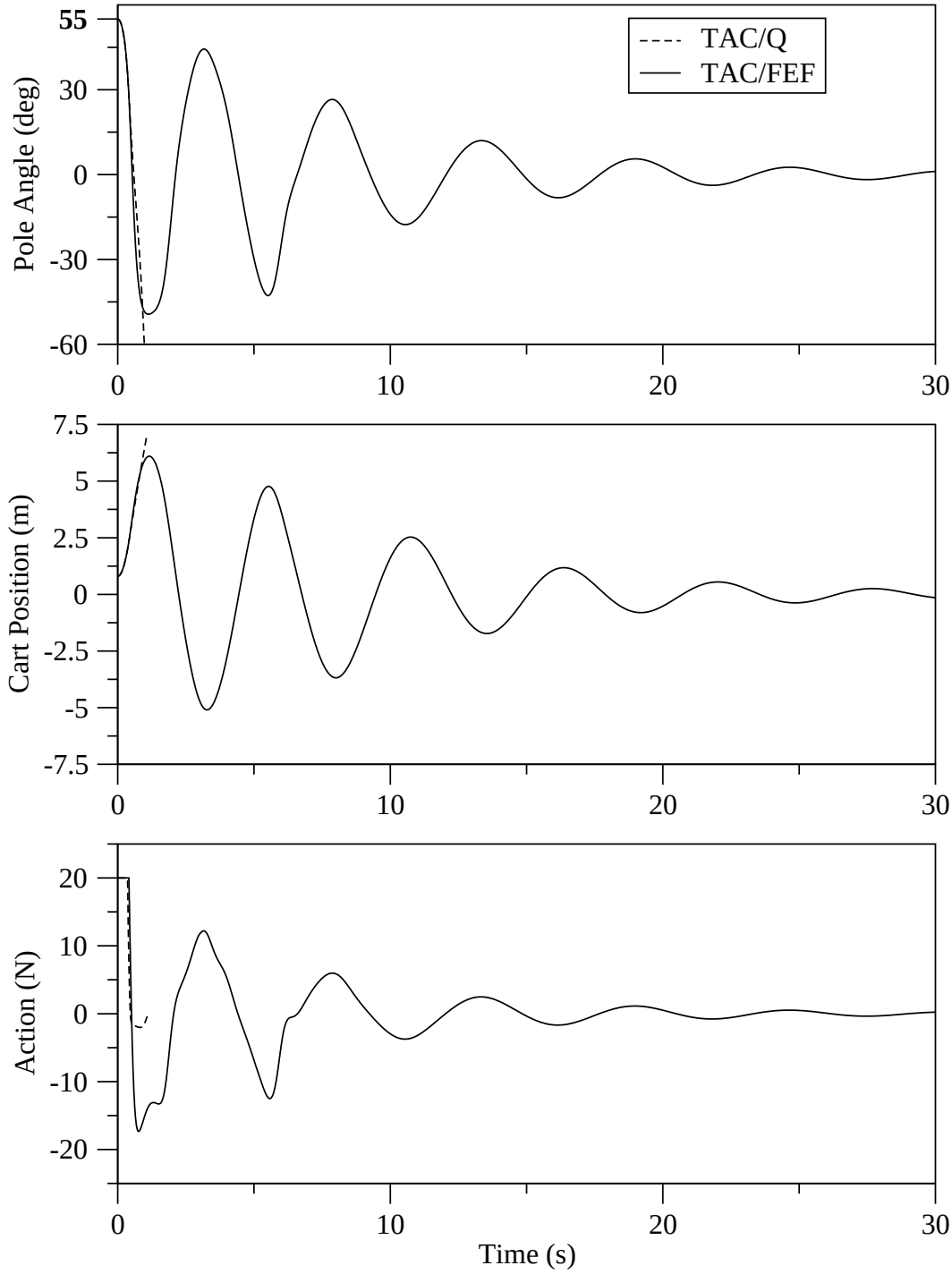


Figure 4.12: TAC/FEF for $\theta(0) = 55$ deg and $x(0) = 0.8$ m.

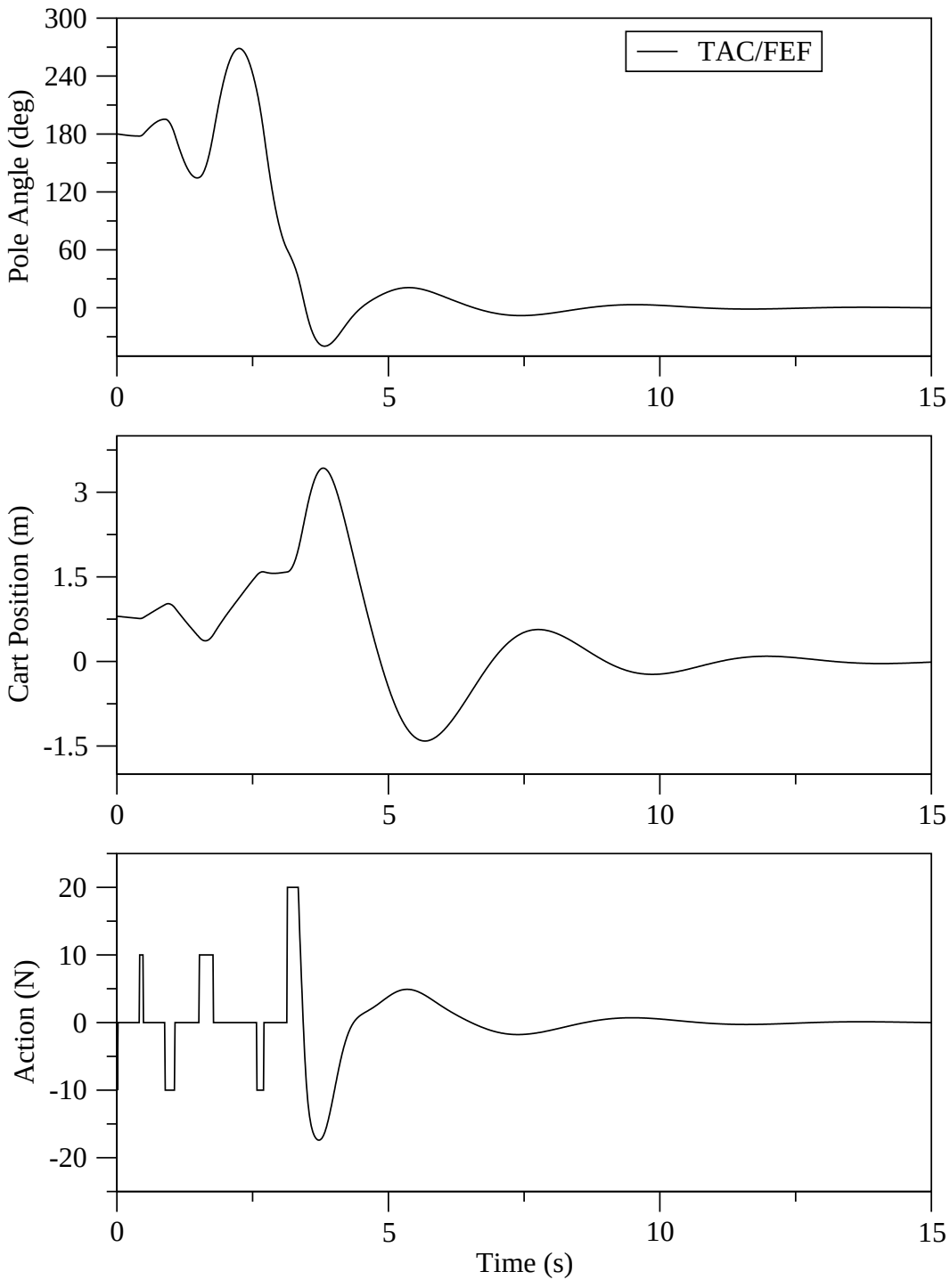


Figure 4.13: TAC/FEF for solving swinging-up problem.

TAC/PEF failed for $\theta(0) = 50$. These results proved that TAC/FEF is better than TAC/PEF in term of the initial pole angle.

4.3.4.2 Swinging-up Problem

In the swinging-up problem, the pole pointing downward at rest initially (i.e., $\theta(0) = 180^0$ and $\dot{\theta}(0) = 0$ deg/s) must be swung up and then balanced in an upright position and at the same time the cart must be maintained at a center position. Since NAC and TAC/Q are much worse than TAC/FEF in the previous experiments, in this section, the author focuses on the experiments with TAC/FEF only.

It will be very difficult for TAC/FEF to solve the swinging-up problem directly. The simplest way to solve this problem is similar to that used by FCAPS/FD, i.e., to swing the pole progressively higher from its rest downward position until it reaches a region where TAC/FEF can work well. To implement this idea, the pole is swung up by applying the following actions:

$$u = \begin{cases} 20 \text{ N}, & \text{if } \theta(0) = 180 \text{ deg}, \\ u_s, & \text{if } \cos(\theta(t)) < \cos(135), \\ 0 \text{ N}, & \text{if } \cos(135) < \cos(\theta(t)) < \cos(60), \end{cases} \tag{4.47}$$

where u_s is the *swinging* actions as follows:

$$u_s = \begin{cases} 0 \text{ N}, & \text{if } \theta(t) < 180 \text{ deg and } \dot{\theta}(t) < 0 \text{ deg/s}, \\ & \text{or } \theta(t) > 180 \text{ deg and } \dot{\theta}(t) > 0 \text{ deg/s}, \\ 10 \text{ N}, & \text{if } \theta(t) < 180 \text{ deg and } \dot{\theta}(t) > 0 \text{ deg/s}, \\ -10 \text{ N}, & \text{if } \theta(t) > 180 \text{ deg and } \dot{\theta}(t) < 0 \text{ deg/s}. \end{cases} \tag{4.48}$$

In (4.47), $u = 20$ N is applied when the pole is pointing downward at rest initially. The swinging actions are applied as long as $\cos(\theta(t)) < \cos(135)$. No action is applied when $\cos(135) < \cos(\theta(t)) < \cos(60)$. Then, TAC/FEF is applied only when $\cos(\theta(t)) > \cos(60)$ (or equivalently, $|\theta| < 60$ deg) in which the author assumes that TAC/FEF can work well. Note that the swinging actions of (4.47) is different form the swinging actions of (3.13) that is only applicable for solving the pole-swinging and balancing problem with CAPS which ignored the cart position. The swinging actions of (4.47) are defined after trials and errors that can prevent the cart position too far moving away from the center. They are required to enable TAC/FEF to work well once the pole angle enters the range $|\theta(t)| < 60$ deg,

When the pole is entering the range $|\theta| < 60$ deg for the first time, TAC/FEF is only given the "initial condition" of angle $|\theta(0)| \approx 60$ deg. But, we cannot know *a priori* the "initial conditions" of $x(0)$, $\dot{\theta}(0)$ and $\dot{x}(0)$ at this time, because it depends on the total amount of energy pumped to the pole by the swinging actions in (4.47). This amount of energy can be determined through experiments only. Intuitively, we can say that TAC/FEF tends to fail to balance the pole and to center the cart, given too high values of $\dot{\theta}(0)$ and $\dot{x}(0)$. To avoid this problem, we must be careful in setting the value of the swinging action for each situation in (4.47), because either too weak or too strong swinging actions may result in too high velocity of the pole angle and the cart. If the swinging actions are too weak, the pole takes long time to swing up and enters the region $|\theta(t)| < 60$ deg. And, swinging the pole for too long time will pump very large energy to it. On the contrary, if the swinging actions are too strong, the pole will also receive very large energy in much shorter time. While it remains unclear how the cart behaves in both cases, the angle velocity of the pole $\dot{\theta}(t)$ becomes very high when the pole enters the region $|\theta(t)| < 60$ deg.

Figures 4.13 shows the responses of TAC/FEF in the swinging up problem. By applying the actions in (4.47), the pole angle could be moved to the desired region after about 3 s, while the cart moved away from the center. Afterward, TAC/FEF succeeded to keep both the pole upright and the cart at the center.

4.3.4.3 Control of Virtual Plant

Using the ODE software, the author developed a virtual cart-pole plant and calls it an ODE cart-pole plant (see Appendix for the source code). Figure 4.14 shows a construction of the plant developed using the ODE sofware. The control of the ODE cart pole plant is done by applying action (i.e., force in Newton) to the center of the cart's mass but kept parallel to the base. Table 4.1 gives the parameters used in the construction of the ODE cart-pole plant. Table 4.2 gives the parameters used by ODE to simulate "real" world of the experiment. Briefly, Error Reduction Parameter (ERP) and Constraint Force Mixing (CFM) are two parameters that need to be set properly in ODE to control the spongyness and springyness of the joint. In the ODE cart-pole plant construction, the author uses a *hinge joint* to connect the pole to the arm and the arm to the cart. In ODE, a positive vertical force is upward, therefore the gravity acceleration is of a negative sign.

Simulation of the control of the ODE cart-pole plant uses the following design parameters: $\eta_w = 3 \times 10^{-8}$, $\eta_q = 2 \times 10^{-6}$, $k = 100$, $\epsilon_u = 0.1$, $\epsilon_H = 1.0$, and $\epsilon_L = 1.0$.

An initial time derivative $\hat{\mathbf{s}}(0)$ of the plant state is set to $\mathbf{0}$. The controller parameters are initialized the same as in the experiments with TAC/PEF, i.e., $\mathbf{w}(0) = [50\ 5\ 5\ 5]^T$. All elements of $\mathbf{w}(t)$ are kept within the range $[1, 10]$ except for $w_1(t)$ within the range $[1, 60]$.

Figure 4.14: Virtual cart-pole plant

Table 4.1: Parameters of Cart-Pole

Item	Size (m^3)	Mass (kg)
Cart	0.30 x 0.20 x 0.14	1.00
Pole	0.02 x 0.02 x 1.00	0.10
Arm	0.02 x 0.25 x 0.02	0.05

Table 4.2: ODE Parameters

Gravity acceleration	-9.8 m/s^2
Error Reduction Parameter (ERP)	0.9
Constraint Force Mixing (CFM)	0.1
Friction Coefficient between cart and base	0.00001

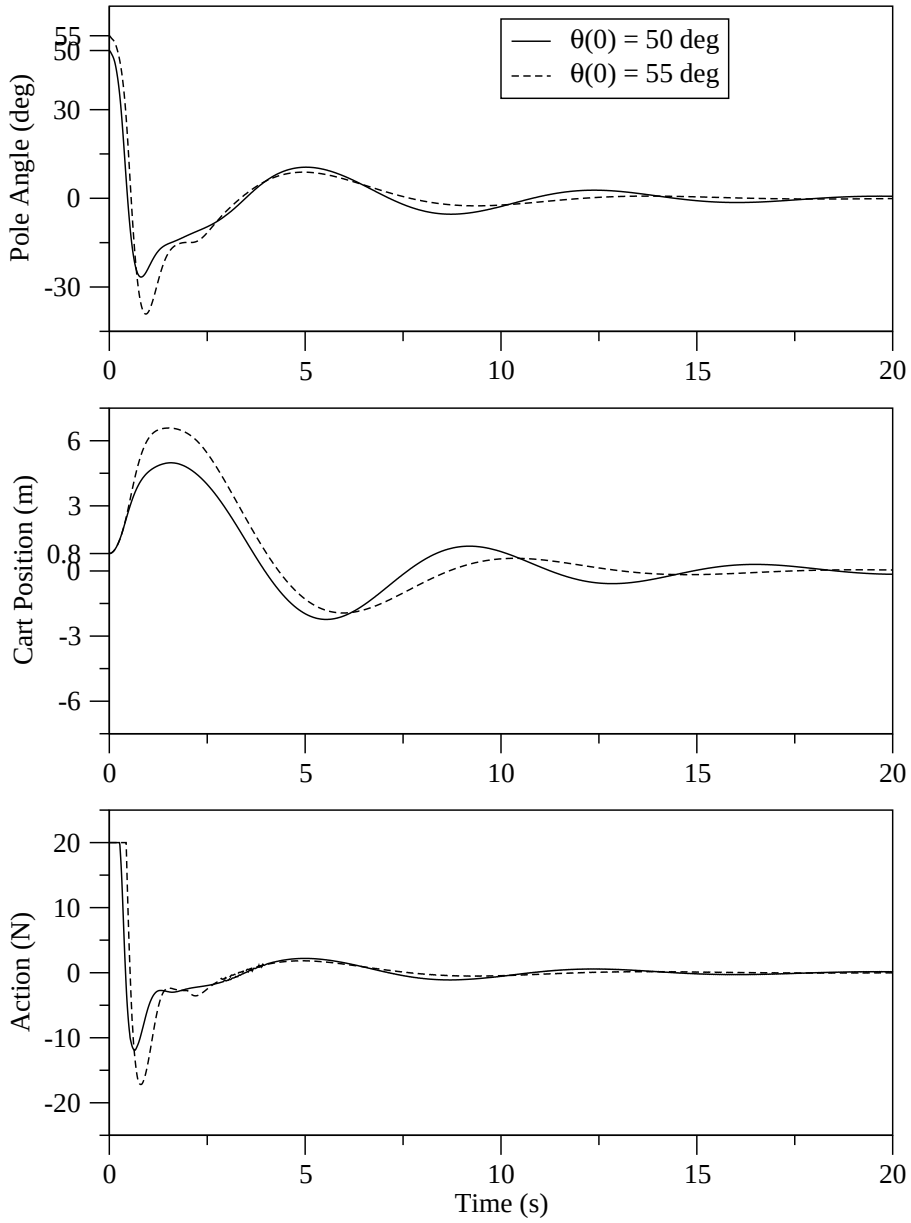


Figure 4.15: TAC/FEF under ODE for solving stabilization problem

Figures 4.15 shows the simulation results of applying TAC/FEF for solving the stabilization problem of the ODE cart-pole plant for the initial pole angles of $\theta(0) = 50$ deg and $\theta(0) = 55$ and the initial cart position of $x(0) = 0.8$ m. TAC/FEF keeps good performance for both those initial conditions. These experiment results are slightly different from those of TAC/FEF in the previous experiments using the source code of

Figure 4.16: Control of ODE cart-pole plant subject to perturbations, i.e., the base is inclined and a box is thrown to the pole

the cart pole dynamics created by the author. These differences seems due to the fact that the ODE cart pole plant does not behave exactly the same as the the author's cart pole dynamics. In addition, the difference is a matter of choice of parameter control design considered as "good" for each dynamics which is generally only based on trial and error.

Figure 4.17 shows the simulation result for TAC/FEF successfully solving the swinging up and then stabilization problem of the ODE cart-pole plant even if the cart pole dynamics is subject to pertubations (see Figure 4.16 for the illustration of this control). To obtain that result, the pole was swung up by applying the following actions:

$$u = \begin{cases} 20 \text{ N}, & \text{if } \theta(0) = 180 \text{ deg}, \\ u_s, & \text{if } \cos(\theta(t)) < \cos(155), \\ 0 \text{ N}, & \text{if } \cos(135) < \cos(\theta(t)) < \cos(60), \end{cases} \tag{4.49}$$

which is similar to (4.47) except that u_s of (4.48) is applied when $\cos(\theta(t)) < \cos(155)$ (or equivalently, 155 deg $< \theta(t) <$ 255 deg). TAC/FEF is then applied when the pole

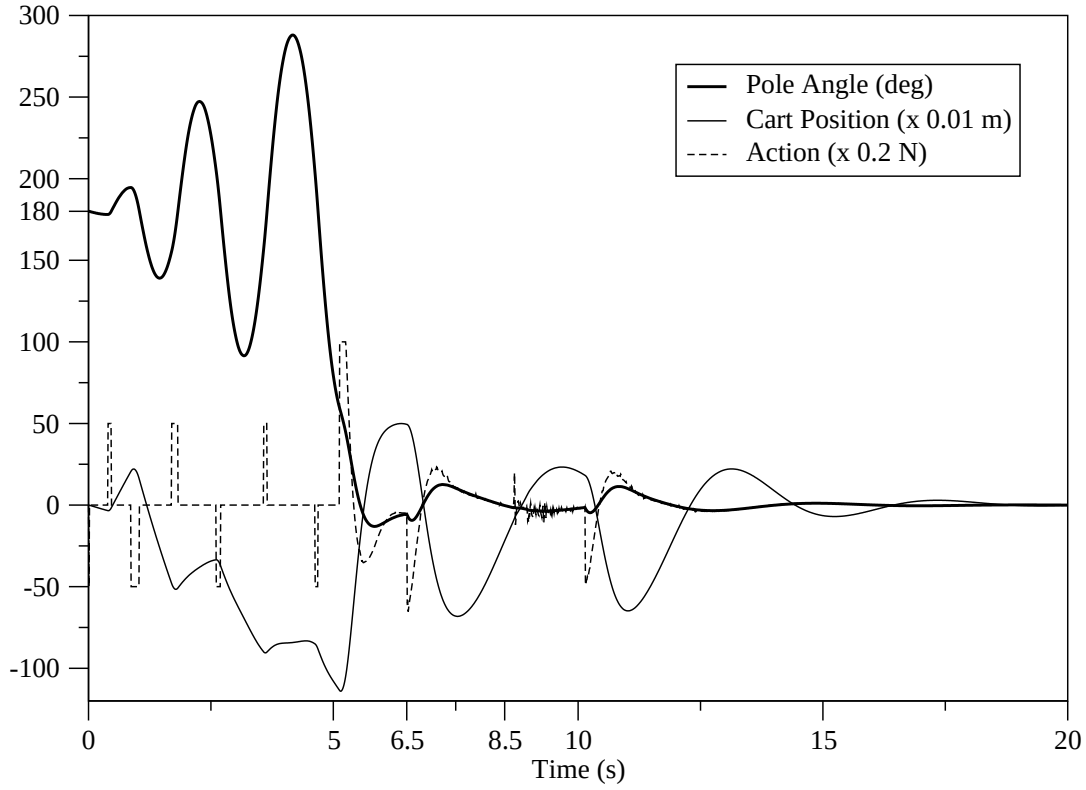


Figure 4.17: TAC/FEF under ODE with perturbations, i.e., the base is inclined and a box is thrown to the pole

angle $\theta(t)$ enters the range $\cos(\theta(t)) < \cos(60)$ within which the author assumes that TAC/FEF under ODE can work well.

The swinging actions generated using (4.49) lead $\theta(t)$ entering $\cos(\theta(t)) < \cos(60)$ after about 5 s. While TAC/FEF is not yet completely balancing the pole and maintaining the cart at the center, the cart-pole plant is subject to several perturbations. A free box is thrown to the pole at the time of 6.5 s. The mass of the free box is 0.4 kg, initially positioned at about as high as the upper end of the pole and then hitted by a force of 100 N. At the time of 8.5 s, the base is inclined by about 4 deg counter-clockwise. Then, once again at the time of 10 s, the free box from the same position and with the same force is thrown to the pole. As can be seen in Figure 4.17, throwing the box to the pole results in more perturbation than inclining the base. Despite these perturbations, TAC/FEF keeps working well and completely balancing the pole and maintaining the cart at the center after about 17 s.

The perturbation due to the inclined base could be simply overcome by adding an extra force to the force generated by TAC/FEF, i.e.,

$$[\text{total mass of cart, pole and arm}] \times g \times \sin[\text{inclination angle}]. \tag{4.50}$$

This success seems due to the fact that when the base is inclined the cart can be thought of being given a perturbation of a horizontal force of

$$-[\text{total mass of cart, pole and arm}] \times g \times \sin[\text{inclination angle}]. \tag{4.51}$$

The forces of (4.51) and (4.50) have the same but opposite-signed values. Hence, the extra force of (4.50) can be thought of having a role to "cancel" the perturbing force of (4.51).

4.4 Discussion

The way of finding the appropriate actions in both TAC/PEF and TAC/FEF can be considered as similar to that of the policy iteration in RL [7]. In the policy iteration of RL, once a policy gets improved due to following state trajectories recommended by a value function, we can then compute the value function and improve it again to yield an even better policy. In both TAC/PEF and TAC/FEF, once the controller parameters get improved, the better actions are produced that make PEF and FEF get improved. Then, the improved PEF and FEF lead to even better controller parameters.

Representing $\mathbf{H}(t) = \mathbf{w}^T(t)\mathbf{w}(t) + \mathbf{Q}(t)$ should make the burden of initialization of $\mathbf{H}(0)$ in FEF not as hard as that in PEF. In FEF, when we have a "good" guess of $\mathbf{Q}(0)$, the initialization problem is only to find "good" guess of $\mathbf{w}(0)$. This is not the case for PEF that requires a "harder" effort for initialization of its parameters due to the fact that PEF has more parameters to be determined, i.e., $\mathbf{Q}$, $\mathbf{H}(t)$, and $\mathbf{w}(t)$ that are not related each other.

A typical appropriate evaluation function includes not only the action and the state at which that action is executed, but also the result of executing the action. This is true for a state-action value function widely used in RL where a precise state-action value function is defined as equal to the reward received at the next time step plus a properly discounted value of the succeeding state-optimal actions. However, that is not true for PEF that does not include any result of executing the action. In contrast, because FEF includes the action, the state, and the result of executing the action (i.e., the next state), we can suppose that FEF has a role similar to the state-action-value

function of RL. The simulation results show that the appropriate actions could be found through an adaptive tuning of the controller parameters that minimizes FEF. This means that the smaller value of FEF represents the better performances of the action $u(t)$ executed at $\mathbf{s}(t)$, and hence, it should correspond to the higher value of the state-action $\{\mathbf{s}(t), u(t)\}$ in RL.

We can summarize comparisons of all the developed adaptive control methods as follows. CAPS is the most practical method among others because it need not to be given any good initial controller parameters, i.e., the initial controller parameters can be simply set to zero. However, for CAPS to work well, it requires the fixed approximate evaluation function defined based a priori assumption. Therefore, its dependency on the appropriate evaluation function is much stronger than TAC. And as approximating the fixed appropriate evaluation function a priori is possible only for a certain simple control problem, the applicability of CAPS is limited.

In contrast, TAC/PEF is applicable to the control problem more general than that considered in CAPS. In addition, TAC/PEF is only weakly dependent on the appropriate initial PEF as it can improve PEF on-line. However, TAC/PEF has a heavy burden of parameter initialization, i.e., it requires "good" guess of the initial parameters of both the controller and PEF. Also, for TAC/PEF to work well, it is required to make PEF a Lyapunov function which is a difficult or even impossible task to accomplish in most real complex control problems.

Finally, TAC/FEF is developed to overcome the limitations of TAC/PEF. TAC/FEF is better than TAC/PEF in terms of initial parameters of FEF that are not necessarily set with "good" guess and its applicability that is more practical. In addition, the simulations results show that TAC/FEF works better than TAC/PEF.

There have been large applications of the RL concept to design adaptive control as can be found in many literatures (see [23–31]). Those past research developed the methods for finding appropriate actions based on adaptive critic approach and approximate dynamic programming (or, AC/ADP for short). AC/ADP is essentially a juxtaposition of RL and Dynamic Programming (DP) ideas [31]. While DP derives the control actions via the optimal value function, AC/ADP utilizes an approximation of the optimal value function as the evaluation function to adjust the controller. Despite its successful applications to reduce inefficiency of RL, the adaptive control using AC/ADP is limited by the need of "crafting" an appropriate utility function (i.e., equivalent to the reward function in RL, *except* that the reward function is not "crafted" by the engineer but provided by the environment). The utility function is the only source of information required in AC/ADP to improve the approximate value function,

which is defined simply as equal to the utility value plus the discounted approximate value of the next state or the next state-action. The utility function is often simply defined as a Eucleadian distance (or, the error) from the goal state [23, 26]. Therefore, it is potential to cause the same control difficulties as in the classical adaptive control methods whose goals are simply to minimize the error. Alternatively, different relative weightings might be used for some of the error components as done in [29, 30] for specific control problems. However, other research of [31] provides examples of situations where seemingly minor changes in formulation of the utility function resulted in a dramatically different ADP convergence behavior and a controller design. Generally, defining an approparite utility function even for a specific control problem is not always an easy task. A common practice to cope with this problem is by performing some trial-and-error modifications of the utility function. TAC/PEF and TAC/FEF do not rely on any utility or reward function to improve the approximate evaluation function. Nevertheless, in the author's experiments no trial-and-error is required to improve the approximate evaluation function and good results were obtained. Thus, the author argues that both TACs are more practical and efficient than the adaptive control in those aforementioned literatures.

CHAPTER 5

Concluding Remarks

5.1 Summary of Dissertation

In the introduction, an evaluation function appropriate for guiding a controller to produce appropriate actions for a closed loop system has been defined as a function that represents a stable dynamics of the closed loop system. The evaluation function is also required to provide a direct measure of its gradient with respect to the action by which tuning of the controller parameters using a gradient method is possible. In this dissertation, the author has focused on showing that for certain types control problems, the evaluation function that satisfies the above requirements can be approximated, approximation errors in the evaluation function can be coped with through on-line tuning of the approximate evaluation function, and using that approximate evaluation function the appropriate actions can be determined in the adaptive control schemes whose architecture is almost the same as that of a policy search approach of reinforcement learning (RL). The author argues that the approaches to find appropriate actions based on the approximate evaluation functions proposed in the dissertation can be considered as the better alternatives than those of RL in terms of a higher efficiency but still of a high applicability to certain types of complex control problems provided that imprecise plant behavior is available.

In Chapter 3, the author dealt with a certain type of control problem in which the closer state to the goal state implies that the smaller actions are required. For

such a control problem, an approximated value function could be defined a priori. It is also readily used as an approximate evaluation function for evaluating and tuning the controller to produce the appropriate actions in an adaptive control scheme referred to as Control with Approximated Policy Search Approach (CAPS). No tuning of the approximate evaluation function is done in CAPS. A pole-balancing problem seems to fall into that certain type of the control problem solvable with CAPS. This is proved by the good results obtained shown in Chapter 3. Those good results are obtained either using a linear state-feedback controller or a fuzzy controller. The use of the fuzzy controller leads to much better results than those obtained with the state-feedback linear controller, but at the price of a heavy burden of tuning more fuzzy controller parameters.

Further, the author presented two types of the evaluation functions for a two-stage adaptive control (TAC) in which the evaluation functions and the controller are tuned in sequence. The first type is a partially adjustable evaluation function (PEF) and the second type is a fully adjustable evaluation function (FEF). To test its effectiveness, TAC was applied to solve a cart-pole balancing problem. The simulation results shown in Chapter 4 proved that using the proposed heuristic approaches TAC has improved both types of the evaluation functions through on-line tuning. And, despite being tuned, those evaluation functions could be used to evaluate and tune the controller to result in the appropriate actions that could keep the pole upright and the cart at a center position.

Comparisons of all the developed adaptive control methods can be summarized as follows. CAPS is the most practical method among others because it need not to be given any good initial controller parameters. However, it requires the fixed approximate evaluation function defined based on a priori assumption that makes its dependency on the appropriate evaluation function much stronger than TAC. This restricts the applicability of CAPS to more complex problems.

TAC based on PEF (i.e., TAC/PEF) is applicable to the control problem more general than that considered in CAPS. TAC/PEF is only weakly dependent on the appropriate initial PEF as it can improve PEF on-line. However, it has a heavy burden of initialization of the initial parameters of both the controller and PEF. Also, it needs to make PEF become a Lyapunov function which is a difficult or even impossible task to accomplish in most real control problems.

TAC based on FEF (TAC/FEF) could overcome the limitations of TAC/PEF. TAC/FEF is more practical and works better than TAC/PEF.

5.2 Future Work

As discussed in Chapter 1, the author desires adaptive control that "combines" two distinct philosophies, i.e., those of classical adaptive control and Reinforcement Learning (RL), as illustrated in Figure 1.1. However, all the proposed adaptive control have not reflected the desired adaptive control for solving wide range of control problems. The applicability of the proposed adaptive control (i.e., TAC in particular) is higher than conventional adaptive control and its efficiency is higher than RL when the restrictive assumptions are satisfied. The most restrictive assumption required in TAC is that we can make "good" guess of initial parameters of the evaluation function, particularly those of controller. Such an assumption is possible to satisfy when we have "good" understanding of plant behavior.

The author's future work will combine TAC with RL. RL is first applied. TAC is then applied to speed up RL when RL results in "good" controller parameters. Such a strategy should eliminate the need of making "good" guess of initial controller parameters. Further, we can hope that strategy will really work based on two distinct philosophies, i.e., those of classical adaptive control and RL, where high applicability and efficiency as illustrated in Figure 1.1 can be achieved.

Appendices: Software Listing

These appendices contain most of the source codes used to generate the experiment results presented in this dissertation. Those source codes are implementations of

1. LCAPS and FCAPS
2. TAC/FEF
3. Cart-pole dynamics
4. Gaussian Fuzzy Inference System
5. Supporting Functions
6. Control of a virtual cart-pole plant.

All codes are written in C and C++. The source code of LCAPS and FCAPS needs to include the source codes of Cart-pole dynamics, Gaussian Fuzzy Inference System, and Supporting Functions. While, the source code of TAC/FEF needs to include the source codes of Cart-pole dynamics and Supporting Functions. Source code of the control of the virtual cart-pole plant is written using the libraries of ODE (Open Dynamics Engine). ODE is an open source available on-line at http://www.ode.org. It is of high performance libraries for simulating rigid body dynamics, fully featured, stable, mature and platform independent with an easy to use C/C++ API.

A LCAPS and FCAPS

```
#include <iostream.h>
#include <stdlib.h>
#include <gaussfuzzy.h>
#include <ipendulum.h>
#include <myfunc.h>
#define MAX_STEP 8000

int main(int argc, char *argv[]){

    float t,u,old_u,du,D,old_D,dD,P,oldP,M[4],e[4],f,ym,lr,alpha_u;
    float state[4],ns[4],k[4];
    int step,trial,i,flag;

    float coi[]={-30.0,-20.0,0.0,20.0,30.0,
            -60.0,-30.0,0.0,30.0,60.0 };

    float sd[]={ 20.0,10.0,10.0,10.0,20.0,
              30.0,15.0,10.0,15.0,30.0};

    int pos[] ={-1,0,0,0,1,
                -1,0,0,0,1};

    GaussFuzzy * FCS = new GaussFuzzy(coi,sd,pos,5,5,-1,-1);
    FCS->SetPar(0.0);

    IPendulum * Pole = new IPendulum;
    Pole->GetState(state);

    FILE * pfile;
    char fname[256];

    if (argc !=7){
      printf("Syntax: %s <option1> <theta0> <x0> <option2> <lr> <\
      alpha_u>\n",argv[0]);
      /***********************************************
      option1:
        fds : setpoint problem, with failure detector
        fdt : tracking problem, with failure detector
```

```
    gds : setpoint problem, without failure detector
    gdt : tracking problem, without failure detector

  option2:
    fc: fuzzy controller
    lc: linear controller

    lr: adaptation rate
  ************************************************/
  return 0;
}

if (argv[1][0]=='f' && argv[1][1]=='d' && argv[1][2]=='s'){
    flag = 0;
}else if (argv[1][0]=='f' && argv[1][1]=='d' && argv[1][2]=='t'↘
){
    flag = 1;
}else if (argv[1][0]=='g' && argv[1][1]=='d' && argv[1][2]=='s'↘
){
    flag = 2;
}else if (argv[1][0]=='g' && argv[1][1]=='d' && argv[1][2]=='t'↘
){
    flag = 3;
}

lr = atof(argv[5]);
alpha_u = atof(argv[6]);

sprintf(fname,"%s-%s-%s-%4.0f-%8.6f.dat",argv[1],argv[2],argv↘
[4],lr,alpha_u);
pfile = fopen(fname,"w");

state[0] = atof(argv[2])/rad;
state[1] =  0.0/rad;
state[2] = atof(argv[3]);
state[3] =  0.0;
Pole->SetState(state);
t = u = 0.0;

if (flag == 1 | flag == 3){
```

```
  e[0] = state[0]-ym;
  e[1] = state[1]-M_PI/30.0*cos(t);
}else if (flag == 0| flag == 2){
  e[0] = state[0];
  e[1] = state[1];
}

e[2] = state[2];
e[3] = state[3];

D = sqrt( state[0]*state[0]+state[1]*state[1]+alpha_u*u*u);
P = 0.5*D*D;
k[0]=k[1]=0.0;

step = 0;
while(step < MAX_STEP)
{
  ym = M_PI/30.0*sin(t);

  if (flag == 1 || flag == 3){ // Tracking
     fprintf(pfile,"%8.3f %10.5f %8.3f %8.3f\n",t,state[0]*rad,↘
     ym*rad,u);//Tracking Problem
     printf("%8.3f %10.5f %8.3f %8.3f %8.3f\n",t,state[0]*rad,↘
     ym*rad,state[2],u);//Tracking Problem
  }else if (flag == 0 || flag == 2){ // Set Point
     fprintf(pfile,"%8.3f %10.5f %8.3f %8.3f\n",t,state[0]*rad↘
     ,0.0,u);    //Set Point Problem
     printf("%8.3f %8.3f %8.3f %8.3f\n",t,state[0]*rad,0.0,u); ↘
       //Set Point Problem
  }

  if (fabs(state[0]) > 60.0/rad ){
     if (fabs(state[0])> 0.5*M_PI &&
         fabs(state[0])< 1.5*M_PI & state[1] < 0.0){
       u = -10.0;
     }else if (fabs(state[0])> 0.5*M_PI &&
         fabs(state[0])< 1.5*M_PI & state[1] >= 0.0){
             u = 10.0;
     }else {
       u = 0.0;
```

```
  }

  Pole->NextState(u);
  Pole->GetState(state);
  D=sqrt(state[0]*state[0]+state[1]*state[1]+alpha_u*u*u);

  if (flag == 1 | flag == 3){
    e[0] = state[0]-ym;
    e[1] = state[1]-M_PI/30.0*cos(t);
  }else if (flag == 0| flag == 2){
    e[0] = state[0];
    e[1] = state[1];
  }
  e[2] = state[2];
  e[3] = state[3];

}else{
    old_u=u;
    if (argv[4][0]=='f' && argv[4][1]=='c' ){ // fcaps
      u = FCS->FIS(e);
    }else{ //lcaps
      u = k[0]*e[0]+k[1]*e[1];
    }
    u = sat(u,20.0);

    du = u-old_u;
    if (du >= 0.0 ) {du = 1.0;} else { du = -1.0;}

    Pole->NextState(u);
    Pole->GetState(state);

    if (flag == 1 || flag == 3){
       e[0] = state[0]-ym;
       e[1] = state[1]-M_PI/30.0*cos(t);
     }else if (flag == 0 || flag == 2){
       e[0] = state[0];
       e[1] = state[1];
     }

     e[2] = state[2];
```

```
            e[3] = state[3];

            old_D=D;
            D = sqrt(e[0]*e[0]+e[1]*e[1]+alpha_u*u*u);
            dD = D-old_D;
            if (dD >= 0.0 ) {dD = 1.0;} else { dD = -1.0;}

            oldP=P;
            P = 0.5 * D * D;
            f =  0.0;
            if ( P > oldP ) { f = 1.0;}

            if (argv[4][0]=='f' && argv[4][1]=='c' ){ // fcaps
              for(i=0;i<FCS->num_rules;i++){
                if (argv[1][0]=='f' & argv[1][1]=='d'){
                FCS->w[i] -= lr*f*D*dD/du*FCS->psi[i]*dt;
              }else{
                FCS->w[i] -= lr*D*dD/du*FCS->psi[i]*dt;
              }
            }else{  //lcaps
               if (argv[1][0]=='f' & argv[1][1]=='d'){
                 k[0] -= lr*f*D*dD/du*e[0]*dt;
                 k[1] -= lr*f*D*dD/du*e[1]*dt;}
               else{
                 k[0] -= lr*D*dD/du*e[0]*dt;
                 k[1] -= lr*D*dD/du*e[1]*dt;
               }
            }
          }
        }
        step++;
        t +=dt;
    }

    fclose(pfile);
    return EXIT_SUCCESS;
}
```

B TAC/FEF

```
#include <iostream.h>
```

```
#include <stdlib.h>
#include <ipendulum.h>
#include <myfunc.h>


#define MAX_STEP 4000
#define ns 4 // state size

int main(int argc, char *argv[]){

  int i,j,step;
  bool lie=true;

  float t,  state[ns],
    // state[0] : angle
    // state[1] : angle velocity
    // state[2]: position of the cart
    // state[3]: velocity of cart
    u,old_u,du,mindu=0.2,L,J,Jdot,Lu,
    e[ns],e_dot[ns],old_e[ns],old_e_dot[ns],
    dsdotdu[ns],th0,x0,w[ns],dw[ns],
    k=10.0,lrw,lrq,detH,
    l, // "lie" detector
    S; // "stuck" detector

  float H[ns*ns],Q[ns*ns];

  float I[ns*ns]={1.0,0.0,0.0,0.0,
                  0.0,1.0,0.0,0.0,
                  0.0,0.0,1.0,0.0,
                  0.0,0.0,0.0,1.0};

  IPendulum * Pole = new IPendulum;
  Pole->GetState(state);

  if (argc < 6){
    printf("Syntax: %s theta0 x0 lrw lrq k \n",argv[0]);
    return 0;
  }
```

```
th0=atof(argv[1])/rad;
x0=atof(argv[2]);
lrw = atof(argv[3]);
lrq = atof(argv[4]);
k = atof(argv[5]);

state[0] = th0;
state[1] =  0.0;
state[2] = x0;
state[3] =  0.0;

Pole->SetState(state);
t = 0.0;
u = 0.0;

FILE * pfile;
char fname[256];

sprintf(fname,"%s%3.1f-%3.1f-%6.4f-%6.4f-%6.4f.dat","tac",
        state[0]*rad,state[2],lrw,lrq,k);

pfile = fopen(fname,"w");

//Setting initial controller parameters
w[0]=50.0;w[1]=5.0;w[2]=5.0;w[3]=5.0;

//Setting initial evaluation function parameters
cpyvec(I,ns*ns,Q);
mvec(10.0,Q,ns*ns);

for(i=0;i<ns;i++){
  for(j=0;j<ns;j++){
    H[i*ns+j]=(w[i]*w[j]+Q[i*ns+j]);
  }
}

step = 0;

while(step < MAX_STEP){
```

```
// Get state
Pole->GetState(state);
cpyvec(state,ns,e);

if (step==0) {
  cpyvec(e,ns,old_e);
  old_u=u;
}

e_dot[0] = e[1]; e_dot[1] = (e[1]-old_e[1])/dt;
e_dot[2] = e[3]; e_dot[3] = (e[3]-old_e[3])/dt;

if (step==0) {cpyvec(e_dot,ns,old_e_dot);}

du=u-old_u;
if (fabs(du)<=mindu && du >=0) du = mindu;
if (fabs(du)<=mindu && du <0) du = -mindu;
old_u=u;

dsdotdu[0]=0.0;
dsdotdu[1]=(e_dot[1]-old_e_dot[1])*dt/du;
dsdotdu[2]=0.0;
dsdotdu[3]=(e_dot[3]-old_e_dot[3])*dt/du;

// ********** Evaluate Q ***************

// Setting H = ww' + Q
for(i=0;i<ns;i++){
  for(j=0;j<ns;j++){
    H[i*ns+j]=(w[i]*w[j]+Q[i*ns+j]);
  }
}

detH= H[0*ns+0]*(H[1*ns+1]*H[2*ns+2]-H[2*ns+1]*H[1*ns+2])
      -H[0*ns+1]*(H[1*ns+0]*H[2*ns+2]-H[2*ns+0]*H[1*ns+2])
      +H[0*ns+2]*(H[1*ns+0]*H[2*ns+1]-H[2*ns+0]*H[1*ns+1]);

J=sqrvec(old_e,H,old_e,ns);
Jdot = sqrvec(e_dot,H,old_e,ns)+sqrvec(old_e,H,e_dot,ns);
Lu=old_u*old_u+sqrvec(old_e,Q,old_e,ns);
```

```
L = Jdot+k*Lu;

// Tuning Q

if ((fabs(detH)>1.0) &&
    ((J<0.0 && Jdot >0.0) ||(J>0.0 && Jdot <0.0)) &&
    ((old_e[0]>0.0 && e_dot[0]<0.0 && old_e[2]>0.0 && e_dot↘
    [2]<0.0)||
     (old_e[0]>0.0 && e_dot[0]<0.0 && old_e[2]<0.0 && e_dot↘
     [2]>0.0)||
     (old_e[0]<0.0 && e_dot[0]>0.0 && old_e[2]>0.0 && e_dot↘
     [2]<0.0)||
     (old_e[0]<0.0 && e_dot[0]>0.0 && old_e[2]<0.0 && e_dot↘
     [2]>0.0))){
  lie=false;
  l=S=0.0;
}else{
  lie=true;
  l=1.0;
  if (fabs(L)<1.0) {
    S=sgn(L);
  }else{
   S=L;
  }
}

for(i=0;i<ns;i++){
  for(j=0;j<ns;j++){
          Q[i*ns+j]-=l*S*1.0e-6*lrq*(e_dot[i]*old_e[j]+old_e[i↘
           ]*e_dot[j]+k*old_e[i]*old_e[j])*dt;
   }
}

//******* Evaluate w **************

// Setting H = ww' + Q
for(i=0;i<ns;i++){
  for(j=0;j<ns;j++){
    H[i*ns+j]=(w[i]*w[j]+Q[i*ns+j]);
  }
```

```
}

J=sqrvec(old_e,H,old_e,ns);
Jdot = sqrvec(e_dot,H,old_e,ns)+sqrvec(old_e,H,e_dot,ns);
Lu=old_u*old_u+sqrvec(old_e,Q,old_e,ns);
L = Jdot+k*Lu;

// Tuning w
for (i = 0;i< ns;i++){
  dw[i]=2.0*(e_dot[i]*w[i]*old_e[i]+k*old_u*old_e[i]);
  for(j=0;j<ns;j++){
    if (j!=i){
            dw[i]+=2.0*w[j]*(old_e[i]*e_dot[j]+e_dot[i]*old_e[j↘
             ]);
    }
  }
  dw[i]+=old_e[i]*(sqrvec(dsdotdu,H,old_e,ns)+sqrvec(old_e,H,↘
   dsdotdu,ns));
  w[i]-=1.0e-8*lrw*L*dw[i]*dt;
 }

 w[0]=sat2(w[0],1.0,60.0);
 w[1]=sat2(w[1],1.0,10.0);
 w[2]=sat2(w[2],1.0,10.0);
 w[3]=sat2(w[3],1.0,10.0);

 cpyvec(e,ns,old_e);
 cpyvec(e_dot,ns,old_e_dot);

 //********* Determine and Execute Action

 u=0.0;
 for(i=0;i<ns;i++){
   u+= w[i]*e[i];
 }
 u = sat(u,20.0);

 if (lie){
   printf("%8.3f %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f - Q ↘
    is tuned !\n",t,e[0]*rad,e[2],u,w[0],w[1],w[2],w[3]);
```

```
    }else{
      printf("%8.3f %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f\n",t↘
      ,e[0]*rad,e[2],u,w[0],w[1],w[2],w[3]);
    }

    // Executing action
    Pole->NextState(u);

    if (fabs(e[0]) > 90.0/rad) break;
    fprintf(pfile,"%8.3f %8.3f %8.3f %8.3f\n",t,state[0]*rad,state↘
    [2],u);

    step++;
    t +=dt;
  }
  fclose(pfile);
  return EXIT_SUCCESS;
}
```

C Cart-pole dynamics

```
#include "ipendulum.h"

#define g 9.8    // gravity acc.
#define mc 1.0   // mass of cart
#define m 0.1    // mass of pole
#define total_m (mc+m)
#define l 0.5  // a half of pole length
#define mu_c 0.0 // coefficient of friction
#define mu_p 0.0 // coefficient of friction

IPendulum::IPendulum(){
  state[0]=state[1]=
  state[2]=state[3]=0.0;
  angle_acc = 0.0;
  pos_acc = 0.0;
  Mc = mc;
  Mp = m;
  M = Mc+Mp;
  Lp = l;
}
```

```
void IPendulum::SetState(float * s){
        int i;
        for (i=0;i<4;i++){state[i] = s[i];}
}

void IPendulum::GetState(float * s){
        int i;
        for (i=0;i<4;i++){s[i]=state[i];}
}

float IPendulum::theta2dot(float theta,float theta_dot, float x_dot
, float u)
{
  float y2dot,temp,costheta, sintheta;

  costheta = cos(theta);
  sintheta = sin(theta);
  temp = Lp*( 4.0/3.0-Mp*costheta*costheta/M);

  y2dot=(g*sintheta+costheta*(-u-Mp*Lp*theta_dot*theta_dot*sintheta
  +mu_c*sgn(x_dot))/M-mu_p*theta_dot/(Mp*Lp))/temp;

  return y2dot;
}

float IPendulum::x2dot(float theta,float theta_dot,float theta_acc,
float x_dot, float u)
{
  float pos2dot,costheta, sintheta;

  costheta = cos(theta);
  sintheta = sin(theta);

  pos2dot = (u+Mp*Lp*(theta_dot*theta_dot*sintheta-theta_acc*
   costheta)-mu_c*sgn(x_dot))/M;

  return pos2dot;
}
```

```
void IPendulum::NextState(float force)
{
    // 4th-order Runge-Kutta Integration Method
    float  h6,h2,h,S[4],k1,k2,k3,k4,
           l1,l2,l3,l4,m1,m2,m3,m4,n1,n2,n3,n4,theta_acc;

    h = dt;h2 = dt/2.0;h6 = dt/6.0;

    k1 = state[1];
    theta_acc = theta2dot(state[0],state[1],state[3],force);
    l1 = theta_acc;
    m1 = state[3];
    n1 = x2dot(state[0],state[1],theta_acc,state[3],force);

    S[0] = state[0]+h2*k1;
    S[1] = state[1]+h2*l1;
    S[2] = state[2]+h2*m1;
    S[3] = state[3]+h2*n1;
    k2 = S[1];
    theta_acc = theta2dot(S[0],S[1],S[3],force);
    l2 = theta_acc;
    m2 = S[3];
    n2 = x2dot(S[0],S[1],theta_acc,S[2],force);

    S[0] = state[0]+h2*k2;
    S[1] = state[1]+h2*l2;
    S[2] = state[2]+h2*m2;
    S[3] = state[3]+h2*n2;
    k3 = S[1];
    theta_acc = theta2dot(S[0],S[1],S[3],force);
    l3 = theta_acc;
    m3 = S[3];
    n3 = x2dot(S[0],S[1],theta_acc,S[2],force);

    S[0] = state[0]+h*k3;
    S[1] = state[1]+h*l3;
    S[2] = state[2]+h*m3;
    S[3] = state[3]+h*n3;
    k4 = S[1];
    theta_acc = theta2dot(S[0],S[1],S[3],force);
```

```
    l4 = theta_acc;
    m4 = S[3];
    n4 = x2dot(S[0],S[1],theta_acc,S[2],force);

    state[0] += h6*(k1+2.0*k2+2.0*k3+k4);
    state[1] += h6*(l1+2.0*l2+2.0*l3+l4);
    state[2] += h6*(m1+2.0*m2+2.0*m3+m4);
    state[3] += h6*(n1+2.0*n2+2.0*n3+n4);
}
```

D Gaussian Fuzzy Inference System

```
#include <math.h>
#include "gaussfuzzy.h"

GaussFuzzy::GaussFuzzy( float COI[],
            float SD[], int POS[], int s0, int s1){
  int i,j,k,l,no_rule=0;
  num_state = 2;

  num_rules = s0*s1;
  num_labels = s0+s1;

  map0 = new int[num_rules];
  map1 = new int[num_rules];

  psi = new float[num_rules];
  w = new float[num_rules];

  coi = new float[num_labels];
  sd = new float[num_labels];
  pos = new int[num_labels];

  for(i=0;i<s0;i++)
    for(j=s0;j<s0+s1;j++){
       map0[no_rule]= i;
       map1[no_rule]= j;
       w[no_rule] = 0.0;
       no_rule++;
  }
```

```
  for(i=0;i<num_labels;i++){
    coi[i]=COI[i];
    sd[i]=SD[i];
    pos[i]=POS[i];
  }
}

float GaussFuzzy::Gauss( float x, float c, float sd, int pos){

  float m;
  m = exp(-((x-c)/sd*(x-c)/sd) );
  if ( (x < c  && pos == -1 ) || (x > c && pos == 1) )
    m = 1.0;
  return m;
}

#define rad  57.2957795130

float GaussFuzzy::FIS(float * s){

  float F;
  int i;

  tot_psi = 0.0;
  for(i = 0;i < num_rules;i++){
   psi[i]= Gauss(s[0],coi[map0[i]]/rad,sd[map0[i]]/rad,pos[map0[i]])*Gauss(s[1],coi[map1[i]]/rad,sd[map1[i]]/rad,pos[map1[i]]);
   tot_psi += psi[i];
  }

  F = 0.0;
  for(i = 0;i < num_rules;i++){
    psi[i] /= tot_psi;
    F += psi[i]*w[i];
  }
  return F;
}

void GaussFuzzy::SetPar(float par){
  for(int i = 0;i < num_rules;i++){w[i] = par;}
```

```
}
```

E Supporting Functions

```
void statecpy(float * s, int l, float * d){
  for(int i=0;i<l;i++) d[i] = s[i];
}
float sat(float x, float b){
  float v;
  if (fabs(x) < b) {v = x;}
   if (x <= -b ) {v = -b;}
  if (x >= b ) {v = b;}
  return v;
}
float sat2(float x, float a, float b){
  float v;
  if (x > a && x < b) {v = x;}
   if (x <= a ) {v = a;}
  if (x >= b ) {v = b;}
  return v;
}
float sgn(float x){
  float s;
  if (x >= 0.0){s=1.0;}
  else {s = -1.0;}
  return s;
}
void cpyvec(float * s, int l, float * d){
  for(int i=0;i<l;i++)
    d[i] = s[i];
}
float sqrvec(float * x, float * G, float *y, int n){
  int i,j;
  float q = 0.0;
  for (i=0;i<n;i++){
    for (j=0;j<n;j++){
      q += x[i]*G[i*n+j]*y[j];
    }
  }
  return q;
}
```

```
void mvec(float a, float * q, int n){
  for (int j=0;j<n;j++){
    q[j]= a*q[j];
  }
}
```

F Control of ODE virtual cart-pole plant

```
#include <ode/ode.h>
#include <drawstuff/drawstuff.h>
#include <iostream.h>

// select correct drawing functions

#ifdef dDOUBLE
#define dsDrawBox dsDrawBoxD
#define dsDrawSphere dsDrawSphereD
#define dsDrawCylinder dsDrawCylinderD
#define dsDrawCappedCylinder dsDrawCappedCylinderD
#endif

#define dt 0.01

#define zoffset 1.1 // height of pilar (in meter)
static dReal base_length=6.0;
#define base_thick 0.1
#define rad 57.2957795130

#define CLENGTH 0.2      // cart length -> y axis (blue)
#define CWIDTH 0.3       // cart width -> x axis (red)
#define CHEIGHT 0.14     // cart height
#define CSTARTZ zoffset+0.5*(base_thick+CHEIGHT)  // zpos of cart ↘
 from base
#define CMASS 1          // cart mass

#define aLENGTH 0.25     // arm length
#define aWIDTH 0.02      // arm width
#define aHEIGHT 0.02     // arm height
#define aSTARTZ (CSTARTZ+0.5*CHEIGHT) // zpos of arm from base
#define aMASS 0.05       // arm mass
```

```
#define PLENGTH 0.02     // pole length
#define PWIDTH 0.02      // pole width
#define PHEIGHT 1.0      // pole height
#define PSTARTZ (CSTARTZ+0.5*CHEIGHT+0.5*PHEIGHT)// zpos of pole ↘
 from base
#define PMASS 0.1  // pole mass

static dWorldID world;
static dSpaceID space;
static dBodyID body[4];
static dGeomID box[4];
static dJointID joint[5];
static dJointGroupID contactgroup;
static dGeomID ground;
static dGeomID pilar;
static dSpaceID cp_space;
static  dReal cppos[3][3];
static  dReal cpsize[3][3];
static  dReal cpmass[3];
static dReal alpha=0.0;

struct MyObject {
  dBodyID body[3];  // the bodies
  dGeomID geom[3];  // geometries representing the bodies
};
static MyObject Base;
static dSpaceID base_space;
static  dReal bpos[3][3];
static  dReal bsize[3][3];

static const dReal *c_pos; // cart position
static const dReal *a_pos; // arm pos.
static const dReal *p_pos; // pole pos.

#define ns 4
static int con =-1;
static dReal t,
  state[ns],old_state[ns],u,old_u,du,dsdotdu[ns],
  e[ns],old_e[ns],e_dot[ns],dw[ns],
  k=10.0,theta0=0.0,x0=0.0,theta00,x00,
```

```
  xsp=0.0, // desired x
  lrw,lrq,detH,alpha_u=0.001f;

static bool swing=false;

FILE * pfile;
char fname[256];

static dReal w0[ns];
static dReal w[ns];
static dReal H0[4*4];
static dReal H[4*4];
static dReal Q[4*4];
static dReal Q0[ns*ns],T[ns*ns];

static dReal I[ns*ns]=
          {1.0,0.0,0.0,0.0,
           0.0,1.0,0.0,0.0,
           0.0,0.0,1.0,0.0,
           0.0,0.0,0.0,1.0};

static float front1_xyz[] = {4.401f,0.01f,3.3100f};
static float front1_hpr[] = {-180.0000f,-25.5000f,0.0000f};

static float front2_xyz[] = {2.5600f,3.7502f,2.3100f};
static float front2_hpr[] = {-131.0000f,-22.5000f,0.0000f};

static float front3_xyz[] = {2.259f,-4.097f,2.090f};
static float front3_hpr[] = {126.5f,-10.5f,0.0f};

static float far_xyz[] = {6.0f,-0.0699f,2.4300f};
static float front_xyz[] = {3.4808f,0.0001f,2.4300f};
static float front_hpr[] = {180.0000f,-25.0000f,0.0000f};

static void nearCallback (void *data, dGeomID o1, dGeomID o2){
  int i,n,numc;
  dBodyID b1 = dGeomGetBody(o1);
  dBodyID b2 = dGeomGetBody(o2);

  const int N = 10;
```

```
  dContact contact[N];
  n = dCollide (o1,o2,N,&contact[0].geom,sizeof(dContact));
  if (n > 0) {
    for (i=0; i<n; i++) {
      contact[i].surface.mode =
      dContactSoftERP | dContactSoftCFM | dContactApprox1;
      if (o1== ground || o2 == ground ){
        contact[i].surface.mu = 0.5;
      }else{
        contact[i].surface.mu = 0.00001;
      }
      contact[i].surface.soft_erp = 0.9;
      contact[i].surface.soft_cfm = 0.1;
      dJointID c=dJointCreateContact (world,contactgroup,&contact[i↘
      ]);
      dJointAttach (c,dGeomGetBody(contact[i].geom.g1),
                    dGeomGetBody(contact[i].geom.g2));
    }
  }

  if (numc = dCollide (o1,o2,N,&contact[0].geom,sizeof(dContact))) ↘
  {
    dMatrix3 RI;
    dRSetIdentity (RI);
    const dReal ss[3] = {0.02,0.02,0.02};
    for (i=0; i<numc; i++) {
      dJointID c=dJointCreateContact (world,contactgroup,contact+i)↘
      ;
      dJointAttach (c,b1,b2);
    }
  }

}


void mvec(dReal a, dReal * q, int n){
    for (int j=0;j<n;j++){
      q[j]= a*q[j];
    }
}
```

```
dReal sat(dReal x, dReal b){
  dReal v;
  v= x;
  if (fabs(x) < b) {v = x;}
  if (x <= -b ) {v = -b;}
  if (x >= b ) {v = b;}
  return v;
}

dReal sat2(dReal x, dReal a, dReal b){
  dReal v;
  if (x > a && x < b) {v = x;}
  if (x <= a ) {v = a;}
  if (x >= b ) {v = b;}
  return v;
}

void statecpy(dReal * s, int l, dReal * d){
  int i;
  for(i=0;i<l;i++) d[i] = s[i];
}

dReal sqrvec(dReal * x, dReal * G, dReal *y, int n){
  int i,j;
  dReal Q;
  dReal * q = (dReal *)malloc(n*sizeof(dReal));

  for (i=0;i<n;i++){
          q[i] = 0.0;
    for (j=0;j<n;j++){
      q[i]+= G[i*n+j]*y[j];
    }
  }
  Q = 0.0;
  for (i=0;i<n;i++){
    Q+=x[i]*q[i];
  }
  free(q);
  return Q;
```

```
}

static void init_sim(){
  int i,j;
  dMatrix3 R;
  dReal lp;
  dMass m;

  lp= 0.5*PHEIGHT; // a half length of POLE

  dJointGroupEmpty (contactgroup);

  for (i=0;i<3;i++){
    dBodyDestroy(body[i]);
    dGeomDestroy(box[i]);
    dBodyDestroy(Base.body[i]);
    dGeomDestroy(Base.geom[i]);
  }
  dGeomDestroy(pilar);

  dJointDestroy(joint[0]);
  dJointDestroy(joint[1]);
  dJointDestroy(joint[2]);
  dJointDestroy(joint[3]);
  dJointDestroy(joint[4]);

  // SIZE and POS of CART-POLE

  //CART Size
  cpsize[0][0]=CLENGTH;
  cpsize[0][1]=CWIDTH;
  cpsize[0][2]=CHEIGHT;

  //ARM Size
  cpsize[1][0]=aLENGTH;
  cpsize[1][1]=aWIDTH;
  cpsize[1][2]=aHEIGHT;

  //POLE Size
  cpsize[2][0]=PLENGTH;
```

```
cpsize[2][1]=PWIDTH;
cpsize[2][2]=PHEIGHT;

//CART Pos
cppos[0][0]=0.0;
cppos[0][1]=x0;
cppos[0][2]=CSTARTZ-0.5*base_length*sin(alpha);

//ARM Pos
cppos[1][0]=0.5*(aLENGTH-CLENGTH);
cppos[1][1]=x0;
cppos[1][2]=cppos[0][2]+0.5*cpsize[0][2];

//POLE Pos
cppos[2][0]=(aLENGTH-CLENGTH)+0.5*CLENGTH;
cppos[2][1]=cppos[0][1]+lp*sin(theta0);
cppos[2][2]=cppos[0][2]+0.5*CHEIGHT+lp*cos(theta0);

//MASS
cpmass[0]=CMASS;
cpmass[1]=aMASS;
cpmass[2]=PMASS;

//CART-POLE Space
cp_space = dSimpleSpaceCreate (space);
dSpaceSetCleanup (cp_space,0);



//CREATE BODY and GEOM of CART-POLE
dMassSetZero(&m);
for (i=0;i<3;i++){
  body[i]=dBodyCreate (world);
  if (i==2){
    dRSetIdentity (R);
    dRFromAxisAndAngle (R,1,0,0,-theta0);
    dBodySetRotation (body[i],R);
  }
  dBodySetPosition (body[i],cppos[i][0],cppos[i][1],cppos[i][2]);
  dMassSetBox (&m,1200.0f,cpsize[i][0],cpsize[i][1],cpsize[i][2])↘
   ;
```

```
  dMassAdjust (&m, cpmass[i]);
  dBodySetMass (body[i],&m);
  box[i] = dCreateBox (0, cpsize[i][0], cpsize[i][1], cpsize[i][2]);
  dGeomSetBody (box[i], body[i]);
  dSpaceAdd (cp_space, box[i]);
}

// cart-pole joint
joint[0] = dJointCreateHinge (world,0);
dJointAttach (joint[0], body[0], body[1]);
a_pos = dBodyGetPosition (body[1]);
dJointSetHingeAnchor(joint[0], a_pos[0], a_pos[1], a_pos[2]);
dJointSetHingeAxis(joint[0],1,0,0);

joint[1] = dJointCreateHinge (world,0);
dJointAttach (joint[1], body[1], body[2]);
p_pos = dBodyGetPosition (body[2]);
dJointSetHingeAnchor(joint[1], p_pos[0], a_pos[1], a_pos[2]);
dJointSetHingeAxis(joint[1],1,0,0);

// BASE SIZE and POS

bsize[0][0]=CLENGTH;
bsize[0][1]=base_length;
bsize[0][2]=base_thick;

bsize[1][0]=0.02;
bsize[1][1]=base_length;
bsize[1][2]=base_thick+0.02;

bsize[2][0]=0.02;
bsize[2][1]=base_length;
bsize[2][2]=base_thick+0.02;

bpos[0][0]=0.0;
bpos[0][1]=0.0;
bpos[0][2]=zoffset -0.5*base_length*sin(alpha);

bpos[1][0]=-(0.5*CLENGTH+0.5*bsize[1][0]+0.0000);
bpos[1][1]=0.0;
```

```
bpos[1][2]=bpos[0][2]+0.01;

bpos[2][0]=(0.5*CLENGTH+0.5*bsize[2][0]+0.0000);
bpos[2][1]=0.0;
bpos[2][2]=bpos[0][2]+0.01;

//BASE Space
base_space = dSimpleSpaceCreate (space);
dSpaceSetCleanup (base_space,0);

//CREATE BODY and GEOM of BASE
for (i=0;i<3;i++){
  Base.body[i] = dBodyCreate (world);
  dRSetIdentity (R);
  dRFromAxisAndAngle (R,1,0,0,alpha);
  dBodySetRotation (Base.body[i],R);
  dBodySetPosition (Base.body[i],bpos[i][0],bpos[i][1],bpos[i↘
  ][2]);
  dMassSetBox (&m,1.0f,bsize[i][0],bsize[i][1],bsize[i][2]);
  dMassAdjust (&m,1.0f*bsize[i][0]*bsize[i][1]*bsize[i][2]);
  dBodySetMass (Base.body[i],&m);
  Base.geom[i] = dCreateBox (0,bsize[i][0],bsize[i][1],bsize[i↘
  ][2]);
  dGeomSetBody (Base.geom[i],Base.body[i]);
  dSpaceAdd (base_space,Base.geom[i]);
}

joint[2] = dJointCreateHinge (world,0);
dJointAttach (joint[2],0,Base.body[0]);
dJointSetHingeAnchor(joint[2],bpos[0][0],base_length*0.5,zoffset)↘
;
dJointSetHingeAxis(joint[2],1,0,0);

joint[3] = dJointCreateHinge (world,0);
dJointAttach (joint[3],0,Base.body[1]);
dJointSetHingeAnchor(joint[3],bpos[0][0],base_length*0.5,zoffset+↘
bsize[1][2]*0.5);
dJointSetHingeAxis(joint[3],1,0,0);

joint[4] = dJointCreateHinge (world,0);
```

```
  dJointAttach (joint[4],0,Base.body[2]);
  dJointSetHingeAnchor(joint[4],bpos[0][0],base_length*0.5,zoffset+↘
   bsize[1][2]*0.5);
  dJointSetHingeAxis(joint[4],1,0,0);

  pilar = dCreateBox (space,CLENGTH+bsize[1][0]+bsize[2][0],0.1,↘
   zoffset-0.5*base_thick-0.5*base_length*sin(alpha));
  dGeomSetPosition (pilar,0.0,0.0,0.5*(zoffset-0.4*base_thick-0.5*↘
   base_length*sin(alpha)));

  //record initial state
  state[0]=theta0;
  state[1]=0.0;
  state[2]=x0;
  state[3]=0.0;

  //Setting initial controller parameters
  w[0]=50.0;w[1]=5.0;w[2]=5.0;w[3]=5.0;

  //Setting initial Q
  statecpy(I,ns*ns,Q);
  mvec(10.0,Q,ns*ns); // set Q=0

  for(i=0;i<ns;i++){
    w0[i]=w[i];
    for(j=0;j<ns;j++){
      H[i*ns+j]=(w[i]*w[j]+Q[i*ns+j]);
    }
  }
  statecpy(H,ns*ns,H0);
  statecpy(Q,ns*ns,Q0);

  u=0.0;

  printf("initial: theta=%3.0f deg; x= %3.1f m\n",theta0*rad,x0);
  printf("w=[%3.1f  \t%3.1f\t%3.1f\t%3.1f]\n\n",w[0],w[1],w[2],w↘
   [3]);
}

char locase (char c){
```

```
  if (c >= 'A' && c <= 'Z') return c - ('a'-'A');
  else return c;
}

static void command (int cmd){
  float xyz[3];
  float hpr[3];
  const dReal * pilar_pos;
  const dReal * base_pos;
  const dReal * q;

  static int p;
  dReal zpilar,a;
  dMass m;

  cmd = locase (cmd);

  if (cmd == 'v'){
    if (p < 3) {p++;} else {p=0;}
      if (p == 0){
        statecpy(front_xyz,3,xyz);
        statecpy(front_hpr,3,hpr);
      }else if (p==2){
        statecpy(front2_xyz,3,xyz);
        statecpy(front2_hpr,3,hpr);
      }else if (p==3){
        statecpy(front3_xyz,3,xyz);
        statecpy(front3_hpr,3,hpr);
      }else if (p==1){
        statecpy(far_xyz,3,xyz);
        statecpy(front_hpr,3,hpr);
      }
      dsSetViewpoint (xyz,hpr);
  }else if (cmd=='-'){
    xyz[0]+=0.5f;
    xyz[1]+=0.5f;
    dsSetViewpoint (xyz,hpr);
  }else if (cmd=='+'){
    xyz[0]-=0.5;
    xyz[1]-=0.5;
```

```
  dsSetViewpoint (xyz,hpr);
}else if (cmd== 'k') {
  dsGetViewpoint (xyz,hpr);
  printf("pos: %8.3f %8.3f %8.3f\n",xyz[0],xyz[1],xyz[2]);
  printf("view: %8.3f %8.3f %8.3f\n",hpr[0],hpr[1],hpr[2]);
}else if (cmd=='s'){
  theta0=180.0/rad;
  x0=x00;
}else if (cmd=='a'){
  theta0=theta00;
  x0=x00;
}else if (cmd=='z'){
  theta0=0.0;
  x0=0.0;
}else if (cmd=='i'){
  theta0+=5.0/rad;
}else if (cmd=='u'){
  theta0-=5.0/rad;
}else if (cmd=='p'){
  x0+=0.2;
}else if (cmd=='o'){
  x0-=0.2;
}else if (cmd=='j'){
  xsp=-1.0;
}else if (cmd=='l'){
        xsp=1.0;
}else if (cmd == 'c'){
  con = 1;
}else if (cmd == 'b' ){
  q = dBodyGetQuaternion (Base.body[0]);
  a = 2.0*acos(q[0]); // inclination angle of base
  base_pos=dBodyGetPosition(Base.body[0]);
  pilar_pos=dGeomGetPosition(pilar);
  zpilar =pilar_pos[2]-(0.5*base_length*tan(a+0.2/rad)-(zoffset-↘
   base_pos[2]));
  dGeomDestroy(pilar);
  pilar = dCreateBox (space,CLENGTH+bsize[1][0]+bsize[2][0],0.1,↘
   zpilar*2.0);
  dGeomSetPosition (pilar,0.0,0.0,zpilar);
}else if (cmd == 'n' ){
```

```
    q = dBodyGetQuaternion (Base.body[0]);
    a = 2.0*acos(q[0]); // inclination angle of base
    base_pos=dBodyGetPosition(Base.body[0]);
    pilar_pos=dGeomGetPosition(pilar);
    zpilar =pilar_pos[2]+(0.5*base_length*tan(a+0.1/rad)-(zoffset-↘
     base_pos[2]));
    dGeomDestroy(pilar);
    pilar = dCreateBox (space,CLENGTH+bsize[1][0]+bsize[2][0],0.1,↘
     zpilar*2.0);
    dGeomSetPosition (pilar,0.0,0.0,zpilar);
  }else if (cmd == 'd' ){ // throw a free box to the pole
    if (dBodyIsEnabled(body[3]) && dGeomIsEnabled(box[3])) {
      dBodyDestroy(body[3]);
      dGeomDestroy(box[3]);
    }
    body[3] = dBodyCreate (world);
    dBodySetPosition (body[3],0.95*CLENGTH,0.5,1.8*CSTARTZ);
    dMassSetBox (&m,500.0f,0.5*CLENGTH,0.5*CWIDTH,0.5*CHEIGHT);
    dBodySetMass (body[3],&m);
    box[3] = dCreateBox (space,0.5*CLENGTH,0.5*CWIDTH,0.5*CHEIGHT);
    dGeomSetBody (box[3],body[3]);
    dBodyAddForceAtRelPos(body[3],0.0,-100.0,0.0,0.0,0.0,0.0);
  }else if (cmd == 'e' ){
    if (dBodyIsEnabled(body[3]) && dGeomIsEnabled(box[3])) {
      dBodyDestroy(body[3]);
      dGeomDestroy(box[3]);
    }
  }else if (cmd == 'x'){
    con = -1;
  }else if (cmd == 'r'){
    init_sim();
  }
}

static void start(){
  float xyz[3];
  float hpr[3];
  statecpy(front_xyz,3,xyz);
  statecpy(front_hpr,3,hpr);
  dsSetViewpoint (xyz,hpr);
```

```
}

dReal sgn(dReal x){
  dReal s;
  if (x >= 0.0){s=1.0;}
  else {s = -1.0;}
  return s;
}

static void simLoop (int pause){
  int i,j;
  static int step;
  static int catch_pole=0;

  const dReal *Vx;
  const dReal *Va;
  const dReal *pole_pos;
  const dReal *cart_pos;
  const dReal * q;

  dReal a,x,v,uy,uz;
  static dReal steps_to_sw=0.0;
  static int direction;
  dReal mindu = 0.1; // minimum du
  dReal L,J,Jdot,Lu,l,S;

  if (!pause){

    dsSetColor (0,0,2);

    Vx =   dBodyGetLinearVel(body[0]);
    Va =   dBodyGetAngularVel(body[2]);
    cart_pos = dBodyGetPosition(body[0]);
    pole_pos = dBodyGetPosition(body[2]);

    q = dBodyGetQuaternion (Base.body[0]);
    a = 2.0*acos(q[0]); // inclination angle of base

    state[1] = -Va[0];
    if (step==0) {state[0] = theta0;}
```

```
else {state[0]+=state[1]*dt;}

x = cart_pos[1];
state[2] = x;
v = Vx[1];
state[3] = v;

// perturb by throwing a free box
if (step==650 || step==1000) command('d');
// perturb by changing the inclination angle of the base by 5 ↘
 deg counterclockwise
if (step>=850 && step <860) command('b');

if (con==1 && step==0 && state[0]== 180.0/rad) {
  if (x0<0){u= 20.0;}
  else {u= -20.0;}
} else if ( con == 1 && cos(state[0]) < cos(155.0/rad) ){
  if (state[0] < 180.0/rad && state[1] < 0.0 ) u=0.0;
  if (state[0] < 180.0/rad && state[1] > 0.0 ) u=10.0;
  if (state[0] > 180.0/rad && state[1] < 0.0 ) u=-10.0;
  if (state[0] > 180.0/rad && state[1] > 0.0 ) u=0.0;
} else if (con ==1  && cos(state[0])> cos(60.0/rad)){

  state[2]-=xsp; // change the desired cart position
  statecpy(state,ns,e);

  if (catch_pole==0) {       // First time pole enters the ↘
   operating angle region of TAC/FEF
    statecpy(e,ns,old_e);
    e_dot[0] = e[1];e_dot[1] = (e[1]-old_e[1])/dt;
    e_dot[2] = e[3];e_dot[3] = (e[3]-old_e[3])/dt;
  }
  catch_pole++;

  du=u-old_u;
  if (fabs(du)<=mindu && du >=0) du = mindu;
  if (fabs(du)<=mindu && du <0) du = -mindu;
  old_u=u;

  dsdotdu[0]=0.0;
```

```
dsdotdu[1]=(e[1]-old_e[1]-e_dot[1]*dt)/du;
dsdotdu[2]=0.0;
dsdotdu[3]=(e[3]-old_e[3]-e_dot[3]*dt)/du;

statecpy(e,ns,old_e);
e_dot[0] = e[1]; e_dot[1] = (e[1]-old_e[1])/dt;
e_dot[2] = e[3]; e_dot[3] = (e[3]-old_e[3])/dt;

// ********** Evaluate Q ***************

// Setting H = ww' + Q
for(i=0;i<ns;i++){
  for(j=0;j<ns;j++){
    H[i*ns+j]=(w[i]*w[j]+Q[i*ns+j]);
  }
}

detH= H[0*ns+0]*(H[1*ns+1]*H[2*ns+2]-H[2*ns+1]*H[1*ns+2])-H↘
 [0*ns+1]*(H[1*ns+0]*H[2*ns+2]-H[2*ns+0]*H[1*ns+2])+H[0*ns↘
 +2]*(H[1*ns+0]*H[2*ns+1]-H[2*ns+0]*H[1*ns+1]);

J=sqrvec(old_e,H,old_e,ns);
Jdot = sqrvec(e_dot,H,old_e,ns)+sqrvec(old_e,H,e_dot,ns);
L = Jdot+k*J;

if ((fabs(detH)>1.0) &&
    ((J < 0.0 && Jdot >0.0) || (J>0.0 && Jdot<0.0)) &&
     ((e[0]<0.0 && e_dot[0]>0.0 && e[2] <0.0 && e_dot[2]>0.0 ↘
     ) || (e[0] <0.0 && e_dot[0]>0.0 && e[2] >0.0 && e_dot↘
     [2]<0.0 ) || (e[0] >0.0 && e_dot[0]<0.0 && e[2] <0.0 &&↘
      e_dot[2]>0.0 ) || (e[0] >0.0 && e_dot[0]<0.0 && e[2] ↘
     >0.0 && e_dot[2]<0.0 ) )){
  l=S=0.0;
} else {
  l=1.0;
  if (fabs(L)<1.0) {
    S=sgn(L);
  }else{
    S=L;
  }
```

```
}

for(i=0;i<ns;i++){
  for(j=0;j<ns;j++){
    Q[i*ns+j]-=l*1.0e-6*lrq*S*(e_dot[i]*old_e[j]+old_e[i]*↘
     e_dot[j]+k*old_e[i]*old_e[j])*dt;
  }
}

//******* Evaluate w **************

// Setting H = ww' + Q
for(i=0;i<ns;i++){
  for(j=0;j<ns;j++){
    H[i*ns+j]=(w[i]*w[j]+Q[i*ns+j]);
    }
}

J=sqrvec(old_e,H,old_e,ns);
Jdot = sqrvec(e_dot,H,old_e,ns)+sqrvec(old_e,H,e_dot,ns);
Lu=old_u*old_u+sqrvec(old_e,Q,old_e,ns);
L = Jdot+k*J;

// Tuning w
for (i = 0;i< ns;i++){
  dw[i]=2.0*(e_dot[i]*w[i]*old_e[i]+k*old_u*old_e[i]);
  for(j=0;j<ns;j++){
    if (j!=i){
      dw[i]+=2.0*w[j]*(old_e[i]*e_dot[j]+e_dot[i]*old_e[j]);
    }
  }
  dw[i]+=old_e[i]*(sqrvec(dsdotdu,H,old_e,ns)+sqrvec(old_e,H,↘
   dsdotdu,ns));
  w[i] -=1.0e-8*lrw*L*dw[i]*dt;
}
w[0]=sat2(w[0],1.0,60.0);
w[1]=sat2(w[1],1.0,10.0);
w[2]=sat2(w[2],1.0,10.0);
w[3]=sat2(w[3],1.0,10.0);
```

```
  u=w[0]*e[0];
  for(i=1;i<ns;i++){
    u+= w[i]*e[i];
  }
  u = sat(u,20.0);
}else {
  u=0.0;
}

uy=(u+(CMASS+aMASS+PMASS)*9.8*sin(a))*cos(a); // horizontal ↘
 action
uz=(u+(CMASS+aMASS+PMASS)*9.8*sin(a))*sin(a); // vertical ↘
 action

if (con==-1 || cart_pos[0] < -CWIDTH*0.5 || cart_pos[0] > ↘
CWIDTH*0.5 ||  x < -base_length*0.5 || x > base_length*0.5){
  uz=uy=0.0;
}

//Apply Action to the Cart body
dBodyAddForceAtRelPos(body[0],0.0,uy,uz,0.0,0.0,0.0);

printf("%7.2f %7.3f %7.3f %7.3f %7.3f %7.3f %7.3f %7.3f\n↘
",t,e[0]*rad, e[2],u,w[0],w[1],w[2],w[3],a*rad);

dSpaceCollide (space,0,&nearCallback);
dWorldStep (world,dt);
dJointGroupEmpty (contactgroup);

// Save data for swinging up problem
if (swing && step <4000 && x > -base_length*0.5 && x < ↘
base_length*0.5){
  fprintf(pfile,"%8.3f %8.3f %8.3f %8.3f\n",t,state[0]*rad,↘
   state[2]*100.0,u*5.0);
}else // Save data for non-swinging up problem
if (!swing && step <4000 && fabs(state[0]) < 90.0/rad){
  fprintf(pfile,"%8.3f %8.3f %8.3f %8.3f\n",t,state[0]*rad,↘
   state[2],u);
}
step ++;
```

```
  t+=dt;
}else{
  step=0;
  t=0.0;
  fclose(pfile);
  pfile = fopen(fname,"w");
}

dsSetColor (188.0/255.0,143.0/255.0,143.0/255.0);//(0,1,1);
dsSetTexture (DS_WOOD);
dReal csides[3] = {CLENGTH,CWIDTH,CHEIGHT};
dsDrawBox (dBodyGetPosition(body[0]),dBodyGetRotation(body[0]),↘
 csides);
dsSetColor (1,0,0);
dReal asides[3] = {aLENGTH,aWIDTH,aHEIGHT};
dsDrawBox (dBodyGetPosition(body[1]),dBodyGetRotation(body[1]),↘
 asides);
dsSetColor (250.0/255.0,250.0/255.0,210.0/255.0);
dsSetColor(1.0,1.0,1.0);
dReal psides[3] = {PLENGTH,PWIDTH,PHEIGHT};
dsDrawBox (dBodyGetPosition(body[2]),dBodyGetRotation(body[2]),↘
 psides);

//free box
dsSetColor(1.0,1.0,1.0);
dReal osides[3] = {0.5*CLENGTH,0.5*CWIDTH,0.5*CHEIGHT};
dsDrawBox (dBodyGetPosition(body[3]),dBodyGetRotation(body[3]),↘
 osides);

dsSetColor (100.0/255.0,149.0/255.0,237.0/255.0);
dVector3 ss;
dGeomBoxGetLengths (pilar,ss);
dsDrawBox (dGeomGetPosition(pilar),dGeomGetRotation(pilar),ss);

for (i=0; i < 3; i++) {
  if (i==0) dsSetColor(1,1,224/255);
  else dsSetColor(0.333,1.0,0.9);
  dsDrawBox (dBodyGetPosition(Base.body[i]),dBodyGetRotation(Base↘
   .body[i]),bsize[i]);
}
```

```
}

int main (int argc, char **argv)
{
  int i;
  if (argc < 9){
      printf("\nSyntax: %s <-1,1>  base_length theta0 x0 lrw lrq k ↘
       alpha\n%s",argv[0],"\n  -1: control off, 1: control on \n ↘
       alpha : inclination angle of base (in rad)\n");
      return 0;
  }

  con = atoi(argv[1]);
  base_length=atof(argv[2]);
  theta0 = atof(argv[3])/rad;
  x0 = atof(argv[4]);
  lrw = atof(argv[5]);
  lrq = atof(argv[6]);
  k = atof(argv[7]);
  alpha=atof(argv[8])/rad;
  theta00=theta0;
  x00=x0;

  if (atof(argv[3])>=175.0 && atof(argv[3])<185.0) swing=true;
  sprintf(fname,"%s%3.1f-%3.1f-%3.1f-%3.1f-%4.0f-%3.1f.dat","tac",
          theta0*rad,x0,lrw,lrq,k,alpha);
  pfile = fopen(fname,"w");

  // setup pointers to drawstuff callback functions
  dsFunctions fn;
  fn.version = DS_VERSION;
  fn.start = &start;
  fn.step = &simLoop;
  fn.command = &command;
  fn.stop = 0;
  fn.path_to_textures = "textures";

  // create world
  world = dWorldCreate();
  space = dHashSpaceCreate (0);
```

```
  contactgroup = dJointGroupCreate (0);
  dWorldSetGravity (world,0,0,-9.8);
  ground = dCreatePlane (space,0,0,1,0);

  //CART-POLE BODY AND GEOM
  body[0] = dBodyCreate (world); // cart
  body[1] = dBodyCreate (world); // arm
  body[2] = dBodyCreate (world); // pole
  box[0] = dCreateBox (0,1.0,1.0,1.0); // cart
  box[1] = dCreateBox (0,1.0,1.0,1.0); //
  box[2] = dCreateBox (0,1.0,1.0,1.0);

  //CREATE FREE BOX BODY AND GEOM
  body[3] = dBodyCreate (world); // free box
  box[3] = dCreateBox (0,1.0,1.0,1.0); // free box
  dBodyDestroy(body[3]); //free box
  dGeomDestroy(box[3]);

  joint[0] = dJointCreateHinge (world,0);
  joint[1] = dJointCreateHinge (world,0);
  joint[2] = dJointCreateHinge (world,0);
  joint[3] = dJointCreateHinge (world,0);
  joint[4] = dJointCreateHinge (world,0);

  //Create BASE
  for (i=0;i<3;i++){
    Base.body[i] = dBodyCreate (world);
    Base.geom[i]=dCreateBox(0,1.0,1.0,1.0);
  }
  pilar = dCreateBox (space,1.0,1.0,1.0);

  init_sim();
  dsSimulationLoop (argc,argv,2*352,2*288,&fn);

  fclose(pfile);
  dJointGroupDestroy (contactgroup);
  dSpaceDestroy (space);
  dWorldDestroy (world);
  return 0;
}
```

REFERENCES

[1] V. Gullapalli, "Reinforcement learning and its application to control," Ph.D. dissertation, University of Massachusetts, Amherst, MA, 1992.

[2] K. J. Åström and B. Witternmark, *Adaptive Control*. Addison-Wesley Publishing Company, 1989.

[3] J.-J. E. Slotine and W. Li, *Applied nonlinear control*. Englewood Cliffs New Jersey: Prentice Hall, 1991.

[4] S. S. Sastry and A. Isidory, "Adaptive control of linearizable systems," *IEEE Transactions on Automatic Control*, vol. 34, pp. 1123–1131, 1989.

[5] L.-X. Wang, *A COURSE IN FUZZY SYSTEMS AND CONTROL*. New Jersey: Prentice-Hall International, Inc., 1997.

[6] G. S.-X. Cheng, Folsom, and Calif, "Model-free adaptive process control," 2000. [Online]. Available: http://www.freepatentsonline.com

[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[8] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Trans. Syst., Man, Cybern.*, vol. 13, no. 5, pp. 834–846, 1983.

[9] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[10] H. R. Berenji and P. Khedkar, "Learning and tuning fuzzy logic controllers through reinforcement," *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 724–740, 1992.

[11] ——, "Using fuzzy logic for performance evaluation in reinforcement learning," *International Journal of Approximate Reasoning*, no. 18, pp. 131–144, 1998.

[12] T. Landelius and H. Knutsson, "Reinforcement learning adaptive control and explicit criterion maximization," Computer Vision Laboratory, SE-581 83 Linköping, Sweden, Report LiTH-ISY-R-1829, 1996.

[13] E. Trulsson and L. Ljung, "Adaptive control based on explicit criterion minimization," *Automatica*, vol. 21, pp. 385–399, 1985.

[14] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in Neural Information Processing System 12*, pp. 1057–1063, 2000.

[15] L. C. Baird, "Residual algorithms: Reinforcement learning with function approximation," in *Proc. of the Twelfth Int. Conf. on Machine Learning*, M. Kaufman, Ed., 1995, pp. 30–37.

[16] G. J. Gordon, "Stable function approximator in dynamic programming," in *Proc. of the Twelfth Int. Conf. on Machine Learning*, M. Kaufman, Ed., 1995, pp. 261–268.

[17] D. P. Bertsekas and J. N. Tsikilis, *Neuro-Dynamic Programming.* Athena Scientific, 1996.

[18] C. W. Anderson, "Strategy learning with multilayer connectionist representations," in *Proc. the Fourt International Workshop on Machine Learning*, Irvine, CA, 1987, pp. 103–114.

[19] H. Kimura and S. Kobayashi, "An analysis of actor/critic algorithms using eligibility traces: Reinforcement learning with imperfect value functions," in *Proc. ICML-98*, 1998, pp. 278–286.

[20] L. Baird and A. Moore, "Gradient descent for general reinforcement learning," *Advances in Neural Information Processing Systems 11*, 1999.

[21] A. Zomaya, "Reinforcement learning for the adaptive control of nonlinear systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 2, pp. 357–363, 1994.

[22] L. Peshkin, "Reinforcement learning by policy search," Ph.D. dissertation, Brown University,Providence, RI, 2001.

[23] A. O. Esogbue, W. E. Hearnes, and Q. Song, "A reinforcement learning fuzzy controller for set-point regulator problems," in *Proc. of the FUZZ-IEEE '96 Conference*, vol. 3, New Orleans, LA, 1996, pp. 2136–2142.

[24] D. Prokhorov, R. Santiago, and D. Wunsch, "Adaptive critic designs: A case study for neurocontrol," *Neural Networks*, vol. 8(9), pp. 1367–1372, 1995.

[25] D. Prokhorov and D. Wunsch, "Adaptive critic designs," *IEEE Transactions on Neural Networks*, vol. 8(5), pp. 997–1007, 1997.

[26] D. Prokhorov, "Adaptive critic designs and their application," Ph.D. dissertation, Texas Tech University, 1997.

[27] P. J. Werbos, "Stable adaptive control using new critic designs," in *Proc. SPIE Vol. 3728, p. 510-579, Ninth Workshop on Virtual Intelligence/Dynamic Neural Networks*, T. Lindblad, M. L. Padgett, and J. M. Kinser, Eds., Mar. 1999, pp. 510–579.

[28] G. G. Lendaris and L. J. Schultz, "Controller design (from scratch) using approximate dynamic programming," in *Proc. of IEEE International Symposium on Intelligent Control (IEEE-ISIC) 2000*, Patras, Greece, July 2000.

[29] G. G. Lendaris, L. J. Schultz, and T. T. Shannon, "Adaptive critic design for intelligent steering and speed control of a 2-axle vehicle," in *Proc. of International Joint Conference on Neural Networks (IJCNN) 2000*, Italy, July 2000.

[30] T. T. Shannon and G. G. Lendaris, "A new hybrid critic training method for approximate dynamic programming," in *Proc. of the World Congress of the Systems Sciences 2000*, Toronto, Canada, July 2000.

[31] J. Si, A. Barto, W. Powell, and D. Wunsch, *Handbook of Learning and Approximate Dynamic Programming*. IEEE Press John Wiley and sons, Inc., 2004.

[32] A. G. Barto, "Some learning task from a control perspective," University of Massachusetts, Amherst, MA, COINS Technical Report 91-122, 1991.

[33] S. Horikawa, T. Furuhashi, S. Okuma, and Y. Uchikawa, "A fuzzy controller using a neural network and its capability to learn expert's control rules," in *Proc. Int. Conf. Fuzzy Logic Neural Networks*, Iizuka, Japan, July 1990, pp. 103–106.

[34] H. Nomura, I. Hayashi, and N. Wakami, "A self-tuning method of fuzzy control by descent method," in *Proc. IFSA'91, 1rst Int. Fuzzy Syst. Assoc. World Congress*, 1991, pp. 155–158.

[35] C. Lin and C. S. G. Lee, "Neural-network-based fuzzy logic control and decision system," *IEEE Trans. Comput.*, vol. 40, pp. 1320–1336, 1991.

[36] L. Jouffe, "Fuzzy inference system learning by reinforcement method," *IEEE Trans. Syst., Man, Cybern. C*, vol. 28, no. 3, pp. 338–355, 1998.

[37] M. I. Jordan, "Forward models: Supervised learning with a distal teacher," *Cognitive Science*, vol. 16, pp. 307–354, 1992.

[38] K. Ogata, *Modern Control Engineering*. Englewood Cliffs New Jersey: Prentice Hall, 1997.

[39] W. J. Rugh, *Linear System Theory*. Prentice-Hall International, Inc., 2nd edition, 1996.

[40] C.-T. Chen, *Linear System Theory and Design*. Holt, Rinehart, and Winston, Inc., 1984.

[41] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[42] J. C. Santamaria, R. R. Sutton, and A. Ram, "Experiment with reinforcement learning in problems with continuous state and action spaces," *Adaptive Behavior*, vol. 6, no. 2, pp. 163–218, 1998.

[43] S. K. Oh, W. Pedrycz, S. B. Rho, and T. C. Ahn, "Parameters estimation of fuzzy controller and its application to inverted pendulum," *Engineering Applications of Artificial Intelligence*, vol. 17, pp. 37–60, 2004.

[44] Y. Y. Nazaruddin, A. Naba, and T. H. Liong, "Modified adaptive fuzzy control system using universal supervisory controller," in *Proc. of SCI/ISAS 2000*, vol. IX, Orlando, USA, July 2000.

[45] F. Hsu and L. Fu, "A novel adaptive fuzzy variable structure control for a class of nonlinear uncertain systems via backstepping," *Fuzzy sets and systems*, vol. 122, pp. 83–106, 2001.

[46] C. Liang and J. Su, "A new approach to the design of fuzzy sliding mode controller," *Fuzzy sets and systems*, vol. 139, pp. 111–124, 2003.

[47] S. Tong and H. Li, "Direct adaptive fuzzy output tracking control of nonlinear systems," *Fuzzy sets and systems*, vol. 128, pp. 107–115, 2002.

[48] C.-K. Lin, "A reinforcement learning adaptive fuzzy controller for robots," *FUZZY sets and system*, vol. 137, pp. 339–352, 2003.

[49] J. Park, G. Park, S. Kim, and C. Moon, "Direct adaptive self-structuring fuzzy controller for nonaffine nonlinear system," *Fuzzy sets and systems*, vol. 153, pp. 429–445, 2005.

[50] A. Naba and K. Miyashita, "Tuning fuzzy controller using approximated evaluation function," in *Proc. of the 4th IEEE International Workshop WSTST05*, Muroran, Japan, May 2005, pp. 113–122.

[51] ——, "Gadient-based tuning of fuzzy controller with approximated evaluation function," in *Proc. of Eleventh International Fuzzy Systems Association (IFSA) World Congress*, Beijing, China, July 2005, pp. 671–676.

[52] ——, "FCAPS: Fuzzy controller with approximated policy search approach," *Journal of Adv. Comput. Intelligence and Intelligent Informatics*, vol. 1, no. 1, pp. 84–92, 2006.

[53] K. J. Åström and K. Furuta, "Swinging up a pendulum by energy control," *Automatica*, vol. 36, pp. 287–295, 2000.

[54] C. C. Chung and J. Hauser, "Nonlinear control of a swinging pendulum," *Automatica*, vol. 31, pp. 851–862, 1995.