Kernel-Based Approximate Dynamic Programming for Real-Time Online Learning Control: An Experimental Study

Xin Xu, Senior Member, IEEE, Chuanqiang Lian, Lei Zuo, and Haibo He, Senior Member, IEEE

Abstract—In the past decade, there has been considerable research interest in learning control methods based on reinforcement learning (RL) and approximate dynamic programming (ADP). As an important class of function approximation techniques, kernel methods have been recently applied to improve the generalization ability of RL and ADP methods but most previous works were only based on simulation. This paper focuses on experimental studies of real-time online learning control for nonlinear systems using kernel-based ADP methods. Specifically, the kernel-based dual heuristic programming (KDHP) method is applied and tested on real-time control systems. Two kernel-based online learning control schemes are presented for uncertain nonlinear systems by using simulation data and online sampling data, respectively. Learning control experiments were performed on a single-link inverted pendulum system as well as a double-link inverted pendulum system. From the experimental results, it is shown that both online learning control schemes, either using simulation data or using real sampling data, are effective for approximating near-optimal control policies of nonlinear dynamical systems with model uncertainties. In addition, it is demonstrated that KDHP can achieve better performance than conventional DHP, which uses multilayer perceptron neural networks.

Index Terms—Approximate dynamic programming (ADP), learning control, Markov decision processes (MDPs), online learning, reinforcement learning (RL).

I. INTRODUCTION

REINFORCEMENT learning (RL) is a machine learning framework for solving sequential decision-making problems which can be modeled using the Markov decision process (MDP) formalism. In the past decades, RL has been widely studied not only by the machine learning and neural network community but also in control theory and operations research [1]–[7]. In RL, the learning agent interacts with an initially unknown environment and modifies its action policies to maximize its cumulative payoffs [2]. Thus, RL provides a very promising framework to solve learning control problems that are difficult or even impossible for supervised learning and mathematical programming methods. In the early stages of RL research, most effort was focused on learning control algorithms for MDPs with discrete state and action spaces. But in many real-world applications, a learning controller has to deal with MDPs with large or continuous state and action spaces. In such cases, earlier RL algorithms such as Q-learning and Sarsa-learning may converge very slowly when tabular representations of value functions are used. This issue is known as the “curse of dimensionality”: the exponential growth of the number of parameters to be learned with the size of any compact encoding of system state [3].

To solve the curse of dimensionality, approximate RL methods, also called approximate dynamic programming or adaptive dynamic programming (ADP), have received increasing attention in recent years. Although the concept of ADP was originally proposed in the areas of operations research and control theory, because of their similar research goals, RL and ADP have become a common interdisciplinary research area in recent years. Currently, there are three main categories of research work on approximate RL methods: value function approximation (VFA) [8], policy search [9], and actor–critic methods [10]. Among these three classes of approximate RL methods, the actor–critic algorithms, viewed as a hybrid of VFA and policy search, have been shown to be more effective than VFA or policy search in online learning control tasks with continuous spaces [11]–[13]. In an actor–critic learning control architecture, there is an actor for policy learning and a critic for VFA or policy evaluation. One pioneering work on RL algorithms using the actor–critic architecture was published in [12]. Recently, there has been much research interest on actor–critic methods for RL, where adaptive critic designs (ACD) [11]–[17] were widely studied as an important class of learning control methods for nonlinear dynamical systems. Generally, ACDs can be categorized into the following major groups: heuristic dynamic programming (HDP), dual heuristic programming (DHP), globalized dual heuristic programming (GDHP), and their action-dependent (AD) versions [8]. Among these ACD architectures, DHP is the most popular one and has been proven to be more efficient than HDP [18].

Manuscript received April 25, 2012; accepted February 3, 2013. Manuscript received in final form February 10, 2013. This work was supported in part by the National Natural Science Foundation of China under Grant 61075072, Grant 61005077, 91220301, and Grant 51228701, the New-Century Excellent Talent Plan of the Ministry of Education of China under Grant NCET-10-0901, and the U.S. National Science Foundation under Grant ECCS 1053717. Recommended by Associate Editor A. Alessandri.

X. Xu, C. Lian, and L. Zuo are with the Institute of Unmanned Systems, College of Mechatronics and Automation, National University of Defense Technology, Changsha 410073, China (e-mail: xuxin_mail@263.net; wzdsdq@163.com; zuo1986lei@163.com).

H. He is with the Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island, Kingston, RI 02881 USA (e-mail: he@ele.uri.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCST.2013.2260866
In the past decade, approximate RL algorithms such as ACDs have been studied in various learning control problems [18]–[21], such as aircraft control, automotive engine control, and power system control. In addition to neural networks, kernel methods have been studied to improve the generalization ability of RL algorithms. The main idea of kernel methods is to implicitly construct a nonlinear mapping in a reproducing kernel Hilbert space (RKHS) while all the computation can still be in linear forms by using the kernel trick. One of the early works in kernel RL was published in [22], where kernel-based locally weighted averaging was used to approximate the state value functions of MDPs. Recently, there have been some attempts to apply Gaussian processes (GPs) or support vector machines (SVMs) to RL problems, e.g., GPs in temporal-difference (TD) learning [23], SVMs for RL [24], and GPs in model-based policy iteration learning [25]. Xu et al. [26] proposed the kernel-based least-squares policy iteration (KLSP) algorithm, where a kernel least-squares TD algorithm (KLSTD-Q) was used for efficient policy evaluation. The work on GPs in TD learning (GPTD) [23] has some similarities with KLSTD-Q. But GPTD is to define a probabilistic generative model for the state value function by imposing a Gaussian prior over value functions and assuming a Gaussian noise model. In [24], support vector regression was applied to batch learning of the state value functions of MDPs, but in discrete state spaces, and there were no theoretical results on the policies obtained. The GP-based policy iteration method [25] uses support points, which are usually selected by manual discretization of the state spaces, and the policy evaluation is performed using the state transition model approximated by a GP model.

Until now, kernel methods have been shown to be effective in improving the nonlinear approximation and generalization ability of RL algorithms [27], [28]. However, most of previous results [23]–[26] were based on simulation studies, and there are few works on testing and evaluating kernel-based online learning control methods on real-world dynamical systems. In this paper, two kernel-based online learning control schemes using KDHP are presented to evaluate the performance of online learning control schemes using KDHP. Section V draws conclusions and suggests future work.

II. SHORT OVERVIEW OF MDP AND ACDs

A. Markov Decision Processes

An MDP $M$ is denoted as a quadruple \{X, A, R, P\}, where X is the state space, A is the action space, P is the state transition probability, and R is the reward function. A stochastic stationary policy $\pi$ (or just stationary policy) maps states to distributions over the action space. When referring to such a policy $\pi$, we use $\pi (a|x)$ to denote the probability of selecting action $a$ in state $x$ by $\pi$. A deterministic stationary policy directly maps states to actions, denoted as $a_t = \pi (x_t)$, $t \geq 0$.

When the actions $a_t$ ($t \geq 0$) satisfy (1), policy $\pi$ is followed in the MDP $M$. A stochastic stationary policy $\pi$ is said to be followed in the MDP $M$ if $a_t \sim \pi (a|x_t)$, $t \geq 0$.

The objective of a decision maker is to estimate the optimal policy $\pi^*$ satisfying

$$J_{\pi^*} = \max_{\pi} J_{\pi} = \max_{\pi} E_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t r_t \right]$$

where $0 < \gamma < 1$ is the discount factor; $r_t$ is the reward at time step $t$; $E_{\pi} [\cdot]$ stands for the expectation with respect to the policy $\pi$ and the state transition probabilities, and $J_{\pi}$ is the expected total reward along the state trajectories by following policy $\pi$. In this paper, $J_{\pi}$ is also called the performance value of policy $\pi$.

The state value function $V^\pi (x)$ of a policy $\pi$ is the expected discounted total rewards when starting from $x$ and following the policy $\pi$ thereafter:

$$V^\pi (x) = E_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t r_t | x_0 = x \right].$$

Similarly, the state-action value function $Q^\pi (x, a)$ is defined as the expected discounted total rewards when taking action $a$ in state $x$ and following policy $\pi$ thereafter:

$$Q^\pi (x, a) = E_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t r_t | x_0 = x, a_0 = a \right].$$

For an MDP, a deterministic optimal policy $\pi^*(x)$ maximizes the expected discounted total reward of state $x$

$$\pi^*(x) = \arg \max_a Q^\pi^* (x, a).$$

B. Adaptive Critic Designs

As a popular class of actor–critic RL methods, ACDs make use of both value function approximation and policy gradient learning to search for near-optimal control policies in continuous spaces. Furthermore, the information from plant models can also be incorporated into some ACD methods such as DHP and GDHP.

In all ACDs, there is a critic and an actor in the learning control structure. However, there are some variations in the
critic for different ACD methods. Different ACD structures can be distinguished from three aspects in the critic training process: the inputs of the critic, the outputs of the critic, and the requirements for a plant model in the training process of the critic. For the first aspect, the plant states are usually used as the inputs of the critic network, while in action-dependent structures the critic also receives the action outputs from the actor. For the second aspect, the critic outputs can be either the approximated value function or its derivatives. For example, in the HDP structure, the critic approximates the value function \( V(x(t)) \). To simplify notations, in the following, we will use \( V(x_t) \) instead of \( V^\pi(x(t)) \) and use \( x_t \) instead of \( x(t) \). In the DHP structure, it approximates \( \dot{\lambda}(t) \), which is the gradient of \( V(x_t) \); in GDHP, it approximates both \( V(x_t) \) and its gradient. For the third aspect, a plant model can be either used or unused during the learning process of the critic. Usually, to approximate the value function \( V(x_t) \) alone, the plant model may not be used, e.g., in HDP. It is necessary to use the plant model to approximate the derivative of \( V(x_t) \) such as in DHP.

III. REAL-TIME ONLINE LEARNING CONTROL BASED ON KDHP

In the following, a framework of KDHP is introduced by integrating kernel methods into the critic learning of DHP. Different from previous works, our focus is on the extension of KDHP to real-time online learning control tasks. The aim is to study whether kernel machines can be used effectively in real-time online learning control problems and whether better performance can be obtained when compared to previous DHPs with MLPNNs. In this section, we will present two real-time online learning control schemes using KDHP. Specifically, we will investigate the real-time implementation, online learning, and performance testing of KDHP.

A. Framework of KDHP

A framework of KDHP is shown in Fig. 1. The main components include a critic, a module for kernel-based feature construction, a reward function, an actor/controller, and a model of the plant. The module for kernel-based feature construction is to implement data-driven feature representation and selection in order to get better learning efficiency and generalization performance. The critic is used to approximate the derivatives of value functions, i.e., \( \dot{\lambda}_t \). The actor receives measurement data about the plant’s current state \( x_t \) and outputs the control \( u_t \). The output of the critic is used in the training process of the actor so that policy gradients can be computed. The plant model receives the control \( u_t \), and estimates the next state \( x_{t+1} \). The state data are provided to both the critic and to the reward function.

Mercer kernels are used in the feature construction process. A Mercer kernel is a kernel function that is positive definite, i.e., for any finite set of points \( \{s_1, s_2, \ldots, s_n\} \), the kernel matrix \( K = [k(s_i, s_j)] \) is positive definite. Let \( X \) be the original state space of an MDP. According to the Mercer theorem [28], there exists a Hilbert space \( H \) and a mapping \( \phi \) from \( X \) to \( H \) such that

\[
k(s_i, s_j) = \langle \phi(s_i), \phi(s_j) \rangle
\]

where \( \langle \cdot, \cdot \rangle \) is the inner product in \( H \). Although the dimension of \( H \) may be infinite and the nonlinear mapping \( \phi \) is usually unknown, all the computation in the feature space can still be performed if it is in the form of inner products. Because of the above properties of kernel functions, kernel methods have attracted much research interest to kernelize or design new forms of machine learning algorithms in linear spaces so that nonlinear feature extraction or function approximation can be realized only by selecting appropriate kernel functions.

In KDHP, the value function derivatives are approximated in the critic, and a recursive algorithm of KLSTD [26] using kernel-based feature representation is used. The value function derivative is approximated as

\[
\dot{\lambda}(x) = \frac{\partial V(x)}{\partial x} = \sum_{i=1}^{t} a_i k(x, x_i)
\]

where \( a_i (i = 1, 2, \ldots, t) \) are the weights, and \( x_i (i = 1, 2, \ldots, t) \) are the selected states in the sample data, i.e., trajectories generated from the MDP.

In the actor of DHP, MLPs are used to approximate the policy function

\[
a_t = g(x_t, \tilde{\theta}_t).
\]

B. Online Learning Control Based on Simulated or Real Data

In this section, we study two online learning control schemes that are based on simulated data and real sampled data, respectively. In the first scheme, a simulation model is used to realize data-driven controller design and optimization. Using the simulation model, online learning control algorithms can be performed based on the simulated data. In the simulation process, there are three main steps. The first step is to perform kernel-based feature construction using simulated samples. The second step is online learning control using KDHP. The third step is to test the final control policy’s performance using the simulation model. After these three steps, the final control policies in the actor are fixed and used in real-time control of the real plant. Although there may be some differences between the simulation model and the real plant, because of the near-optimality of the final policies, the learning controller can be expected to have robustness to deal with uncertainties and disturbances.
The second online learning control scheme is to use sampled data from the real plant for both feature construction and policy learning. During the policy learning process, all the iteration and computation of actor–critic learning must be performed in an online real-time style. After convergence of the learning controller, the final policy in the actor is fixed and tested on the plant. Because of the observation noises and disturbances in the physical system, this online learning control scheme may require more learning episodes to obtain a near-optimal policy with good performance. The experimental results in Section IV will show this observation.

Fig. 2 shows a flowchart of the KDHP algorithm. The threshold ε for termination condition can be selected as a small positive value. For the two learning control schemes, the main differences include the way of data generation and the performance test of final policies. Details about the online learning control schemes based on KDHP are discussed in the following.

1) Feature Construction Using Simulated/Real Samples: After the sample collection process, the kernel-based features are constructed in a data-driven way, and a sparsification method based on ALD analysis [26] is employed. Let  \( S_n = \{x_1, x_2, \ldots, x_n\} \) denote a set of data samples and \( \phi \) be a feature mapping on the data, which can be determined by the Mercer kernel function defined in (6). A feature vector set can be obtained as \( \Phi_n = \{\phi(x_1), \phi(x_2), \ldots, \phi(x_n)\} \), \( \phi(x_i) \in \mathbb{R}^{m \times 1}, i = 1, 2, \ldots, n \). To perform ALD analysis on the feature set, a data dictionary is defined as a subset of the feature vector set. The data dictionary \( D \) is initially empty, and ALD analysis is implemented by testing every feature vectors in \( \Phi_n \), one at a time. If a feature vector \( \phi(x) \) cannot be approximated, within a predefined precision, by the linear combination of the feature vectors in the dictionary, it will be added to the dictionary. Otherwise, it will not be added to the dictionary. Thus, after the ALD analysis process, all the feature vectors of the data samples in \( S_n \) can be approximately represented by linear combinations of the feature vectors in the dictionary within a given precision.

In KDHP, the ALD-based sparsification procedure mainly includes the following steps.

Step 1: Input the sample set \( \{x_i\} (i = 1, 2, \ldots, n) \), and initialize the dictionary as an empty set, i.e., \( D_1 = \emptyset, t = 1 \).

Step 2: Compute the optimization solutions

\[
\delta_t = \min_{\epsilon} \left| \sum_{j \in D_t} c_j \phi(x_j) - \phi(x_t) \right|^2.
\]

Because of the kernel trick, after substituting (6) into (9), we can obtain

\[
\delta_t = \min_{\epsilon} \left| c^T K_{t-1} c - 2 c^T k_{t-1}(x_t) + k_{tt} \right|
\]

where \( [K_{t-1}]_{i,j} = k(x_i, x_j), x_i (i = 1, 2, \ldots, d(t-1)) \) are the elements in the dictionary, \( d(t-1) \) is the length of the data dictionary, \( k_{t-1}(x_t) = [k(x_1, x_t), k(x_2, x_t), \ldots, k(x_{d(t-1)}, x_t)]^T, c = [c_1, c_2, \ldots, c_d]^T \), and \( k_{tt} = k(x_t, x_t) \).

The optimal solution for (10) is

\[
c_t = K_{t-1}^{-1} k_{t-1}(x_t)
\]

\[
\delta_t = k_{tt} - k_{t-1}(x_t)c_t.
\]

Step 3: Update the data dictionary by comparing \( \delta_t \) with a predefined threshold \( \mu \). If \( \delta_t < \mu \), the dictionary is unchanged, otherwise, \( x_t \) is added to the dictionary, i.e., \( D_t = D_{t-1} \cup \{x_t\} \).

Step 4: If \( t = n \), then go to Step 2, else end the procedure.

2) Online Learning Control Using KDHP: The critic learning in KDHP is to approximate the derivatives of state value functions that satisfy the following Bellman equation:

\[
\frac{\partial V(x_t)}{\partial x_t} = \frac{\partial r_t}{\partial x_t} + \gamma \frac{\partial E[V(x_{t+1})]}{\partial x_t}
\]

where \( r_t \) is the expected reward, and \( E[\cdot] \) is with respect to the state transition probability by following a stationary policy.

Let

\[
D(x_t) = \frac{\partial x_{t+1}}{\partial x_t} + \frac{\partial x_{t+1}}{\partial a_t} \frac{\partial a_t}{\partial x_t}.
\]

Let \( \bar{a}_t, \beta_t \) denote the weight vector and the learning rate in the critic, respectively. \( P_t \) denotes the variance matrix computed in the weight updating process. To realize online learning in the critic, by using the kernel trick, the following update rules based on the kernel RLS-TD(0) algorithm [26]...
are used in the critic of KDHP

\[
\beta_{t+1} = \frac{P_t \tilde{k}(x_t)}{\mu + (k^T(x_t) - \gamma D(x_t)k^T(x_{t+1}))P_t \tilde{k}(x_t)}
\]

(15)

\[
\tilde{\alpha}_{t+1} = \tilde{\alpha}_t + \beta_{t+1} \left( \frac{\partial R_t}{\partial x_t} - (k^T(x_t) - \gamma D(x_t)k^T(x_{t+1}))\tilde{\alpha}_t \right)
\]

(16)

\[
P_{t+1} = \frac{1}{\mu} \left[ P_t - P_t \tilde{k}(x_t) \times \frac{(k^T(x_t) - \gamma D(x_t)k^T(x_{t+1}))P_t}{\mu + (k^T(x_t) - \gamma D(x_t)k^T(x_{t+1}))P_t \tilde{k}(x_t)} \right]
\]

(17)

where \( \beta_t \) is the step size in the critic, \( \mu (0 < \mu \leq 1) \) is the forgetting factor, \( P_0 = \delta I, \delta \) is a positive number, and \( I \) is the identity matrix.

The actor network is used to generate the control actions based on the observed states of the plant. The output of the actor can be given by (8). The learning objective of the actor is to minimize the performance index of the closed-loop system, which can be computed by the value functions of the MDP

\[
J(x) = V(x) = E_x \left[ \sum_{t=0}^{\infty} \gamma^t r_t | x_0 = x \right].
\]

(18)

In order to optimize the objective function, gradient-based learning techniques are popularly employed in the actor learning process. Particularly, based on the outputs of the critic, the following policy gradient methods can be used to train the actor:

\[
\tilde{\alpha}_{t+1} = \tilde{\alpha}_t - \eta_t \Delta \tilde{\alpha}_t
\]

\[
= \tilde{\alpha}_t - \eta_t \frac{\partial V(x_{t+1})}{\partial \tilde{\alpha}_t} = \tilde{\alpha}_t - \eta_t \frac{\partial V(x_{t+1})}{\partial \alpha_t} \frac{\partial \alpha_t}{\partial \tilde{\alpha}_t}.
\]

(19)

Since \( \lambda(x_{t+1}) \) and \( \partial x_{t+1}/\partial \alpha_t \) can be computed by the critic and the model network, respectively, and \( \partial \alpha_t/\partial \tilde{\alpha}_t \) is given by (8), the above policy gradient learning can be implemented along with the critic learning.

3) Performance Test of the Final Policy: To test the performance of the final policy, the policy function in the actor is fixed after learning and it is used as a real-time controller for the plant.

Let \( \tilde{\theta}_T \) be the weights of the final policy in the actor; the real-time controller is determined as

\[
a = g(x, \tilde{\theta}_T).
\]

(20)

For online learning control using real sampled data, the performance of the final near-optimal policy can be evaluated directly from the experimental results. For the learning control scheme using simulated data, the performance of the final policy needs to be tested under different simulation conditions, e.g., different initial states, disturbances, etc. When the performance of the final policy is ready for different simulation conditions, it is finally tested on the real plant. Otherwise, more learning episodes are needed for the learning controller by using the simulation model.

C. Convergence Analysis

Actor–critic methods are online approximations to policy iteration in which the value function parameters are estimated using temporal difference learning and the policy parameters are updated by stochastic gradient methods. In [10], a class of actor–critic algorithms with linear function approximators was proved to converge when the features for the critic span a subspace prescribed by the choice of parameterization of the actor. Recently, Bhatnagar et al. [29] proved the convergence of natural actor-critic algorithms, where the critic also uses linear function approximators. All of the above theoretical results are based on the theory of two-timescale stochastic approximation, and one of the most important conditions for convergence is that the learning update in the critic is a faster recursion than the actor.

In KDHP, the update rules in the critic use recursive least-square learning and the policy parameters are updated with faster recurrences. Moreover, RLS-TD learning is employed in the critic training process, which has the following update rules:

\[
\tilde{\alpha}_{t+1} = \tilde{\alpha}_t + \beta_{t+1} \left( \frac{\partial R_t}{\partial x_t} - (k^T(x_t) - \gamma D(x_t)k^T(x_{t+1}))\tilde{\alpha}_t \right)
\]

(21)

where \( \beta_{t+1} \) is determined by (15).

In the actor of KDHP, policy gradient learning is performed based on the output from the critic, where the weight update rule has the following form:

\[
\tilde{\theta}_{t+1} = \tilde{\theta}_t - \gamma_{t+1} \Delta \tilde{\theta}_t = \tilde{\theta}_t - \gamma_{t+1} \frac{\partial J(t)}{\partial \tilde{\theta}_t}.
\]

(22)

Similar to the analysis in [17], the update rules (21) and (22) in KDHP can be modeled as a general setting of two-timescale stochastic approximations

\[
X_{t+1} = X_t + \beta_t (f(X_t, Y_t) + N^1_{t+1})
\]

(23)

\[
Y_{t+1} = Y_t + \gamma_t (g(X_t, Y_t) + N^2_{t+1})
\]

(24)

where \( f, g \) are Lipschitz continuous functions and \( \{N^1_{t+1}\}, \{N^2_{t+1}\} \) are martingale difference sequences with respect to the field

\[
E[||N^i_{t+1}||^2 | F_t] \leq D_1 (1 + ||X_t||^2 + ||Y_t||^2), \quad i = 1, 2, t \geq 0
\]

(25)

for some constant \( D_1 < \infty \).

It is assumed that the step sizes \( \beta_t, \gamma_t \) are deterministic and nonincreasing, and satisfy

\[
\sum_t \beta_t = \sum_t \gamma_t = \infty, \sum_t \beta_t^2 < \infty, \sum_t \gamma_t^2 < \infty
\]

(26)

\[
\sum_t \left( \frac{\beta_t}{\beta_t} \right)^d < \infty
\]

(27)

where \( d \) is a positive constant.

In KDHP, the update rules in the critic use recursive least-squares methods and the step sizes are adaptively determined by online computation rules (15). When the step sizes satisfy (26) and (27), the update in the critic is a faster recursion than the update in the actor, and the weights in the critic have
uniformly higher increments compared with the weights in the actor. In [17], it was proved that, when (26) and (27) are satisfied, a class of actor–critic algorithms will converge almost surely to a small neighborhood of a local minimum of the averaged reward $J$. This result provides the theoretical basis for the convergence analysis of actor–critic algorithms. In KDHP, by making use of kernel-based linear features and the RLS-TD learning algorithm in the critic, the update in the critic can be a faster recursion than the actor. Thus, it will be more beneficial to ensure the convergence of the online learning process. In the following section, extensive performance tests and comparisons on the real-time control problem of single-link and double-link inverted pendulums are conducted, and it is shown that KDHP has better performance than DHP in terms of both convergence speed and the quality of the final policies.

IV. EXPERIMENTAL STUDIES

In this section, we will use two real-time online learning control problems to evaluate and compare the performance of KDHP and previous DHP algorithms. One problem is the balancing control of a single-link inverted pendulum system, and the other is a more difficult problem of controlling a double-link inverted pendulum. The inverted pendulum problem has been widely studied as a benchmark control problem with nonlinearity and instability. The controller design for inverted pendulums becomes more difficult when there are model uncertainties and unknown disturbances in the plant dynamics. In recent years, various intelligent control methods have been developed for real-time control of inverted pendulums [30]–[32], but there are still some difficulties, such as how to improve robustness under model uncertainties, the absence of online learning ability, and so on. In the following, KDHP will be applied based on the two online learning control schemes and it is shown that KDHP not only has good real-time control performance but can also obtain good near-optimal policies after learning, which are superior to previous DHP approaches with MLPNNs.

A. Online Learning Control Using Simulated Data—the Case for Single-Link Inverted Pendulum

The dynamics of a single-link inverted pendulum can be described by the following equations:

\[
\begin{align*}
(M + m) \ddot{x} + b \dot{x} - m l \dot{\theta} \cos \theta + m l^2 \sin \theta &= F \\
(I + m l^2) \ddot{\theta} + m g l \sin \theta &= m l \ddot{\theta} \cos \theta
\end{align*}
\]

where $g$ is the acceleration due to gravity, which is 9.8 m/s$^2$. The mass of the cart is $M = 1.096$ kg, the mass of the pole is $m = 0.109$ kg, the half-pole length is $l = 0.25$ m, $b$ is the friction coefficient of the cart on the track, and $l$ is the inertia of the pole on the cart.

In the learning control problem of the single-link inverted pendulum, the state space is spanned by four continuous state variables, i.e., $x$, $\dot{x}$, $\theta$, and $\dot{\theta}$, where $x$ and $\dot{x}$ are the cart position and velocity, and $\theta$ and $\dot{\theta}$ are the pole angle and angle velocity, respectively. The action is the expected acceleration of the cart, which is a continuous variable. The reward function is defined as follows:

\[
r(t) = 0.25(\theta(t) - \theta_d(t))^2 + 0.25(x(t) - x_d(t))^2
\]

where $\theta_d(t)$ is the expected angle of the pole and $x_d(t)$ is the expected position of the cart, which are both zeros in the experiments. The discount factor $\gamma$ is 0.95.

In DHP, three-layer neural networks are used to construct the action module and the critic module, which are called the action neural network (ANN) and the critic neural network (CNN), respectively. The structures of ANN and CNN are determined by the number of nodes in each layer. For ANN, the numbers of nodes in input layer, hidden layer, and output layer are set as 4-5-1, while for CNN, they are set as 4-5-4. Both ANN and CNN have the same transfer functions: from the input layer to the hidden layer, the transfer function is $f(x) = (1 + e^{-x})^{-1}$, and from the hidden layer to the output layer, the transfer function is $L(x) = k x$. Both ANN and CNN have the same learning rate $\alpha = 0.3$. The weights of ANN and CNN are randomly initialized from $-0.5$ to $0.5$. In the simulation, the initial state vector $[x, \dot{x}, \theta, \dot{\theta}]$ is randomly initialized within the following intervals:

$[-2^\circ, 2^\circ], [-2^\circ/s, 2^\circ/s], [-0.1 \text{ m}, 0.1 \text{ m}], [-0.05 \text{ m/s}, 0.05 \text{ m/s}].$

For the RLS-TD learning algorithm, the forgetting rate is set to be 1.0 and $\lambda$ is equal to 0.6. The time step is 0.02 s both in simulation and experiments. A learning trial starts from an initial position and ends after 10,000 time steps or the controller is unsuccessful. If the pole can be stabilized at the expected position for 10,000 time steps, the controller is regarded as successful. One run of the learning control process consists of at most 200 trials. The initial conditions are independently set among different runs. If a successful controller is obtained in one run, this run ends and a new run starts. For the learning control algorithms, the performance is evaluated based on the averaged results of 100 independent runs.

In the KDHP algorithm, different types of kernel functions may influence the performance of the learning controller. In the simulation, four types of kernel functions, namely linear kernel function, polynomial kernel function, Gaussian kernel function, and sigmoid kernel function, are used for performance evaluation. The four types of kernel functions are described by (30)–(33), respectively. From the simulation and experimental results, it is shown that KDHP with linear kernel functions and polynomial kernel functions cannot obtain satisfactory performance. The main reason for this is that linear and polynomial kernel functions have only limited approximation ability, which has been shown in [33]. Gaussian kernel function and sigmoid kernel function exhibit good performance under some parameters

\[
\begin{align*}
k(x_i, x_j) &= x_i \cdot x_j \\
k(x_i, x_j) &= (1 + x_i \cdot x_j)^d \\
k(x_i, x_j) &= \exp(-\|x_i - x_j\|^2 / 2\sigma^2) \\
k(x_i, x_j) &= \tanh(k \cdot x_i \cdot x_j + \theta).
\end{align*}
\]
Fig. 3. Learning control process of DHP for the single-link inverted pendulum system using simulation data.

Fig. 4. Learning control process of KDHP for the single-link inverted pendulum system using simulation data.

Fig. 5. Real-time responses of the pole angle and the cart position using the control policies obtained by DHP, KDHP, and LQR.

TABLE I
AVERAGE LEARNING CONTROL PERFORMANCE EMPLOYING GAUSSIAN KERNEL FUNCTION WITH DIFFERENT PARAMETERS

<table>
<thead>
<tr>
<th>σ</th>
<th>Average Trials</th>
<th>Success Rate (%)</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7</td>
<td>37.1</td>
<td>12</td>
<td>12.745</td>
</tr>
<tr>
<td>2.4</td>
<td>25.7</td>
<td>41</td>
<td>7.397</td>
</tr>
<tr>
<td>2.1</td>
<td>22.4</td>
<td>75</td>
<td>5.214</td>
</tr>
<tr>
<td>1.8</td>
<td>15.9</td>
<td>93</td>
<td>4.548</td>
</tr>
<tr>
<td>1.5</td>
<td>17.1</td>
<td>90</td>
<td>4.801</td>
</tr>
<tr>
<td>1.2</td>
<td>17.8</td>
<td>84</td>
<td>5.069</td>
</tr>
<tr>
<td>0.9</td>
<td>19.5</td>
<td>86</td>
<td>5.281</td>
</tr>
</tbody>
</table>

TABLE II
AVERAGE LEARNING CONTROL PERFORMANCE EMPLOYING SIGMOID KERNEL FUNCTION WITH DIFFERENT PARAMETERS

<table>
<thead>
<tr>
<th>k, θ</th>
<th>Average Trials</th>
<th>Success Rate (%)</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5, 0.5</td>
<td>62.4</td>
<td>22</td>
<td>11.483</td>
</tr>
<tr>
<td>0.5, 1.0</td>
<td>55.7</td>
<td>34</td>
<td>9.827</td>
</tr>
<tr>
<td>1.0, 0.5</td>
<td>36.1</td>
<td>52</td>
<td>7.688</td>
</tr>
<tr>
<td>1.0, 1.0</td>
<td>45.5</td>
<td>45</td>
<td>7.894</td>
</tr>
<tr>
<td>2.0, 0.5</td>
<td>19.3</td>
<td>82</td>
<td>5.104</td>
</tr>
<tr>
<td>2.0, 1.5</td>
<td>23.9</td>
<td>61</td>
<td>6.217</td>
</tr>
<tr>
<td>2.0, 2.0</td>
<td>31.3</td>
<td>47</td>
<td>7.300</td>
</tr>
</tbody>
</table>

Tables I and II show the average learning control performance when Gaussian and sigmoid kernel functions are employed with different parameters in 100 independent runs, respectively. It is shown that the choice of the parameters is critical to the proposed algorithm, and, when employing Gaussian kernel function with σ = 1.8, the best control performance is obtained.

Figs. 3 and 4 show typical learning processes of DHP and KDHP using simulation data, respectively. It is shown that, in the learning process, the control policies are improved, and in the DHP algorithm it needs 24 trials to balance the pole while in the KDHP algorithm only 18 trials are needed and it costs less time to keep the pole stabilized around the equilibrium. Table III shows the performance comparisons of KDHP and DHP in terms of the average learning control performance in 100 runs. It is clear that the performance of KDHP is much better than that of DHP. Thus, KDHP has faster convergence and can obtain better control policies than DHP.
Table V

<table>
<thead>
<tr>
<th></th>
<th>Min. No. of Trials</th>
<th>Max. No. of Trials</th>
<th>Average Trials</th>
<th>Success Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DHP</td>
<td>56</td>
<td>143</td>
<td>95.2</td>
<td>31</td>
</tr>
<tr>
<td>KDHP</td>
<td>29</td>
<td>115</td>
<td>67.5</td>
<td>53</td>
</tr>
</tbody>
</table>

Fig. 6: Pole angle variation using DHP with real sampled data for the inverted pendulum system.

To test the performance of different control policies obtained from simulated data, the real-time control system of a single-link pendulum, developed by Googol Technology Ltd., was used for experimental studies. Due to the unmodeled dynamics and disturbances in real systems, the control policies obtained from simulation data may not be near-optimal because, in the last successful learning trial in simulation, the weights of ANN and CNN are still updated. Thus, the control policies need to be tested with fixed weights in the simulation until a satisfactory control policy is found. In our experiments, after the above performance tests, the control policies obtained from the simulation can always control the real pendulum system with good performance.

Fig. 5 shows that, in the real-time control experiments, compared to DHP, the proposed KDHP algorithm takes a shorter time to balance the pole near the upright position. Moreover, the pole angle can be stabilized to a smaller region around the upright position and the angle variations are minimized. For the DHP algorithm, although the pole can be balanced, it takes a longer time to balance the pole near the upright position and there are larger oscillations in the pole angle.

Fig. 5 also depicts the state variations when a linear quadratic regulator (LQR) is applied in the real pendulum system. The LQR is an optimal controller based on the linear simulation model. It is shown that the performance of KDHP is close to that of LQR. For LQR, due to the observation noises and unknown disturbances in the real system, there are small oscillations in the pole angle. For KDHP, due to the generalization ability of kernel machines, the angle oscillations are smaller.

B. Online Learning Control Using Real Sampled Data—the Case for a Single-Link Inverted Pendulum

The above simulation and experimental results show that when simulated data are used, the near-optimal policies obtained by DHP and KDHP can both be used to control the
real system even if there are some unmodeled dynamics and unknown disturbances in the physical system. It is illustrated that KDHP has better performance than DHP in terms of higher convergence speed and improved quality of final policies. However, the difference between the simulation model and the real system cannot be too large. For many complex systems, it is difficult to get a good simulation model which is close to the real system. Therefore, online learning control using the real sampled data is necessary for such cases. In the following, experimental results will be given for both DHP and KDHP in online learning control with sampled data from the real system of the single-link inverted pendulum.

In the online learning control experiments, the settings of the parameters such as the actor module and the critic module are the same as that in the simulation. In each trial, the initial cart position and pole angle are initialized around the upright position. If a control policy cannot balance the pole or the cart exceeds the boundary, the trial is unsuccessful. The cart position and pole angle are reinitialized and the next trial is started until the controller can successfully balance the pole or the trial number reaches 200, which is regarded as an unsuccessful run. After 200 trials, initialize all parameters afresh and start a new run.

In the online learning control experiments using real sampled data, learning control performances were evaluated based on 10 independent runs. The averaged results are shown in Table IV. Because of the disturbances and noises in the real system, it takes more trials to learn a successful control policy for both DHP and KDHP. Nevertheless, it is shown that KDHP still exhibits much better performance than DHP. In experiments based on simulation data, there is no noise in state observations and control actions. The control process can be modeled as a deterministic MDP. So, the successful rate of KDHP is 93%. However, in experiments on practical sampled data, due to the properties of physical sensors and actuators, there are unknown noises both in state observations and control actions. The control process can be modeled as a stochastic MDP, which is harder for a learning controller to obtain a near-optimal control policy. Therefore, the successful rates of KDHP decline to 70%.
C. Learning Control Experiments on the Double-Link Inverted Pendulum

As shown in Fig. 8, the double-link inverted pendulum system, developed by Googol Technology Ltd., was used in the experiments. The dynamics model of the double-link inverted pendulum can be described by the following equations:

\[
\begin{bmatrix}
\dot{x}
\dot{\theta}_1
\dot{\theta}_2
\dot{\check{x}}
\dot{\check{\theta}}_1
\dot{\check{\theta}}_2
\end{bmatrix}
= \begin{bmatrix}
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 \\
86.69 & -21.62 & 0 & 0 & 0 & 1 \\
-40.31 & 39.45 & 0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
x \\
\theta_1 \\
\theta_2 \\
\check{x} \\
\check{\theta}_1 \\
\check{\theta}_2
\end{bmatrix}
+ \begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
6.64 \\
-0.088
\end{bmatrix}
u.
\]

(34)

In the learning control of the double-link inverted pendulum system, the state space is spanned by six continuous state variables, i.e., x, \dot{x}, \theta_1, \dot{\theta}_1, \theta_2, and \dot{\theta}_2, where x and \dot{x} are the cart position and velocity, \theta_1 and \dot{\theta}_1 are the pole-1 angle and angular velocity, and \theta_2 and \dot{\theta}_2 are the pole-2 angle and angular velocity, respectively. The action is the acceleration of the cart, which is a continuous variable. The reward function at time-step t is defined as follows:

\[
r(t) = ((\theta_1(t) - \theta_{1d}(t))^2 + (\theta_2(t) - \theta_{2d}(t))^2 + (x(t) - x_{d}(t))^2)/4
\]

where \theta_{1d}(t) and \theta_{2d}(t) are the expected angles of pole-1 and pole-2, respectively, and x_{d}(t) is the expected position of the cart. The structures of ANN and CNN are set as 6-5-1 and pole-2, respectively, and xd, \check{x}, \theta_1, \dot{\theta}_1, \theta_2, and \dot{\theta}_2 are the expected positions of the cart and the track and the friction in the joint of two poles are ignored. The second is that, since only a simplified linear model is used in simulation, there are more uncertainties and parameter errors.

From Fig. 10, it is demonstrated that, although there are oscillations of the pole angles for the two algorithms, the oscillation is much smaller in KDHP than that in DHP, which means better control performance.

Fig. 10 also shows the performance of an optimal controller based on LQR. Compared with DHP and KDHP, LQR exhibits better control performance since DHP and KDHP can only obtain near-optimal solutions while LQR can calculate the optimal solution exactly. However, almost all the optimal control methods including LQR need the accurate model information while DHP and KDHP do not, which is a clear advantage.

V. Conclusion

This paper investigated real-time online learning control methods using KDHP. Two kernel-based online learning control schemes were presented for real-time control systems by using simulation data and online sampling data, respectively. Learning control experiments were performed on a single-link as well as a double-link inverted pendulum system. From the experimental results, it was shown that both online learning control schemes, either using simulation data or using real sampling data, are effective for approximating near-optimal control policies of real-time dynamical systems with model uncertainties. In addition, it was demonstrated that the KDHP approach can achieve much better performance than conventional DHP with MLPNNs. Future work may include the design of stable control strategies combined with the online learning method based on KDHP so that the closed-loop system can have stable performance during the online learning control process. Since DHP has been applied in various nonlinear systems such as aircraft control, automotive engine control, and power system control [17]–[19], KDHP can also be applied in many other nonlinear control systems with better learning efficiency and generalization performance.

REFERENCES


