Adaptive Control with Approximated Policy Search Approach

Agus Naba

Instrumentation and Measurement Laboratory, Faculty of Sciences,
Brawijaya University
Jl. Veteran Malang, Telp./Fax.: +62 (341) 575833/575834
e-mail: anaba@brawijaya.ac.id

Abstract. Most of existing adaptive control schemes are designed to minimize error between plant state and goal state despite the fact that executing actions that are predicted to result in smaller errors only can mislead to non-goal states. We develop an adaptive control scheme that involves manipulating a controller of a general type to improve its performance as measured by an evaluation function. The developed method is closely related to a theory of Reinforcement Learning (RL) but imposes a practical assumption made for faster learning. We assume that a value function of RL can be approximated by a function of Euclidean distance from a goal state and an action executed at the state. And, we propose to use it for the gradient search as an evaluation function. Simulation results provided through application of the proposed scheme to a pole-balancing problem using a linear state feedback controller and fuzzy controller verify the scheme's efficacy.

Keywords: *adaptive control; evaluation function; policy search approach; reinforcement learning; value function.*

1 Introduction

Every control method involves manipulating a controller to achieve prespecified control objective. In many control designs, the control objective is defined as to minimize output error (i.e., the error between a goal and an actual plant state), assuming that smaller error implies always better instantaneous control performance or immediate reward. However, in many control problems, executing actions that are predicted to result in smaller errors only can mislead to non-goal states. This is because smaller error does not always mean better control performance. In general, the error is more *instructional* than *evaluative*, and therefore, not universally suitable as a control performance measure.

In Reinforcement Learning (RL) [1], evaluative information about an action performance (i.e., an action value) is not readily available and not simply equivalent to the error. In contrast with a customary in control designs where the actions are rewarded based on the control performance measure defined a

Received February 20th, 2008, 1st Revised May 19th, 2008, 2nd Revised August 17th, 2009, Accepted for publication September 11th, 2009.

priori by an engineer, RL views that the actions must be rewarded by the environment, not by the engineer. The reward is usually a *weak* evaluative feedback 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. Applied to the control problems, the objective of RL is then defined as to improve the control performance, as evaluated by the value function, by generating appropriate actions. A certain defect of RL is that many time-consuming trials-and-errors are often required to learn the precise value function.

This paper addresses control design that involves manipulating the controller using a gradient method to improve its performance as measured by an evaluation function. In reality, many control problems can be modeled as a combination of simple sub-problems. Given a relatively simple problem, an 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, we assume that we can use the distance of the problem states to the goal state and actions as an *approximate 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.

Various kinds of adaptive method proposed in many literatures [2–11] might work as well as the method proposed in this paper. However, those methods require the plant model or its assumed structure to be available. This makes the control design inflexible for certain plants which are difficult to be represented by the assumed structure of the plant model. A preliminary research on adaptive control based on approximate evaluation function has been reported in [12–14]. But, it only focuses on solving adaptive control problem using a fuzzy controller. Using the same adaptive control design as reported in those preliminary research, this paper shows that the proposed adaptive control design can actually work using more general controller including the fuzzy controller.

The remainder of this paper is organized as follows: Section 2 briefly discusses the theories of RL. Section 3 presents the proposed scheme for tuning controller of a general type. Section 4 presents and discusses an application of the proposed scheme to a benchmark cart-pole balancing problem. Finally, Section 5 concludes the paper.

2 Reinforcement Learning

A model of RL is depicted in Figure 1. It 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 1 includes the *critic* by which it can also evaluate the actions it executes. The "plant" not only produces outputs but also rewards for the actions applied to it.

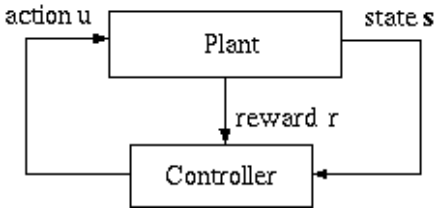

Figure 1 Model of reinforcement learning control problem.

At a discrete time step t, given a state $\mathbf{s}(t)$, the controller applies an action $u(t)$ to the plant and in the next time step $t + 1$ receives a *reward* $r(t+1)$. The goal of the controller is to find an optimal policy that determines a sequence of actions $\bar{u}(t) = \left(u(t), u(t+1), u(t+2), \cdots\right)$ maximizing the total amount of the discounted reward received in the long run:

$$V(\mathbf{s}(t)) = \max_{\bar{u}} \sum_{k=0}^{\infty} \gamma^k r(t+1+k), \tag{1}$$

which is referred to as a *value function*. $\gamma \in [0,1]$ denotes a discount factor against a future reward.

Comprehensive description of the terms "policy", "reward", "value" can be found in [1]. 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 controllers 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.

In this paper, we are concerned with the approaches the controller can use to learn the optimal policy, i.e., *value function approach* and *policy search approach*.

2.1 Value Function Approach

The equation (1) 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 $\bar{u}(t)$. This function is the solution of the following Bellman equation:

$$V(\mathbf{s}(t)) = \max_{\bar{u}} \left[r(t+1) + \gamma V(\upsilon(\mathbf{s}(t), u(t))) \right], \tag{2}$$

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

From (2), one can deduce the optimal policy:

$$u^*(s(t)) = \arg\max_{\bar{u}} \left[r(t+1) + \gamma V(\upsilon(s(t+1), u(t+1))) \right]. \tag{3}$$

When the controller is not given the plant dynamics function, the value function can be incrementally computed by using a *state-action-value function* (a.k.a. *Q-function*),

$$Q(\mathbf{s}(t), u(t)) = r(t+1) + \gamma Q(\mathbf{s}(t+1), u^*(\mathbf{s}(t+1))), \tag{4}$$

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

$$u^*(s(t+1)) = \arg\max Q(\mathbf{s}(t+1), u(t+1)).$$

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 [1]. 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 have to be chosen as well for exploration. Such action selection method is known as an ε-greedy action selection. Figure 2 describes how all these processes take place in the controller.

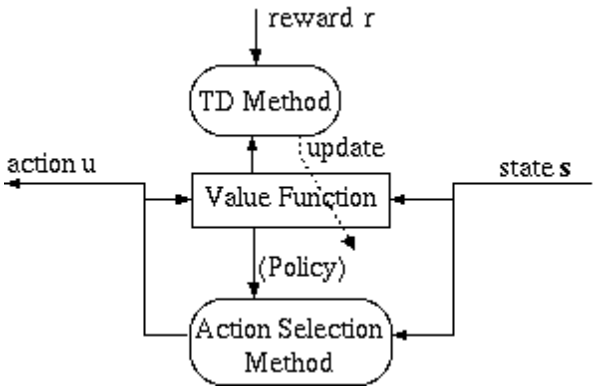

Figure 2 Value function approach.

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

$$Q(\mathbf{s}(t),u(t)) \leftarrow Q(\mathbf{s}(t),u(t)) + \alpha\delta(t), \tag{5}$$

where α is a *learning rate*, and $\delta(t)$ is a TD *error*:

$$\delta(t) = r(t+1) + \gamma Q(\mathbf{s}(t+1), u^*((t+1))) - Q(\mathbf{s}(t),u(t)). \tag{6}$$

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 neural network or fuzzy system 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 be a function approximator of the Q-function with a weight vector $\Omega = \{\omega\}$. The TD method updates ω of the Q-function Q as follows:

$$\omega(t) \leftarrow \omega(t) + \eta\delta(t)\mathbf{G}(t)$$

$$G(t) \leftarrow \lambda\gamma G(t) + \frac{\partial Q(\mathbf{s}(t),u(t))}{\partial\omega(t)}$$

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

2.2 Policy Search Approach

Figure 3 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.

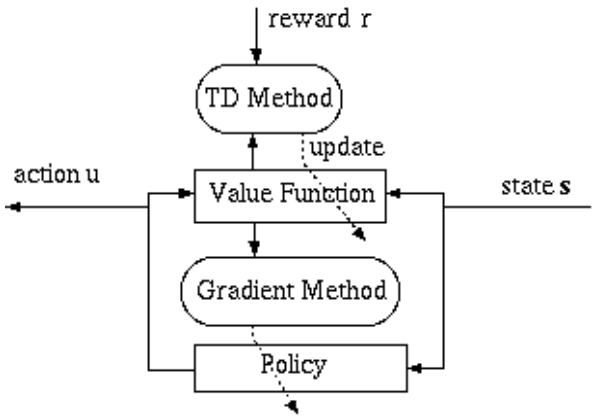

Figure 3 Policy search approach.

Let $u = \pi_w(\mathbf{s})$ 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 [15]:

$$w(t) \leftarrow w(t) + \eta \frac{\partial \rho(\delta(t))}{\partial w(t)}.$$

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

$$w(t) \leftarrow w(t) + \eta \frac{\partial \rho(Q(\mathbf{s}(t), u(t)))}{\partial w(t)}$$

(see [16] for an example of the exact solution of $\partial \rho(Q) / \partial w$).

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:

$$w(t) \leftarrow w(t) + \eta \frac{\partial \rho(Q(\mathbf{s}(t), u(t)))}{\partial u(t)} \frac{\partial u(t)}{\partial w(t)}. \tag{7}$$

Several researchers [17,18] have proposed an update rule similar to (7). They used V as the evaluation function to be maximized to tune the weights of the policy using the following update rule:

$$w(t) \leftarrow w(t) + \eta \frac{\partial V(\mathbf{s}(t))}{\partial u(t)} \frac{\partial u(t)}{\partial w(t)}. \tag{8}$$

Based on the update rule (8), 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 (8). Nevertheless, in [17], 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:

$$\frac{\partial V(\mathbf{s}(t))}{\partial u(t)} \approx \frac{\Delta V(\mathbf{s}(t))}{\Delta u(t)} = \frac{V(\mathbf{s}(t)) - V(\mathbf{s}(t-1))}{u(t) - u(t-1)}.$$

It seems that the update rule (8) is not efficient to improve the weights of the policy. In (8), 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. Hence, there might be situations where we cannot assume that $V(\mathbf{s}(t))$ is dependent on $u(t)$ either directly or indirectly. When such situations occur, we can no longer use even the approximation of $\partial V(\mathbf{s}(t))/\partial u(t)$.

2.3 Disadvantages of Reinforcement Learning

Despite many successful applications of reinforcement learning using the above approaches [1,19,20], reinforcement learning has several difficulties when applied to the control problem, which are as follows. (i) The value function is not readily available. To obtain the best approximation of the value function, many trial-and-error interactions are required. (ii) When the value function approach is used, action selection can be very sensitive to an arbitrarily small change in the estimated value function. (iii) When the goal state is changed, the

learned value function may be no longer useful for the controller to determine the optimal actions.

3 Control with Approximated Policy Search Approach

To solve the aforementioned control problems, we develop CAPS (Control with Approximated Policy Search) as depicted in Figure 4. CAPS tunes parameters of the controller based on an evaluation function. But, it does not use the Q-function as the evaluation function because considerable efforts 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. In general, the architecture of CAPS is similar to the policy search approach described in Figure 3. The difference is that CAPS does not necessarily learn the evaluation function. The evaluation function in CAPS is fixed and directly used to derive an update rule for tuning controller weights. The resulting update rule is of a form that follows one of forms of the update rules used in the policy search approach, i.e., the update rule (7), where the performance measure $\rho(Q)$ is replaced by the proposed evaluation function that will be discussed later within this section.

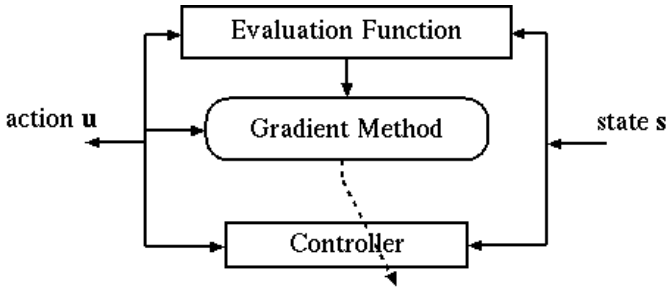

Figure 4 Control with approximated policy search approach.

CAPS is not limited to work with a controller of a certain type, rather a general type. In this paper, we focus on implementing CAPS using two types of controller: a linear state feedback controller [21] and a fuzzy controller [8].

3.1 Linear State Feedback Controller

In this paper, by the linear state feedback controller we mean 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 $\mathbf{w}(t)$ denote a weight column vector of the linear state feedback controller at time t. Similarly, let

$\mathbf{s}(t)$ denote a column state vector of the plant. Using the above definition, the linear state feedback controller can be written as follows

$$u(t) = \mathbf{w}^T(t)\mathbf{s}(t). \tag{9}$$

3.2 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 $s_i (i = 1, 2, 3, \cdots, n)$, we define p_i fuzzy sets or membership functions, each of which is denoted by $A_i^{l_i} (l_i = 1, 2, 3, \cdots, p_i)$. The fuzzy controller is constructed with all possible combinations of the predefined membership functions, i.e., we will have $\Pi_i^n p_i$ rules, each of which is:

$$\text{IF } s_1 \text{ is } A_1^{l_1} \text{ and ... and } s_n \text{ is } A_n^{l_n} \text{ THEN } f(\mathbf{s}) \text{ is } W^{l_1 \cdots l_n} \tag{10}$$

where $W^{l_1 \cdots l_n}$ 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 $\mu_A(s_i)$, and the parameters of the output part are represented by an adjustable column vector $\mathbf{w} = \left\{ w_{l_1 \cdots l_n} \right\}$.

Using a product inference system, singleton fuzzifier, and center of average defuzzifier [8], the fuzzy controller can be written in the following form:

$$u(\mathbf{s}) = \frac{\sum_{l_1=1}^{p_1} \cdots \sum_{l_n=1}^{p_n} w_{l_1 \cdots l_n} \left(\Pi_{i=1}^{n} \mu_{A_i^{l_i}}(s_i) \right)}{\sum_{l_1=1}^{p_1} \cdots \sum_{l_n=1}^{p_n} \left(\Pi_{i=1}^{n} \mu_{A_i^{l_i}}(s_i) \right)}. \tag{11}$$

3.3 Approximate Evaluation Function

Let us use $u = f_{\mathbf{w}}(\mathbf{s})$ to denote a controller where $\mathbf{w}$ is a parameter column vector. In CAPS, all the elements of $\mathbf{w}$ of $f_{\mathbf{w}}$ are set to zero initially 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 [2–11], 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 mean better instantaneous control performances or immediate rewards, and vice versa.

In general, we think that typical evaluative information feedback that will be received after executing an action should depend on at least the action itself, a state at which the action is executed, and the result of executing the action (i.e., the next state). Beside the plant output error cannot be considered as universally *evaluative*, it also obviously does not meet such philosophy. This is exactly the same issue addressed in RL. Several researchers [19,17,15] have proposed different forms of the evaluation function for solving control problems using RL. However, in their research, the evaluation functions are the value functions to be learned online which we think are unsuitable for solving the control problems.

For CAPS, rather than using a learned value function as an evaluation function, we propose an approximated value function of the form:

$$P(\mathbf{s}(t), u(t)) = \frac{1}{2}\left(e^2(t + \Delta t) + \alpha u^2(t) \right), \tag{12}$$

where $e(t + \Delta t)$ denotes a Euclidean distance between a next state $\mathbf{s}(t + \Delta t)$ and a goal state. Without loss of generality, the goal state is assumed at $\mathbf{s} = \mathbf{0}$, and

$e^2(t) = \mathbf{s}^T(t)\mathbf{s}(t)$. t denotes continuous time. Δt is elapsed time between time steps. α is a weighting factor for an action $u(t)$ executed at the state $\mathbf{s}(t)$.

Motivated by the aforementioned philosophy of the typical evaluative information, we suppose that $P(\mathbf{s}(t), u(t))$ is of a role similar to that of the state-action-value function $Q(\mathbf{s}(t), u(t))$. Given $\mathbf{s}(t)$, CAPS produces and executes the action $u(t)$ and its performance is represented as $P(\mathbf{s}(t), u(t))$. But, CAPS does not know the performance $P(\mathbf{s}(t), u(t))$ until it obtains the result of applying $u(t)$ to the plant, i.e., the next state $\mathbf{s}(t + \Delta t)$. CAPS uses $P(\mathbf{s}(t), u(t))$ as the evaluation function at the next time step $t + \Delta t$ 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.

As the plant model is assumed unknown, the next state $\mathbf{s}(t + \Delta t)$ is unknown. From that fact, it does not mean that CAPS is useless. In real application CAPS does not necessarily predict the next state to follow the aforementioned scenario. Rather, at current time t CAPS can position itself as if at $t - \Delta t$, i.e., considering $t - \Delta t$ as if the ”current” time, and the current time t as if the ”next” time step. Hence, the previous state and action are then considered as if the ”current” state and action, respectively, and oppositely, the current state and action are considered as if the ”next” state and action, respectively. Given those ”next” state and action, $P(\mathbf{s}(t - \Delta t), u(t - \Delta t))$ can be computed and considered as the performance measure for the ”current” action $u(t - \Delta t)$. Based on $P(\mathbf{s}(t - \Delta t), u(t - \Delta t))$, CAPS then updates the controller weights used to generate the ”current” action $u(t - \Delta t)$ that leads to the ”next” state $\mathbf{s}(t)$. Thus, in this way CAPS can still be implemented in real application without violating the scenario proposed above.

Of course, availability of the evaluation function P appropriate for solving wide range of the control problems cannot be assured, and therefore, it restricts applicability of CAPS. CAPS is supposed to be applicable when the control problem is simple where 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 could be determined, it could 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 follow. 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 $u(t)$ 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 (12) as follows:

$$P_{\mathbf{w}(t)} = \frac{1}{2} D^2(t), \; D = \sqrt{e^2(t+\Delta t) + \alpha u^2(t)}.$$

CAPS in Figure 4 adjusts $\mathbf{w}(t)$ by

$$\Delta \mathbf{w}(t) = -\eta \frac{\partial P_{\mathbf{w}(t)}}{\partial \mathbf{w}(t)} \Delta t, \tag{13}$$

where η is a positive-definite step size. This is a gradient descent method that requires the partial derivative of $P_{\mathbf{w}(t)}$ with respect to $\mathbf{w}(t)$ to exist. 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, we apply the following chain rule:

$$\frac{\partial P_{\mathbf{w}(t)}}{\partial \mathbf{w}(t)} = D(t) \frac{\partial D(t)}{\partial u(t)} \frac{\partial u(t)}{\partial \mathbf{w}(t)}. \tag{14}$$

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

$$\frac{\partial D(t)}{\partial u(t)} \approx \frac{\Delta D(t)}{\Delta u(t)} = \frac{D(t) - D(t-\Delta t)}{u(t) - u(t-\Delta t)}.$$

Substituting this approximation into (14), and using (13), we obtain the following update rule:

$$\Delta \mathbf{w}(t) = -\eta D(t) \frac{\Delta D(t)}{\Delta u(t)} \frac{\partial u(t)}{\partial \mathbf{w}(t)} \Delta t . \tag{15}$$

In this paper, the method of tuning based on the update rule (15) is referred to as the *Approximated Gradient Descent Method* (AGDM).

3.4 Coping with Approximation Errors

Equation (15) is a crude approximation because any state change between consecutive time steps is missed and ignored. Hence, the update rule (15) requires modifications to cope with the approximation errors.

In Equation (15), when $u(t) - u(t-\Delta t)$ becomes very small as the plant state approaches to the goal state, the approximated gradient might become unacceptable. To avoid such deleterious influences of possible errors in Equation (3.3) when tuning the values of $\mathbf{w}(t)$, we modify the update rule (15) as follows:

$$\Delta \mathbf{w}(t) = -\eta D(t) \text{sign}\left(\frac{\Delta D(t)}{\Delta u(t)} \right) \frac{\partial u(t)}{\partial \mathbf{w}(t)} \Delta t .$$

In this update rule, only slight changes are made to the values of $\mathbf{w}(t+\Delta t)$, no matter how big the value of $\Delta D(t) / \Delta u(t)$ is. And, Figure 5 depicts the modified CAPS where a *failure detector* is introduced to tune $\mathbf{w}(t)$ only when 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:

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

After incorporating the failure detector, the AGDM updates $\mathbf{w}(t)$ by

$$\Delta \mathbf{w}(t) = -\eta f(t) D(t) \text{sign}\left(\frac{\Delta D(t)}{\Delta u(t)} \right) \frac{\partial u(t)}{\partial \mathbf{w}(t)} \Delta t . \tag{16}$$

The role of $f(t)$ can be intuitively explaned 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 $f(t) = 1$.

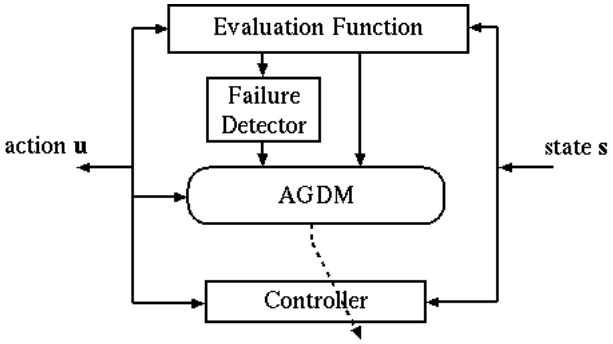

Figure 5 CAPS with failure detector.

The update rule of (16) is *generic* in that it can be used to tune any type of controller provided that the derivative of the action with respect to the controller weights (i.e., $\partial u(t) / \partial \mathbf{w}(t)$) exists. The derivatives of the controllers used in this paper can be easily obtained.

3.5 Convergence Analysis

By its definition in (12), $P_{\mathbf{w}(t)}$ is a positive definite function. Suppose that $P_{\mathbf{w}(t)}$ is a Lyapunov function candidate. The goal is to make the time derivative of $P_{\mathbf{w}(t)}$ negative either definite or semidefinite (or equivalently, to make $\Delta P_{\mathbf{w}(t)} < 0$) until the goal state is achieved at which $\Delta P_{\mathbf{w}(t)} = 0$.

Since $P_{\mathbf{w}(t)}$ is a function of the action $u = u(\mathbf{w}(t), \mathbf{s}(t))$ and the next state $\mathbf{s}(t + \Delta t)$, it can be rewritten as a function of $\mathbf{s}(t + \Delta t)$, $\mathbf{s}(t)$, and $\mathbf{w}(t)$:

$$P_{\mathbf{w}(t)} = P_{\mathbf{w}(t)}(\mathbf{s}(t + t), \mathbf{s}(t), \mathbf{w}(t)).$$

We obtain

$$\Delta P_{\mathbf{w}(t)} = P_1 + P_2 , \tag{18}$$

where

$$P_1 = \left(\frac{\partial P_{\mathbf{w}(t)}}{\partial \mathbf{s}(t+\Delta t)} \right)^T \Delta \mathbf{s}(t+\Delta t) + \left(\frac{\partial P_{\mathbf{w}(t)}}{\partial \mathbf{s}(t)} \right)^T \Delta \mathbf{s}(t) ,$$

$$P_2 = \left(\frac{\partial P_{\mathbf{w}(t)}}{\partial \mathbf{w}(t)} \right)^T \Delta \mathbf{w}(t) . \tag{19}$$

Nothing we can do to make P_1 negative either definite or semidefnite. The second term P_2, however, includes the term $\Delta \mathbf{w}(t)$ that allows us to introduce any adaptation law for the controller weights to make $P_2 < 0$.

Substituting (14) and (15) into P_2, we obtain

$$P_2 = -\eta D^2(t) \frac{\partial D(t)}{\partial u(t)} \operatorname{sign}\left(\frac{\Delta D(t)}{\Delta u(t)} \right) \left(\frac{\partial u(t)}{\partial \mathbf{w}(t)} \right)^T \left(\frac{\partial u(t)}{\partial \mathbf{w}(t)} \right) \Delta t \le 0 .$$

Now, P_2 is guaranteed to be negative definite except the goal state is achieved.

Since $P_2 < 0$, we can hope that by choosing η sufficiently large, we would obtain $|P_2| > |P_1|$, which results in $\Delta P_{\mathbf{w}(t)} < 0$. However, the condition $\Delta P_{\mathbf{w}(t)} < 0$ is not necessarily obtained by applying the update rule of (15) all the time to make $P_2 < 0$, i.e., when P_1 is negative. Keeping the update rule of (15) working to make $P_2 < 0$ when P_1 is negative makes $P_1 + P_2 = \Delta P_{\mathbf{w}(t)}$ more negative, which might be unnecessary, and moreover, make the situation worse. In such a case, introducing the failure detector into the update rule of (15) makes sense in that it cancels P_2 when necessary, i.e., when $P_1 < 0$.

Substituting (14) and the update rule with the failure detector of (16) into P_2 in (19), we obtain

$$P_2 = -\eta f(t) D^2(t) \frac{\partial D(t)}{\partial u(t)} \operatorname{sign}\left(\frac{\Delta D(t)}{\Delta u(t)} \right) \left(\frac{\partial u(t)}{\partial \mathbf{w}(t)} \right)^T \left(\frac{\partial u(t)}{\partial \mathbf{w}(t)} \right) \Delta t \le 0 .$$

Now, P_2 is guaranteed to be negative definite only when $f(t)=1$ and the goal state is not achieved yet, otherwise zero.

Suppose that we choose η sufficiently large that can make $|P_2|>|P_1|$. When P_1 is negative already, $P_{\mathbf{w}(t)}$ must be decreasing, then $f(t)$ can be set to zero to cancel P_2, keeping the current $\mathbf{w}(t)$ considered "good" unchanged. Conversely, when P_1 is positive, $P_{\mathbf{w}(t)}$ would be increasing if P_2 is cancelled, leaving the current $\mathbf{w}(t)$ gets "worse". To prevent such a situation from happening, $f(t)$ should be set to 1 to make $P_2<0$, and we can hope $P_{\mathbf{w}(t)}$ to decrease at the next time step.

4 Experimental Results

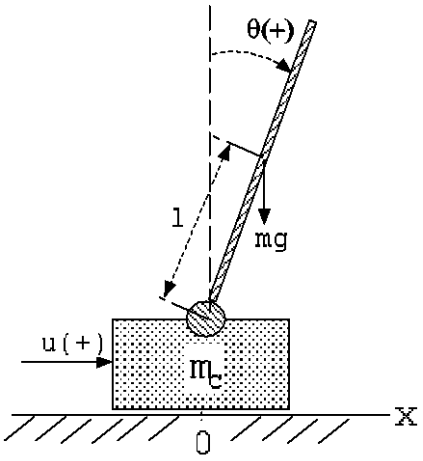

Figure 6 Cart-pole plant.

We implement CAPS using two different types of controller, i.e., the linear state feedback controller (9) and the fuzzy controller (11). We use an extended name to call CAPS when the controller is replaced with a certain type of controller. When the state feedback controller is used, CAPS is referred to as LCAPS (Linear Controller with Approximated Policy Search). When the controller is replaced with the fuzzy controller, CAPS is called by FCAPS (Fuzzy Controller with Approximated Policy Search).

To evaluate performances of both LCAPS and FCAPS, a cart-pole balancing plant as shown in Figure 6 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). In this simulation, only the first two state variables are taken into consideration.

Dynamic equations of the cart-pole plant are as follows:

$$\begin{aligned}
\dot{s}_1(t) &= s_2(t) \\
\dot{s}_2(t) &= \frac{g \sin s_1(t) + \cos s_1(t)\left(\dfrac{-u(t) - ml s_2^2(t) \sin s_1(t)}{m_c + m}\right)}{l\left(\dfrac{4}{3} - \dfrac{m \cos^2 s_1(t)}{m_c + m}\right)} \\
\dot{s}_3(t) &= s_4(t) \\
\dot{s}_4(t) &= \frac{u(t) + ml\left(s_2^2(t) \sin s_1(t) - \dot{s}_2(t) \cos s_1(t)\right)}{m_c + m}
\end{aligned} \tag{20}$$

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 (20), the coefficients of friction of the pole on the cart and the cart on the track are ignored. The cart-pole plant dynamics of (20) 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.

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 Δt = 10 ms.

All the controller parameters in LCAPS and FCAPS are initialized to zero and the controller output is limited within the range of [−20, 20] N. As the linear state feedback controller is given two input, i.e., $s_1 = \theta$ and $s_2 = \dot{\theta}$, its parameter vector is of the size of 2. For the fuzzy controller, 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. 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). By defining five membership functions for each state variables, the fuzzy controller parameter vector will have the size of 25. Given such a 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 parameter vector. And, since we do not introduce any prior knowledge of the plant to initialize the parameter vector (instead, all its elements are simply set to zero), FCAPS will have a heavy burden of adjusting the parameters initially.

There have been many proposed methods to balance the pole [6,8]. They include prior knowledge of the plant and reduce the number of parameters of the 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 CAPS. Instead, we simply set the controller parameters to zero to make the problem of balancing the pole more difficult. 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 two types of problems. (1) Set-point problems where the goal state is fixed. (2) Tracking problems where the goal state is changing. The learning rate is chosen as η = 5000, while α is set to 0,00001.

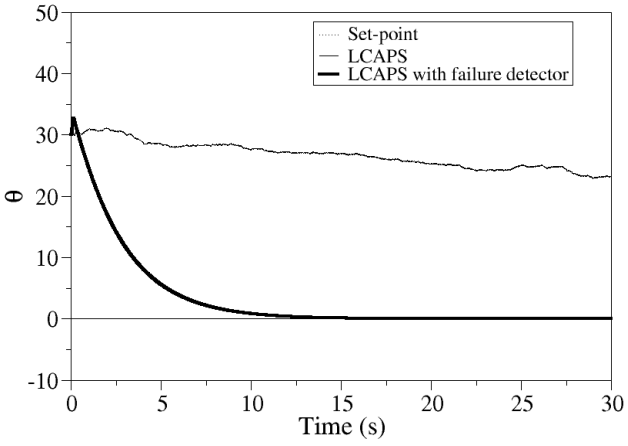

Figure 7 Solving set point problems with LCAPS.

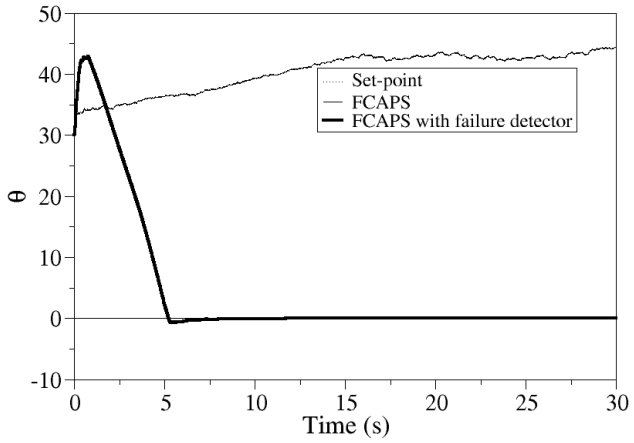

Figure 8 Solving set point problems with FCAPS.

In the set-point problems, the goal is to balance the pole in an upright position. The angular responses due to the application of both LCAPS and FCAPS to the plant are shown in Figure 7 and Figure 8. These graphs show that using the failure detector both LCAPS and FCAPS successfully balanced the pole initialized at 30 deg, but failed when they do not use the failure detector.

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

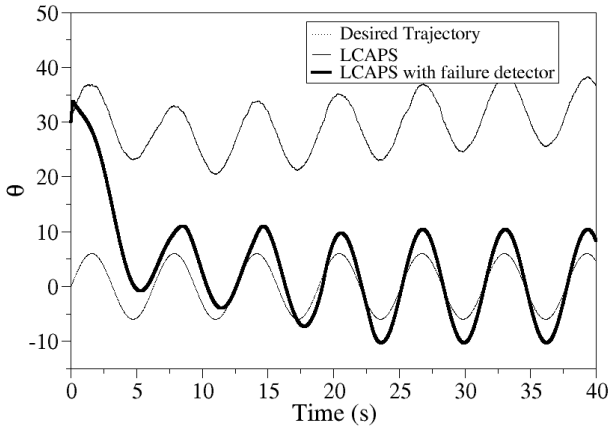

Figure 9 Solving tracking problems with LCAPS.

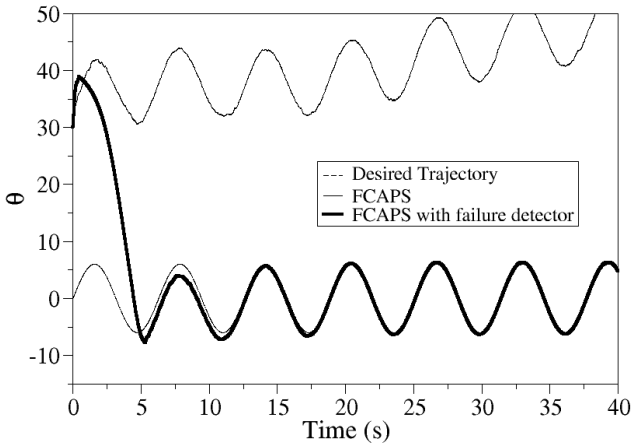

Figure 10 Solving tracking problems with FCAPS.

Figure 9 and Figure 10 shows the angular responses of both LCAPS and FCAPS in the tracking problems. The graphs show that using the failure detector both LCAPS and FCAPS controlled the pole successfully to follow the desired trajectory, but failed when they do not use the failure detector.

In the experiments, the fuzzy controller in FCAPS has 25 parameters while the linear state feedback controller in LCAPS only 2 parameters. The simulation results show that FCAPS is better than LCAPS where FCAPS could follow the desired trajectory more closely than LCAPS. 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.

5 Conclusions

In this paper, an adaptive control scheme that involves manipulating a controller using a gradient method to improve its performance as measured by an approximated evaluation function was implemented. Represented as a function of Euclidean distance from a goal state and an action, the approximated evaluation function could tell the controller the appropriate actions to be executed to solve a control problem whose plant dynamics is not known. Simulation results show that the proposed scheme is effective for controlling the pole-balancing plant despite an unknown plant dynamics. Further, the proposed scheme works better when implemented using a fuzzy controller than using a linear state feedback controller.

References

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

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

[3] Liang, C. & Su, J., *A New Approach to The Design of Fuzzy Sliding Mode Controller*, Fuzzy Sets and Systems, **139**, pp. 111–124, 2003.

[4] Lin, C.-K., *A Reinforcement Learning Adaptive Fuzzy Controller for Robots*, Fuzzy Sets and System, **137**, pp. 339–352, 2003.

[5] Nazaruddin, Y.Y., Naba, A. & Liong, T.H., *Modified Adaptive Fuzzy Control System Using Universal Supervisory Controller*, in Proc. of SCI/ISAS 2000, **IX**, Orlando, USA, July 2000.

[6] Oh, S.K., Pedrycz, W., Rho, S.B. & Ahn, T.C., *Parameters Estimation of Fuzzy Controller and Its Application to Inverted Pendulum*, Engineering Applications of Artificial Intelligence, **17**, pp. 37–60, 2004.

[7] Park, J., Park, G., Kim, S. & Moon, C., *Direct Adaptive Self-Structuring Fuzzy Controller for Nonaffine Nonlinear System*, Fuzzy Sets and Systems, **153**, pp. 429–445, 2005.

[8] Wang, L.-X., *A Course in Fuzzy Systems and Control*. New Jersey: Prentice-Hall International, Inc., 1997.

[9] Liuzzo, S., Marino, R. & Tomei, P., *Adaptive Learning Control of Linear Systems by Output Error Feedback*, Automatica, **43**, pp. 669–676, 2007.

[10] Marino, R., Tomeia, P. & C. Verrelli, *An Adaptive Tracking Control from Current Measurements for Induction Motors with Uncertain Load Torque and Rotor Resistance*, Automatica, **44**, pp. 2593–2599, 2008.

[11] Krstic, M., *On Using Least-Squares Updates without Regressor Filtering in Identification And Adaptive Control of Nonlinear Systems*," Automatica, **45**, pp. 731–735, 2009.

[12] Naba, A. & Miyashita, K., *Tuning Fuzzy Controller Using Approximated Evaluation Function*, in Proc. of the 4th IEEE International Workshop WSTST05, Muroran, Japan, May 2005, pp. 113–122.

[13] ——, *Gradient-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.

[14] ——, *FCAPS: Fuzzy controller with approximated policy search approach*, Journal of Adv. Comput. Intelligence and Intelligent Informatics, **1**(1), pp. 84–92, 2006.

[15] Baird, L. & Moore, A., *Gradient Descent for General Reinforcement Learning*, Advances in Neural Information Processing Systems, **11**, 1999.

[16] Sutton, R.S., McAllester, D., Singh, S. & Mansour, Y., *Policy Gradient Methods for Reinforcement Learning with Function Approximation*,

Advances in Neural Information Processing System, **12**, pp. 1057–1063, 2000.

[17] Berenji, H.R. & Khedkar, P., *Learning and Tuning Fuzzy Logic Controllers Through Reinforcement*, IEEE Trans. Neural Networks, **3**(5), pp. 724–740, 1992.

[18] ——, *Using Fuzzy Logic for Performance Evaluation in Reinforcement Learning*, International Journal of Approximate Reasoning, **18**, pp. 131–144, 1998.

[19] Barto, A.G., Sutton, R.S. & Anderson, C.W., *Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems*, *IEEE Trans. Syst., Man, Cybern.*, **13**(5), pp. 834–846, 1983.

[20] Santamaria, J.C., Sutton, R.R. & Ram, A., *Experiment with Reinforcement Learning in Problems with Continuous State and Action Spaces*, Adaptive Behavior, **6**(2), pp. 163–218, 1998.

[21] Ogata, K., *Modern Control Engineering*, Englewood Cliffs New Jersey: Prentice Hall, 1997.