Kernel-Based Least Squares Policy Iteration for Reinforcement Learning

Xin Xu, Dewen Hu, *Senior Member, IEEE*, and Xicheng Lu

***Abstract*—In this paper, we present a kernel-based least squares policy iteration (KLSPI) algorithm for reinforcement learning (RL) in large or continuous state spaces, which can be used to realize adaptive feedback control of uncertain dynamic systems. By using KLSPI, near-optimal control policies can be obtained without much *a priori* knowledge on dynamic models of control plants. In KLSPI, Mercer kernels are used in the policy evaluation of a policy iteration process, where a new kernel-based least squares temporal-difference algorithm called KLSTD-Q is proposed for efficient policy evaluation. To keep the sparsity and improve the generalization ability of KLSTD-Q solutions, a kernel sparsification procedure based on approximate linear dependency (ALD) is performed. Compared to the previous works on approximate RL methods, KLSPI makes two progresses to eliminate the main difficulties of existing results. One is the better convergence and (near) optimality guarantee by using the KLSTD-Q algorithm for policy evaluation with high precision. The other is the automatic feature selection using the ALD-based kernel sparsification. Therefore, the KLSPI algorithm provides a general RL method with generalization performance and convergence guarantee for large-scale Markov decision problems (MDPs). Experimental results on a typical RL task for a stochastic chain problem demonstrate that KLSPI can consistently achieve better learning efficiency and policy quality than the previous least squares policy iteration (LSPI) algorithm. Furthermore, the KLSPI method was also evaluated on two nonlinear feedback control problems, including a ship heading control problem and the swing up control of a double-link underactuated pendulum called acrobot. Simulation results illustrate that the proposed method can optimize controller performance using little *a priori* information of uncertain dynamic systems. It is also demonstrated that KLSPI can be applied to online learning control by incorporating an initial controller to ensure online performance.**

***Index Terms*—Approximate dynamic programming, kernel methods, least squares, Markov decision problems (MDPs), reinforcement learning (RL).**

I. Introduction

In recent years, reinforcement learning (RL), which was originally conceived as descriptive models for phenomena observed in animal behavior, has attracted many research interests not only in machine learning, but also in operations research, control engineering, and other related disciplines. In RL, the learning agent interacts with an initially unknown environment and modifies its action policies to maximize its cumulative payoffs [1], [2]. Thus, RL provides a general methodology to solve complex uncertain sequential decision problems, which are very challenging in many real-world applications. The environment of RL is typically modeled as a Markov decision process or Markov decision problem (MDP), which has been popularly studied in operations research. However, different from traditional dynamic programming methods in operations research, an RL agent is assumed to learn the optimal or near-optimal policies from its experiences without knowing the parameters of the MDP.

To estimate the optimal policy of an MDP, RL algorithms usually estimate the value functions by observing data generated from the state transitions and the rewards. There is a large literature on RL algorithms using various value-function estimation techniques in the last decade. Since no explicit teacher signals can be obtained in RL, the estimation of value functions in RL differs from the regression problems in supervised learning. An earlier breakthrough on value-function estimation is the temporal difference (TD) learning algorithm proposed by Sutton [3]. After that, several algorithms have been studied to implement value-function estimation of finite-state MDPs, and various theoretical results have been proven. For example, the famous Q-learning algorithm proposed by Watkins [4] was proven to be asymptotically convergent to optimal value functions provided every state of the finite-state MDP has been visited an infinite number of times [4], [5]. A general form of TD learning algorithm, i.e., TD(λ), was proven to converge when the cardinality of tunable parameters was the same as that of the state space [6], [7]. In [8], the convergence results of a variant of Q-learning called the state–action–reward–state–action (SARSA) algorithm were also established.

Despite the successful developments in RL theory and algorithms for discrete-state MDPs, an open fundamental problem in RL research is to study RL algorithms and theories based on approximate value functions or policies since most real-world applications have large or continuous state spaces, which make earlier tabular RL algorithms impractical. Aiming at this problem, the research on approximate RL or approximate dynamic programming methods becomes a hot topic in the literature and the existing work on approximate RL can be mainly classified into three categories, i.e., value-function approximation (VFA) [9], direct policy search [10], [11], and the actor–critic approaches [12], [13].

Among these three kinds of approximate RL methods, VFA is the most popular one, and lots of empirical results as well

Manuscript received November 30, 2005; revised September 20, 2006; accepted February 5, 2007. This work was supported by the National Natural Science Foundation of China under Grants 60303012, 60234030, 60225015, and 60675005, and by the National Fundamental Research Program of China under Grant 2005CB321801.

X. Xu is with the Institute of Automation, College of Mechatronics and Automation, National University of Defense Technology, Changsha 410073, P. R. China (e-mail: xuxin_mail@263.net).

D. Hu is with the Department of Automation Control, College of Mechatronics and Automation, National University of Defense Technology, Changsha 410073, P. R. China (e-mail: dwhu@nudt.edu.cn).

X. Lu is with the School of Computer, National University of Defense Technology, Changsha 410073, P. R. China.

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/TNN.2007.899161

as some theoretical analyses have been given. According to the basic properties of function approximators, there are two different kinds of VFA methods in RL. One is RL using linear function approximators, such as linear basis functions [14], [15], etc., and the other includes RL algorithms using nonlinear VFA, e.g., RL based on multilayer perceptrons (MLPs). Since linear VFA algorithms in RL usually have worse approximation and generalization abilities than nonlinear VFA, most successful applications or empirical results of RL are based on nonlinear approximators, which include elevator group control, job–shop scheduling, and the game of TD-Gammon [16]–[18], etc. In these applications, MLPs are commonly employed as the nonlinear approximators for VFA. However, the empirical results of successful RL applications using nonlinear VFA commonly lack a rigorous theoretical analysis and the nonlinear features are usually based on manual selection, e.g., the structures of MLPs. Moreover, the weights are computed by various forms of approximate gradient learning rules, e.g., the direct gradient rules used in [17] and [18]. In [19] and [7], negative results concerning divergence were reported for Q-learning and TD learning based on direct gradient rules. In [19], a class of residual gradient learning algorithms was proposed to keep the stability of VFA. However, the residual gradient algorithms discussed in [19] and later in [20] only guarantee local convergence based on Bellman residual minimization, so it is hard to make the obtained policy optimal or near-optimal due to the manual selection of initial weights as well as the approximation structures.

In contrast to VFA methods in RL, policy search is another class of approximate solution approaches, where the policies of MDPs are represented and approximated directly. Earlier work on policy search for RL is the REINFORCE algorithm [21]. Recently, the gradient in partially observable Markov decision processes (GPOMDP) algorithm [11] was proposed for partially observable MDPs. In these policy search algorithms, stochastic policies are parameterized and the parameters are adjusted by computing the gradient of the average reward, so they are also called policy gradient approaches in RL. Although it has been proven that, under certain conditions, local convergence can be guaranteed for policy gradient algorithms such as GPOMDP, the computational costs are very large for large-scale MDPs and due to the local minima of gradient algorithms, the optimality of ultimate policies is sensitive to initial conditions.

The third class of approximate RL methods for large-scale MDPs is the actor–critic method, which can be viewed as a hybrid of VFA and policy search. In an actor–critic 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 have been increased research interests on actor–critic methods for RL, where adaptive critic designs (ACDs) [38], [39] were widely studied as an important class of approximate dynamic programming methods for nonlinear optimal control problems. Until now, several learning control architectures based on ACDs have been proposed, which include heuristic dynamic programming (HDP), dual heuristic programming (DHP), and globalized dual heuristic programming (GDHP), etc. [38]. Among these ACD architectures, DHP is the most popular one and has been proven to be more efficient than HDP [38]. In addition to the research on ACDs, there are other works about making use of some recent results on policy gradient algorithms in an actor–critic architecture. In [22], the value and policy search (VAPS) algorithm was proposed by combining the residual gradient learning method with a parameterized policy and the update of value functions and policies was based on a measure combining value-function accuracy and policy performance. Although the stability of gradient learning is guaranteed, VAPS will not converge to a local optimal policy except that no weight is put on value-function accuracy. In [23], a policy gradient algorithm was presented for a class of actor–critic methods, where a compatibility condition was required between the VFA and the policy parameterization.

As a popular method studied in operations research, policy iteration (PI) can also be viewed as a class of actor–critic learning algorithms, since in PI, the value functions and the policies are approximated separately, which correspond to the critic and the actor, respectively. In [24], based on the work of least squares temporal difference learning methods in [15], the least squares policy iteration (LSPI) algorithm was proposed. In LSPI, the data efficiency of least squares temporal difference learning, i.e., the LSTD(λ) algorithm, is employed and it offers an RL method with better properties in convergence, stability, and sample complexity than previous RL algorithms. However, the approximation structure in value function and policy representation may have degenerated performance when the features are improperly selected, which was discussed and illustrated in the experiments of [24]. In addition, the application of LSPI to feedback control of complex dynamic systems, from the control engineering perspective, has not been well studied.

As stated previously, despite many advances in RL theory and algorithms, two main obstacles still remain for the wider applications of RL. One is the local convergence of various gradient-based VFA or policy learning methods. Due to the local convergence of gradient algorithms, it is hard to ensure the quality of the ultimate policies to be optimal or near-optimal. The other obstacle is that many RL algorithms and theories with good convergence properties rely heavily on manually selected approximation structure so that the optimal policies are difficult to be well approximated without carefully selected features, for example, the LSPI algorithm, TD(λ) algorithms, etc. Furthermore, the applications of RL methods in feedback control of dynamic systems still need to be investigated in depth.

In this paper, to solve those problems, we propose a kernel-based least squares policy iteration (KLSPI) algorithm, where a novel kernel-based least squares temporal-difference algorithm (KLSTD-Q) is used for efficient policy evaluation. In KLSPI, the main novelty is that Mercer kernels are used in the policy evaluation process and a kernel sparsification procedure based on approximate linear dependency (ALD) is performed to keep the sparsity and improve the generalization ability of the KLSTD-Q solutions. Compared to the previous works on approximate RL methods, KLSPI makes two contributions to eliminate the main difficulties of existing results. One is the convergence and (near) optimality guarantee by using the

KLSTD-Q algorithm for policy evaluation with high precision. The other is the automatic feature selection using the ALD-based kernel sparsification approach. Therefore, the KLSPI algorithm provides a general RL method with generalization performance and convergence guarantee for large-scale MDPs. Experimental results on a stochastic Markov decision problem show that the proposed KLSPI algorithm converges within fewer iterations than LSPI and the optimal or near-optimal solutions can be more easily estimated only with small size of collected samples. Moreover, the application of KLSPI in feedback control of nonlinear uncertain systems was also studied, where two adaptive optimal control problems, including the swing-up control of an underactuated double-link pendulum and a tanker ship heading control problem, were considered. It was verified that the proposed KLSPI method could be used as an online learning control method by combining with a conventional PD controller since the performance can be well guaranteed by using the conventional PD controller as the initial policy. Therefore, for KLSPI in online learning control, it is not necessary for the random data gathering stages in which random/intermediate controls are applied to the system. Simulation results show that KLSPI can optimize the controller performance based on the data generated from the initial policy, where the online data gathering and controller optimization processes are implemented alternatively.

This paper is organized as follows. In Section II, an introduction on MDPs as well as the previous TD learning algorithms is given. The KLSPI algorithm, the KLSTD-Q algorithm, and the convergence analysis are presented in Section III. In Section IV, experimental results on a stochastic chain problem as well as two feedback control problems of uncertain nonlinear systems are provided to illustrate the effectiveness of the proposed algorithm. Some related work is discussed in Section V. Section VI draws conclusions and suggests future work.

II. Markov Decision Processes and TD Learning

A. Markov Decision Processes

A Markov decision process is denoted as a tuple $\{S, A, R, P\}$, where S is the state space, A is the action space, P is the state transition probability, and R is the reward function. The policy of the MDP is defined as a function $\pi : S \to Pr(A)$, where $Pr(A)$ is a probability distribution in the action space. The objective is to estimate the optimal policy π^* satisfying the following:

$$J_{\pi^*} = \max_{\pi} J_{\pi} = \max_{\pi} E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \tag{1}$$

where γ is the discount factor and r_t is the reward at time-step t, $E\pi[\cdot]$ stands for the expectation with respect to the policy π and the state transition probabilities, and $J\pi$ is the expected total reward.

The state value function for a stationary policy π and the optimal state value function for the optimal policy π^* are defined as follows:

$$V^{\pi}(s) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\right] \tag{2}$$

$$V^{*}(s) = E_{\pi^*}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\right]. \tag{3}$$

According to the theory of dynamic programming [25], the optimal value function satisfies the following Bellman equation:

$$V^{*}(s) = \max_{a}\left[R(s,a) + \gamma E\left[V^{*}(s')\right]\right] \tag{4}$$

where $R(s,a)$ is the expected reward received after taking action a in state s.

In RL, to facilitate policy improvement, a variant of the state value functions is usually used, which is the state-action value function defined as

$$Q^{\pi}(s,a) = E^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a\right]. \tag{5}$$

The state-action value function of an MDP satisfies the following Bellman equation:

$$Q^{\pi}(s_t, a_t) = E\Bigg[r(s_t, a_t) + \gamma \sum_{a_{t+1} \in A} p^{\pi}(s_{t+1}, a_{t+1}) Q^{\pi}(s_{t+1}, a_{t+1})\Bigg] \tag{6}$$

where the expectation $E[\cdot]$ is with respect to the state transition probability.

The optimal state-action value function is

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a). \tag{7}$$

When $Q^*(s,a)$ is computed, the optimal policy is easy to be obtained by

$$\pi^{*}(s) = \arg\max_{a} Q^{*}(s,a). \tag{8}$$

B. Policy Iteration and TD Learning Algorithm

As stated in Section I, policy iteration is closely related to the actor–critic learning control architecture of RL, which can be depicted in Fig. 1.

In Fig. 1, policy iteration is implemented in an actor–critic learning control architecture. The critic and the actor perform the procedures of policy evaluation and policy improvement, respectively. Policy evaluation usually makes use of TD learning algorithms to estimate the value functions $Q^{\pi[t]}$ without any model information of the underlying MDPs. Based on the estimation of $Q^{\pi[t]}$, the policy improvement in the actor produces a greedy policy $\pi[t+1]$ over $Q^{\pi[t]}$ as

$$\pi[t+1] = \arg\max_{a} Q^{\pi[t]}(s,a). \tag{9}$$

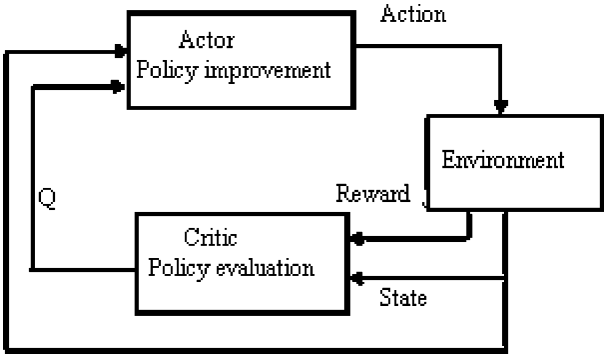

Fig. 1. Policy iteration and actor–critic learning.

Thus, the greedy policy $Q^{\pi[t+1]}$ is a deterministic policy and when the value function $Q^{\pi[t]}$ approximates $\pi[t]$ very well, $\pi[t+1]$ will be at least as good as $\pi[t]$ if not better. This iteration process is repeated until there is no change between the policies $\pi[t]$ and $\pi[t+1]$. After the convergence of policy iteration, the optimal policy may be obtained, usually within very few iterations. However, the convergence of policy iteration greatly relies on the approximation precision of the real value functions of policies. If the value functions are exactly represented, e.g., in cases of tabular state spaces, or the approximation errors are small enough to be neglected, the convergence and the performance of policy iteration will be very satisfactory. Thus, the success of model-free policy iteration will mainly depend on the performance of TD learning algorithms for policy evaluation. In the following, we will briefly introduce some previous work on TD learning, which include tabular TD(λ), linear TD(λ), and LSTD(λ), and in Section III, we will present a new kernel-based TD-learning algorithm, i.e., the KLSTD-Q algorithm, for KLSPI. First, some mathematical notations are presented.

Consider a Markov chain with states in a finite or countable infinite space S. The states of the Markov chain can be indexed as $\{1, 2, \ldots, n\}$, where n is possibly infinite. Let the trajectory generated by the Markov chain be denoted by $\{x_t | t = 0, 1, 2, \ldots; x_t \in X\}$. Although the algorithms and results in this paper are applicable to Markov chains with general state space, the following discussion will be restricted within the cases with a countable state space to simplify the notation.

In TD(λ), there are two basic mechanisms which are the temporal difference and the eligibility trace, respectively. Temporal differences are defined as the differences between two successive estimations and have the following form:

$$\delta_t = r_t + \gamma \widetilde{V}_t(x_{t+1}) - \widetilde{V}_t(x_t) \tag{10}$$

where x_{t+1} is the successive state of x_t, $\widetilde{V}(x)$ denotes the estimate of value function $V(x)$, and r_t is the reward received after the state transition from x_t to x_{t+1}.

As discussed in [1] and [3], the eligibility traces can be viewed as an algebraic trick to improve learning efficiency without recording all the data of a multistep prediction process. This trick is originated from the idea of using a truncated reward sum of Markov chains. In TD learning with eligibility traces, an n-step truncated return is defined as

$$R_t^n = r_t + \gamma r_{t+1} + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n \widetilde{V}_t(x_{t+n}). \tag{11}$$

For an absorbing Markov chain whose length is T, the weighted average of truncated returns is

$$R_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^n + \lambda^{T-t-1} R_T \tag{12}$$

where $0 \le \lambda \le 1$ is a decaying factor and

$$R_T = r_t + \gamma r_{t+1} + \ldots + \gamma^T r_T. \tag{13}$$

In (13), R_T is the Monte Carlo return at the terminal state. In each step of TD(λ), the update rule of value-function estimation is determined by the weighted average of truncated returns defined previously, i.e.,

$$\Delta \widetilde{V}_t(s_i) = \alpha_t \left(R_t^\lambda - \widetilde{V}_t(x_i) \right) \tag{14}$$

where a_t is a learning factor.

The updated equation (14) can be used only after the whole trajectory of the Markov chain is observed. To realize incremental or online learning, eligibility traces are defined for each state as follows:

$$z_t(x_i) = \begin{cases} \gamma\lambda z_{t-1}(x_i) + 1, & \text{if } x_i = x_t \\ \gamma\lambda z_{t-1}(x_i), & \text{if } x_i \ne x_t \end{cases}. \tag{15}$$

The online TD(λ) update rule with eligibility traces is

$$\widetilde{V}_{t+1}(x_i) = \widetilde{V}_t(x_i) + \alpha_t \delta_t z_t(x_i) \tag{16}$$

where δ_t is the temporal difference at time step t, which is defined in (10) and $z_0(x) = 0$ for all x.

The convergence of tabular TD learning algorithms has been well studied in [26]. Nevertheless, in most applications, function approximators have to be used for generalization in large and continuous state spaces.

C. Linear and Least Squares TD(λ) Algorithm

For Markov chains with large or continuous state spaces, function approximators are commonly used in TD learning algorithms, where the value functions are represented as

$$\widetilde{V}(x) = \phi^T(x) W = \sum_{j=1}^{n} \phi_j(x) w_j \tag{17}$$

where $\phi(x) = [\phi_1(x), \phi_2(x), \ldots, \phi_n(x)]^T$ is a vector of basis functions, x is an observation state in the trajectory $\{x_t | t = 0, 1, 2, \ldots; x_t \in X\}$ generated by a Markov chain, and $W = [w_1, w_2, \ldots, w_n]^T$ is the weight vector.

The update rule for TD(λ) algorithms with function approximation becomes

$$W_{t+1} = W_t + \alpha_t \left(r_t + \gamma \phi^T(x_{t+1}) W_t - \phi^T(x_t) W_t \right) \vec{z}_t \tag{18}$$

where the eligibility trace vector $z_t(x)$ is defined as

$$\vec{z}_t = \gamma\lambda \vec{z}_{t-1} + \phi(x_t). \tag{19}$$

In [7], the previous TD(λ) algorithm was proven to converge with probability 1 under certain assumptions and the limit of convergence W^* was also derived, which satisfies the following:

$$E_0\left[A(X_t)\right]W^* - E_0\left[b(X_t)\right] = 0 \tag{20}$$

where $X_t = (x_t, x_{t+1}, z_t)$ $(t = 1, 2, \ldots)$ form a Markov process, $E_0[\cdot]$ stands for the expectation with respect to the unique invariant distribution of $\{X_t\}$, and $A(X_t)$ and $b(X_t)$ are defined as

$$A(X_t) = \vec{z}_t\left(\phi^T(x_t) - \gamma\phi^T(x_{t+1})\right) \tag{21}$$

$$b(X_t) = \vec{z}_t r_t. \tag{22}$$

The least squares TD(λ) algorithm, i.e., LSTD(λ), proposed in [15], computes the weight vector W by solving (20) directly, i.e.,

$$W_{\mathrm{LSTD}(\lambda)} = A_T^{-1}b_T = \left(\sum_{t=1}^{T} A(X_t)\right)^{-1}\left(\sum_{t=1}^{T} b(X_t)\right). \tag{23}$$

As studied in [15] and [14], LSTD(λ) and recursive LSTD(λ) algorithms have better data efficiency than conventional TD(λ) algorithms. However, for Markov chains with nonlinear value functions, the performance of LSTD(λ) algorithms may degrade significantly when the basis functions which are not properly selected cannot approximate nonlinear value functions with good precision and generalization ability.

III. KLSPI

As discussed previously, the convergence of policy iteration greatly depends on the approximation ability of TD learning algorithms. In this section, by introducing a new kernel-based least squares TD learning method for state-action value functions, which is called KLSTD-Q, the KLSPI algorithm is proposed. In the policy evaluation of KLSPI, least squares TD learning is implemented in a kernel-induced linear feature space so that linear LSTD(λ) algorithms can be applied while having a nonlinear value-function representation in the original space. The KLSTD-Q algorithm is a new variant of the kernel-based LSTD algorithm, which was first presented in [27]. Furthermore, to reduce the computational costs of kernel methods, a kernel sparsification method based on approximate linear dependence (ALD) analysis is integrated into KLSTD-Q. In the following, we will present the framework of KLSPI first, and then, the KLSTD-Q algorithm with the kernel sparsification procedure, as well as the convergence analysis of KLSPI will be given.

A. Framework of KLSPI

In KLSPI, the kernel function and its induced feature space play important roles both in policy evaluation and in policy improvement. Let S denote the original state space. A kernel function is a mapping from $S \times S$ to R, which is usually assumed to be continuous. 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. According to the Mercer theorem [28], there exists a Hilbert space H and a mapping φ from S to H such that

$$k(s_i, s_j) = \langle\varphi(s_i), \varphi(s_j)\rangle \tag{24}$$

where $\langle\cdot,\cdot\rangle$ is the inner product in H. Although the dimension of H may be infinite and the nonlinear mapping φ is usually unknown, all the computation in the feature space can still be performed if it is in the form of inner products. Due to the aforementioned properties of kernel functions, kernel methods have attracted many research interests to kernelize or design new forms of previous machine learning algorithms in linear spaces so that nonlinear feature extraction or function approximation can be realized only by selecting appropriate kernel functions, e.g., support vector machines (SVMs), kernel principal component analysis (KPCA), kernel independent component analysis (ICA), etc. [29], [30].

By introducing Mercer kernels in the policy evaluation and policy improvement process of policy iteration, KLSPI can be viewed as a kernelized version of the previous LSPI algorithm. However, there are two important problems to be considered specifically in KLSPI. The first one is how to integrate kernel methods in TD learning algorithms for approximating the state-action value functions of a given policy. A slightly different problem has been recently studied by Xu *et al.* [27], where the kernel-based TD learning method—KLSTD—was proposed. In the policy evaluation process of KLSPI, we will employ a new variant of KLSTD, the KLSTD-Q algorithm, which will be described in detail in Section III-B. By using KLSTD-Q, the state-action value function is represented by

$$\widetilde{Q}(s) = \sum_{i=1}^{t} \alpha_i k(s, s_i) \tag{25}$$

where s and s_i are the combined features of state-action pairs (x, a) and (x_i, a_i), respectively, α_i $(i = 1, 2, \ldots, t)$ are the coefficients, and (x_i, a_i) $(i = 1, 2, \ldots, t)$ are selected state-action pairs in the sample data, i.e., trajectories generated from a Markov decision process.

The second problem for KLSPI is how to guarantee the sparsity of solutions and decrease the computational costs of kernel methods. As is well known, a key problem for the applications of kernel methods is that the number of adjustable parameters or coefficients in the solutions is originally equal to the number of sample data points so that when the size of training data increases, there will be severe computational problems as well as the decrease of generalization performance. Aiming at this problem, various regularization methods for sparsifying kernel machines were studied in the literature. For example, the sparsity of support vector regression is obtained by making use of the structural risk minimization (SRM) principle and the e-insensitive cost function [29]. In KLSPI, we will employ an ALD analysis method which has been applied in some kernel-based supervised learning algorithms including sparse online SVMs [31] and the kernel recursive LS (RLS) algorithm in [32]. Although other supervised kernel regression methods such as SVMs can realize the sparsification process in a more direct way, the extension of these methods to RL may need more complex theoretical analysis to ensure convergence and

optimality since the SRM principle was originally dedicated to supervised learning problems. On the contrary, our method integrates kernel methods with LSTD(λ) so that the existing theoretical works on LSTD(λ) can be well employed to analyze the convergence and optimality of KLSPI.

Let $S_n = \{s_1, s_2, \ldots, s_n\}$ denote a set of observation data samples and ϕ be a feature mapping on the data. A feature vector set can be obtained as $\Phi_n = \{\phi(s_1), \phi(s_2), \ldots, \phi(s_n)\}$. To perform ALD analysis on the feature vector set, a data dictionary is defined as a subset of the feature vector set, whose elements are approximately linearly independent. The data dictionary is initially empty and the ALD analysis is implemented by testing every feature vectors in Φ_n, one at a time. If the feature vector $\phi(s)$ of a data sample 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.

Suppose we have tested $t-1 (1 < t \leq n)$ feature vectors of the samples in the original data set S_n and $D_{t-1} = \{\phi(s_j)\} (j = 1, 2, \ldots d(t-1))$ is the obtained data dictionary. The approximately linearly dependent condition of a new feature vector $\phi(s_t)$ is tested as follows [32]:

$$\delta_t = \min_c \left\| \sum_j c_j \phi(s_j) - \phi(s_t) \right\|^2 \leq \mu \tag{26}$$

where $c = [c_j]$ and μ is a threshold parameter to determine the approximation accuracy and the sparsity level. When μ is appropriately selected, the sparsity of kernel-based solutions can be guaranteed without sacrificing much in approximation accuracy.

In KLSPI, the sparsification procedure using ALD analysis is integrated into the KLSTD-Q algorithm, where the feature vectors $\phi(s_t)$ are implicitly constructed by the kernel function. Different from the previous discussion of ALD analysis, where the feature vectors are tested and stored in the dictionary, in KLSPI, we will use the data samples directly in the dictionary since the feature vectors are only implicitly determined by the kernel functions. Moreover, the computation of (26) will also be replaced by corresponding kernel functions.

The sparsification procedure in KLSPI mainly includes two steps. The first step is to compute the following optimization solutions:

$$\delta_t = \min_c \left\| \sum_j c_j \phi(s_j) - \phi(s_t) \right\|^2 \tag{27}$$

which is equivalent to

$$\delta_t = \min_c \left\{ \sum_{i,j} c_i c_j \langle \phi(s_i), \phi(s_j) \rangle - 2 \sum_i c_i \langle \phi(s_i), \phi(s_t) \rangle + \langle \phi(s_t), \phi(s_t) \rangle \right\}. \tag{28}$$

Due to the kernel trick, after substituting (24) into (28), we can obtain

$$\delta_t = \min_c \left\{ c^T K_{t-1} c - 2 c^T k_{t-1}(s_t) + k_{tt} \right\} \tag{29}$$

where $[K_{t-1}]_{i,j} = k(s_i, s_j)$, $s_i (i = 1, 2, \ldots, d(t-1))$ are the elements in the dictionary, $d(t-1)$ is the length of the data dictionary, $k_{t-1}(s_t) = [k(s_1, s_t), k(s_2, s_t), \ldots, k(s_{d(t-1)}, s_t)]^T$, $c = [c_1, c_2, \ldots, c_d]^T$ and $k_{tt} = k(s_t, s_t)$.

The optimal solution for (29) is

$$c_t = K_{t-1}^{-1} k_{t-1}(s_t) \tag{30}$$

$$\delta_t = k_{tt} - k_{t-1}^T(s_t) c_t. \tag{31}$$

The second step of the ALD-based sparsification is to update the data dictionary by comparing δ_t with a predefined threshold μ. If $\delta_t < \mu$, the dictionary is unchanged; otherwise, s_t is added to the dictionary, i.e., $D_t = D_{t-1} \cup s_t$.

By introducing the ALD-based sparsification method, the computational complexity as well as the memory cost of kernel methods can be greatly reduced and more benefits of generalization ability are also obtained.

For a given set of data samples generated from an MDP, which has the form of $\{(x_i, a_i, r_i, x_i', a_i')\} (i = 1, 2, \ldots, n)$, a combined data vector $s(x, a)$ or s can be defined for each state-action pair. After the sparsification procedure, a data dictionary D_n with reduced number of data vectors will be obtained and the approximated state-action value function can be represented as follows:

$$\widetilde{Q}(x, a) = \sum_{j=1}^{d(n)} \alpha_j k(s, s_j) \tag{32}$$

where $d(n)$, usually much smaller than the original sample size n, is the length of the dictionary and $s_j = s(x_j, a_j) (j = 1, 2, \ldots, d(n))$ are the elements of the data dictionary.

Based on the previous discussion, the framework of the KLSPI algorithm, where the ALD-based kernel sparsification procedure is integrated into KLSTD-Q, can be described in the following.

Algorithm 1: KLSPI Algorithm

1. Given:
   - A kernel function $k(\cdot, \cdot)$ and its parameters.
   - A termination criterion for the algorithm and the other parameters such as the sparsification threshold μ.
   - An initial policy $\pi[0]$, which can be randomly generated or obtained from an *a priori* policy.
   - Sources of data samples $\{(x_i, a_i, r_i, x_{i+1}, a_{i+1})\}$ generated by an MDP under the initial policy $\pi[0]$.

2. Initialize:
Let iteration number $t = 0$.

3. Loop:
3.1. For the current set of data samples, using the KLSTD-Q algorithm with ALD-based kernel sparsification, which will be presented in Section III-B, to approximate the state-action value functions.

3.2. Use (9) to perform policy improvement so that a new greedy policy $\pi[t+1]$ with respect to the approximated value functions is obtained.
3.3. Generate new data samples using the policy $\pi[t+1]$.
3.4. $t = t+1$, return to (3.1).
until the termination criterion is satisfied.

The termination criterion for KLSPI can be selected as the maximum iteration number or the distance between two successive policies $\pi[t]$ and $\pi[t+1]$. When generating new data samples, some of the old data samples may be reused, i.e., a sample $(x_i, a_i, r_i, x_{i+1}, a_{i+1})$ in the last iteration may be replaced by

$$\left(x_i, a_i, r_i, x_{i+1}, a'_{i+1}\right) \tag{33}$$

where a'_{i+1} is the action selected by the new policy $\pi[t+1]$, which is greedy with respect to the estimated state-action value function, i.e.,

$$a'_{i+1} = \arg\max_a \widetilde{Q}_{\pi[t]}\left(s(x_{i+1}, a)\right). \tag{34}$$

B. KLSTD-Q Algorithm With Kernel Sparsification

As discussed in Section III-A, KLSPI makes use of a new version of kernel-based TD learning algorithms, called KLSTD-Q, for model-free nonlinear approximation of state-action value functions. The details of KLSTD-Q will be given in this section.

Based on the idea of kernel methods, KLSTD-Q represents the state-action value functions of an MDP as follows:

$$\widetilde{Q}(x, a) = \sum_{i=1}^{t} \alpha_i k(s, s_i) \tag{35}$$

where s is a combined feature vector of state-action pair (x, a).

From (20) to (22), the regression equation for linear LSTD(0) $(\lambda = 0)$ learning algorithms is

$$E_0\left[\phi(s_i)\left(Q(s_i) - \gamma Q(s_{i+1})\right)\right] = E_0\left[\phi(s_i) r(s_i)\right] \tag{36}$$

where

$$Q(s) = \phi^T(s) W \tag{37}$$

and (36) can be rewritten as

$$E_0\left[\phi(s_i)\left(\phi^T(s_i) - \gamma \phi^T(s_{i+1})\right)\right] W = E_0[\phi(s_i) r(s_i)] \tag{38}$$

$$s_i = s(x_i, a_i). \tag{39}$$

Similar to the discussion in [32], the weight vector W in (37) can be represented by the weighted sum of the state feature vectors

$$W = \sum_{i=1}^{T} \phi(s_i)\alpha_i \tag{40}$$

where $s_i (i = 1, 2, \ldots, T)$ are the observed state-action features and α_i are the corresponding coefficients.

Since the unbiased estimations of expectation $E_0[\cdot]$ can be obtained as

$$E_0[y] = \frac{1}{T}\sum_{i=1}^{T} y_i \tag{41}$$

where $y_i (i = 1, 2, \ldots, T)$ are observed data of random variable y, the least squares regression (38) can be expressed as follows:

$$\sum_{i=1}^{T}\left[\phi(s_i)(\phi^T(s_i) - \gamma\phi^T(s_{i+1})\right]\sum_{j=1}^{T}\phi(s_j)\alpha_j = \sum_{i=1}^{T}\phi(s_i) r_i. \tag{42}$$

The single-step observation function of (42) is

$$\phi(s_i)\left(\phi^T(s_i) - \gamma\phi^T(s_{i+1})\right)\sum_{j=1}^{T}\phi(s_j)\alpha_j = \phi(s_i) r_i + \varepsilon_i \tag{43}$$

where ε_i is the observation noise.

Let

$$\Phi_T = \left(\phi^T(s_1), \phi^T(s_2), \ldots, \phi^T(s_T)\right)^T \tag{44}$$

$$\vec{k}(s_i) = \left(k(s_1, s_i), k(s_2, s_i), \ldots, k(s_T, s_i)\right)^T. \tag{45}$$

By multiplying Φ_T to both sides of the observation (43), due to the kernel trick, we can get

$$\vec{k}(s_i)\left[\vec{k}^T(s_i)\alpha - \gamma\vec{k}^T(s_{i+1})\alpha\right] = \vec{k}(s_i) r_i + \vec{v}_i \tag{46}$$

where $\vec{v}_i \in R^{T\times 1}$ is a transformed noise vector and

$$\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_T]^T. \tag{47}$$

Let

$$A_T = \sum_{i=1}^{T}\vec{k}(s_i)\left[\vec{k}^T(s_i) - \gamma\vec{k}^T(s_{i+1})\right] \tag{48}$$

$$b_T = \sum_{i=1}^{T}\vec{k}(s_i) r_i. \tag{49}$$

Then, the kernel-based least squares solution to the TD learning problem is as follows:

$$\alpha = A_T^{-1} b_T. \tag{50}$$

Based on (48) and (49), the incremental update equations for the KLSTD-Q algorithm are

$$A_t = A_{t-1} + \vec{k}(s_t)\left[\vec{k}^T(s_t) - \gamma\vec{k}^T(s_{t+1})\right] \tag{51}$$

$$b_t = b_{t-1} + \vec{k}(s_t) r_t. \tag{52}$$

As is well known, TD learning is aimed at solving policy evaluation of MDPs with stationary policies, which can be modeled as ergodic or absorbing Markov chains. For ergodic Markov chains, the aforementioned update formula of KLSTD-Q can be applied without modification. However, for Markov chains

with absorbing states, the state features of absorbing states are all zeros so that the update (51) will be

$$A_t = A_{t-1} + \vec{k}(s_t)\vec{k}^T(s_t). \tag{53}$$

To make the aforementioned KLSTD-Q algorithm practical, one key problem is to decrease the computational and memory costs of kernel vectors $k(s)$, whose dimension is originally equal to the number of data points. This problem is common to almost all the kernel-based learning algorithms and, in KLSTD-Q, we use the ALD method for sparsification of kernel vectors $k(s)$. As stated in Section III-A, in the ALD-based kernel sparsification, a dictionary is defined for the sparsification procedure, which is initially empty and new elements are added when the approximation error δ_t computed by (31) is greater than a threshold μ. After the data samples are collected for a given policy, the ALD-based sparsification method is used to select a part of the data samples both for sparsification and for generalization. When the sparsification procedure is completed, the feature vectors produced by the elements in data dictionary are employed to replace the feature vectors of the whole data set and the following vectors with reduced dimensions can be obtained:

$$\Phi^d = \left(\phi^T(s_1), \phi^T(s_2), \ldots, \phi^T(s_d)\right)^T \tag{54}$$

$$k^d(s_i) = \left(k(s_1, s_i), k(s_2, s_i), \ldots, k(s_d, s_i)\right)^T \tag{55}$$

where $D_T = \{s_i = s(x_i, a_i)\}(i = 1, 2, \ldots, d)$ is the data dictionary, here $d \ll T$.

Then, the incremental update equation for KLSTD-Q is modified as

$$A_t^d = A_{t-1}^d + k^d(s_t)\left[k^d(s_t) - \gamma k^d(s_{t+1})\right]^T \tag{56}$$

$$b_t^d = b_{t-1}^d + k^d(s_t) r_t. \tag{57}$$

The solution to the kernel-based LSTD problem is

$$\alpha = \left(A_t^d\right)^{-1} b_t^d \tag{58}$$

$$\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_d]^T. \tag{59}$$

Compared to the LSTD-Q algorithm studied in [24], the main difference in KLSTD-Q is that the manually selected linear feature vector $\phi(s)$ is replaced by a kernel-based feature vector $k(s)$, which can be automatically generated by the kernel function and the ALD-based kernel sparsification procedure. Thus, the kernel sparsification procedure in KLSTD-Q can be viewed as an automated feature selection and optimization method in a kernel-induced linear feature space. This feature selection and optimization method provides an efficient solution to the main difficulty of the LSPI algorithm, which was pointed out as a fundamental problem of LSPI to be solved [24]. Furthermore, the approximation and generalization ability of kernel methods will greatly contribute to the convergence and performance of policy iteration algorithms, which will be analyzed in theory and illustrated in our experiments.

The KLSTD-Q algorithm with the ALD-based kernel sparsification procedure for ergodic Markov chains can be presented as follows.

Algorithm 2: KLSTD-Q Algorithm

1. Given:
   - A sample data set $\{(x_i, a_i, r_i, x_i', a_i')\}(i = 1, 2, \ldots, n)$.
   - A kernel function $k(\cdot, \cdot)$.
   - Threshold parameter μ for kernel sparsification.

2. Perform sparsification:
2.1. Let $i = 0$, the initial dictionary $\mathrm{Dic} = \{\ \}$.
2.2. Loop for the whole data set or selected part of the data set.
   1) $i = i + 1$.
   2) For state-action pair (x_i, a_i), use (30) and (31) to compute the ALD coefficients c_t and δ_t.
   3) If $\delta_t < \mu$, the dictionary Dic is unchanged, Else $\mathrm{Dic} = \mathrm{Dic} \cup s(x_i, a_i)$, where $s(x_i, a_i)$ is the combined feature of state-action pair (x_i, a_i).
   4) For state-action pair (x_i', a_i'), perform the same computation and update to the dictionary.

3. Compute A_t^d and b_t^d incrementally:
3.1. Let $t = 0$, $A_t^d = 0$, and $b_t^d = 0$.
3.2. Loop for the whole data set:
   1) $t = t + 1$.
   2) For the current sample $(x_t, a_t, r_t, x_{t+1}, a_{i+1})$, use (55) to compute the kernel-based feature vectors of state-action pairs (x_t, a_t) and (x_{t+1}, a_{t+1}).
   3) Use (56) and (57) to compute A_t^d and b_t^d.

4. Compute the KLSTD solution using (58), output α as well as the data dictionary.

For absorbing Markov chains, the KLSTD-Q algorithm is only different in dealing with the absorbing states, i.e., if state-action pair (x_t, a_t) is an absorbing state, the update equation for A_t^d becomes

$$A_t^d = A_{t-1}^d + k^d(s_t)\left[k^d(s_t) - 0\right]^T. \tag{60}$$

C. Convergence Analysis

The convergence of KLSPI is determined by three factors. One is the convergence of the ALD-based kernel sparsification process. Second is the approximation error of KLSTD-Q and the third one is the convergence of approximate policy iteration based on approximate policy evaluation and greedy policy improvement. In this section, the convergence theorem of KLSPI will be presented by analyzing the three factors in KLSPI. First, the Lemmas 3.1–3.3 are introduced.

Lemma 3.1 [32]: For the ALD-based kernel sparsification procedure, assume the following: 1) $k(.,.)$ is a continuous Mercer kernel and 2) S is a compact subset of a Banach space. Then, for any training sequence $\{s_i\} \in S(i = 1, 2, \ldots, \infty)$ and for any $\mu > 0$, the number of dictionary vectors is finite.

In Lemma 3.1, it is shown that if the original state space S is compact, the ultimate dictionary set will be finite regardless of the dimension of the Hilbert space H. The proof of Lemma 3.1 as well as the relationship between ALD and KPCA was given in [32]. From Lemma 3.1, it can be guaranteed that when the ALD-based sparsification procedure is completed, there will be

a finite dictionary set for computing kernel-based feature vectors by using (55) and the kernel-based feature vectors are approximately linearly independent.

The convergence of KLSTD-Q is straightforward since it uses a batch least squares update to obtain the solutions. The remained problem is the approximation error between the true state-action value function $Q^*(x,a)$ and the solution based on the least squares regression equation (58). Since KLSTD-Q essentially implements linear TD learning using kernel-based feature vectors, (58) is equivalent to the regression equation of linear LSTD learning algorithms and the existing convergence and approximation error analysis of linear LSTD algorithms can be applied [7].

As noted in Section II, to simplify the notation, a countable state-action space is considered; however, the following results on TD-learning can also be extended to general spaces [7]. Let the cardinality of the state-action pairs be N. The kernel matrix can be denoted as

$$K = \left[k^d(s_1), k^d(s_2), \ldots, k^d(s_N)\right]^T \in R^{N\times d}. \tag{61}$$

Based on the analysis of temporal difference learning using linear basis functions in [7], Lemma 3.2 can be easily obtained.

Lemma 3.2: Let a^* be the weight vector determined by (58) and Q^* be the true state-action value function of the Markov chain. Let S be the space of state-action pairs. Let the assumptions A1)–A3) be satisfied.

A1) Markov chain $\{x_t\}$, whose states are produced by the state-action pairs of an MDP with a stationary policy, is ergodic and the transition probability matrix is P. There is a unique distribution π that satisfies $\pi P^T = \pi$ with $\pi(i) > 0$ for all $i \in S$ and π is a finite or infinite vector, depending on the cardinality of S.

A2) The transition rewards $r(x_t, x_{t+1})$ satisfy

$$E_0\left[r^2(x_t, x_{t+1})\right] < \infty \tag{62}$$

where $E_0[\cdot]$ is the expectation with respect to the distribution π.

A3) For every $i(i = 1, 2, \ldots, d)$, the basis function $k_i(x) = k(x_i, x)$

$$E_0\left[k_i^2(x_t)\right] < \infty. \tag{63}$$

Then, the following relation holds:

$$\|K\alpha^* - Q^*\|_D \le \frac{1-\lambda\gamma}{1-\gamma}\|\Pi Q^* - Q^*\|_D \tag{64}$$

where $D = \mathrm{diag}\{\pi_i\}$, $\Pi = K(K^TDK)^{-1}K^TD$, $0 \le \lambda \le 1$ ($\lambda = 0$ in KLSTD-Q), and $\|X\|_D = \sqrt{X^TDX}$.

Similar to the discussion in [7], assumptions A1)–A3) are easily satisfied and the linear independence assumption of basis functions is not necessary. From Lemma 3.2, we can see that the solution obtained by KLSTD-Q will approximate the real value functions with errors bounded by (64).

Lemma 3.3 [24]: Let $\pi[0], \pi[1], \pi[2], \ldots, \pi[m]$ be the sequence of policies generated by policy iteration algorithms and let $\widetilde{Q}_1, \widetilde{Q}_2 \ldots \widetilde{Q}_m$ be the corresponding approximate value functions. Let δ be a positive scalar that bounds the errors between the approximate and the true state-action value functions over all iterations

$$\forall m \qquad \|\widetilde{Q}_m - Q^*\|_\infty \le \delta. \tag{65}$$

Then, this sequence eventually produces policies whose performance is at most a constant multiple of δ away from the optimal performance

$$\limsup_{m\to\infty} \|\widetilde{Q}_m - Q^*\|_\infty \le \frac{2\gamma\delta}{(1-\gamma)^2}. \tag{66}$$

□

Based on the Lemmas 3.1–3.3, the convergence property of the proposed KLSPI algorithm can be described in Theorem 3.1.

Theorem 3.1: If the initial data samples $\{(x_i, a_i, r_i, x_{i+1}, a_{i+1})\}$ are generated by an MDP using a stationary initial policy, the policies produced by the KLSPI algorithm will at least converge to an area of policy space having suboptimal performance bounds determined by the approximation error of KLSTD-Q. Furthermore, if the approximation error becomes zero, KLSPI will converge to the optimal policy of the MDP.

Proof: In KLSPI, after the ALD-based kernel sparsification procedure, a finite dictionary set can be produced due to Lemma 3.1, and a set of approximately linearly independent basis functions can be formed. In the policy evaluation process, the data samples generated by the MDP using stationary policies can be transformed to the state transitions of corresponding Markov chains. Thus, in KLSTD-Q, kernel-based feature vectors are computed for the Markov chains, which is equivalent to the form of linear feature vectors. Then, due to Lemma 3.2, the approximation errors of KLSTD-Q are bounded by (64). At last, the convergence results of KLSPI can be established using Lemma 3.3.

IV. Experimental Results

In this section, three illustrative examples are given to show the effectiveness of the KLSPI algorithm. Since KLSPI is a substantial extension of the LSPI algorithm using manually selected basis functions, the performance of KLSPI and LSPI is compared in the experiments. The results clearly show that KLSPI usually converges in fewer iterations than LSPI and the optimal policies are easier to be produced in KLSPI due to the approximation and generalization ability of kernel methods in policy evaluation. Furthermore, the feature selection problems in LSPI are simplified since the kernel-based features can be automatically constructed by kernel sparsification and only very few parameters are to be selected for kernel functions. One of the experiments, i.e., the 20-state chain problem, was also studied in the previous work on LSPI and their publicly available simulation code is used in our experiments so that the performance comparison will be in the same experimental setup. Although the 20-state chain problem is a simple stochastic control problem, it was used both in [24] and in this paper to study the convergence behavior of approximated value functions and policies since the real optimal value functions and policies can be exactly computed when the model is known. To evaluate the effectiveness of KLSPI in feedback control

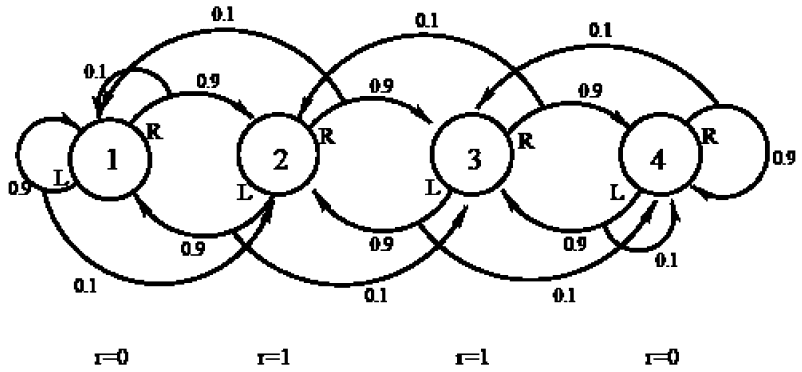

Fig. 2. Four-state problematic MDP [24].

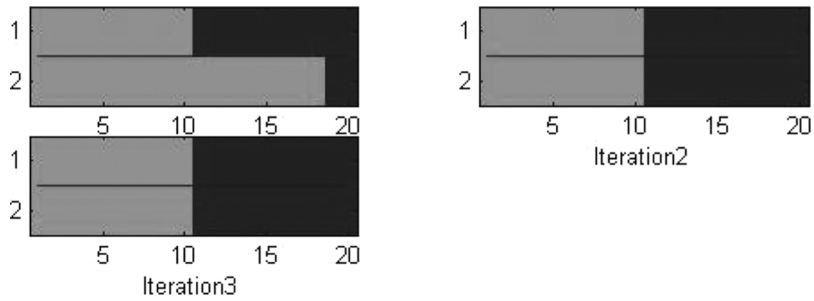

Fig. 3. Improved policy of KLSPI after each iteration (R action—dark shade; L action—light shade; KLSPI—top stripe; exact—bottom stripe). Top left—iteration 1. Bottom left—iteration 3. Right—iteration 2.

of complex dynamic systems, simulation experiments on the swing-up control of an underactuated double-link pendulum and the steering control of a tanker ship were conducted and the results show that the KLSPI algorithm can optimize controller performance by collecting data samples online and the performance of the learning control systems can be well guaranteed by incorporating conventional controller as initial policies.

A. The 20-State Chain Problem

The first experiment is a 20-state chain problem which is a problematic MDP noted in [24]. The MDP consists of a chain with 20 states (numbered from 1 to 20) and a simplified example with four states is shown in Fig. 2. For each state, there are two actions available, i.e., "left" (L) and "right" (R). Each action succeeds with probability 0.9, changing the state in the intended direction, and fails with probability 0.1, changing the state in the opposite direction. The two boundaries of the chain are dead-ends. For the four-state problem in Fig. 2, the reward vector over states is (0, $+1$, $+1$, 0) and the discount factor is set to 0.9. It is clear that the optimal policy is RRLL. In [33], a policy iteration method was used to solve the four-state problem, where LSTD was employed for state VFA. However, the resulting policies obtained in [33] only oscillated between the suboptimal policies RRRR and LLLL. The reason is mainly due to the limited approximation abilities of linear basis functions in policy evaluation.

The 20-state problem has the same dynamics as the four-state problem, except the reward of $+1$ given only at the boundaries (states 1 and 20). The optimal policy in this case is to go left in states 1–10 and right in states 11–20. In [24], LSPI was tested on the same problem. However, for the 20-state problem, careful selection of basis functions is required since the state space is larger than the four-state problem.

In the following, the experimental setup is the same as the experiments for LSPI in [24]. For the LSPI algorithm, a polynomial of 4° was used to approximate the value function for each of the two actions, giving a block of five basis functions per action. The two algorithms use a single set of samples collected from a single episode in which actions were chosen uniformly at random for 5000 steps. The performance is evaluated by the iterations for convergence as well as the ultimate policies after convergence. Figs. 3 and 4 show the improved policy after each iteration of KLSPI and LSPI, respectively. In Figs. 3 and 4, the both policies learned by KLSPI and the exact policies computed based on the model are depicted, where different colors correspond to different actions at every state. From Figs. 3 and 4, it is illustrated that although both algorithms converge to the optimal policy, KLSPI converges to the optimal policy only after three iterations while LSPI converges to the optimal policy after seven iterations. In fact, KLSPI finds the optimal policy only after two iterations, and then, it stabilizes in the optimal policy. Thus, compared with LSPI, KLSPI is much more efficient in convergence rates and little work is required on feature selection. In KLSPI, we use a radius basis function (RBF) kernel function and the unique parameter to be selected is the width σ of RBF. In the experiments, the RBF width is selected as $\sigma = 0.4$, which was simply tuned using a 1-D search process. The other parameter for KLSPI is the precision threshold μ of ALD-based kernel sparsification procedure. In all the following experiments, μ is equal to 0.001.

The convergence of KLSPI is mainly due to the powerful approximation and generalization ability of the kernel-based

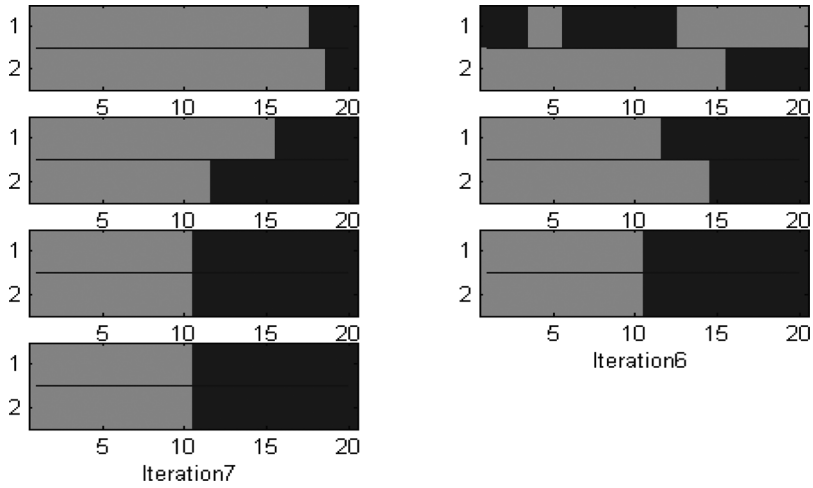

Fig. 4. Improved policy of LSPI after each iteration (R action—dark shade; L action—light shade; LSPI—top stripe; exact—bottom stripe). Top left—iteration 1. Bottom left—iteration 7. Others—iterations 2–6.

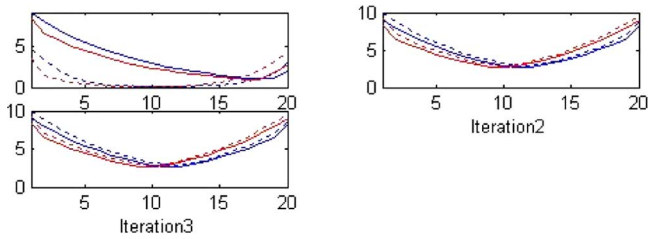

Fig. 5. State-action value function $Q(s, a)$ of the policy being evaluated in each iteration (KLSPI approximation—solid line; exact values—dotted line). Top left—iteration 1. Bottom left—iteration 3. Right—iterations 2.

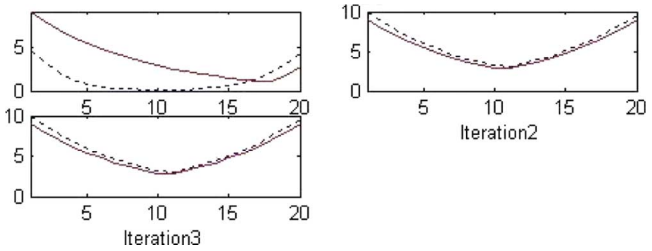

Fig. 7. State value function $V(s)$ of the policy being evaluated in each iteration (KLSPI approximation—solid line; exact values—dotted line). Top left—iteration 1. Bottom left—iteration 3. Right—iteration 2.

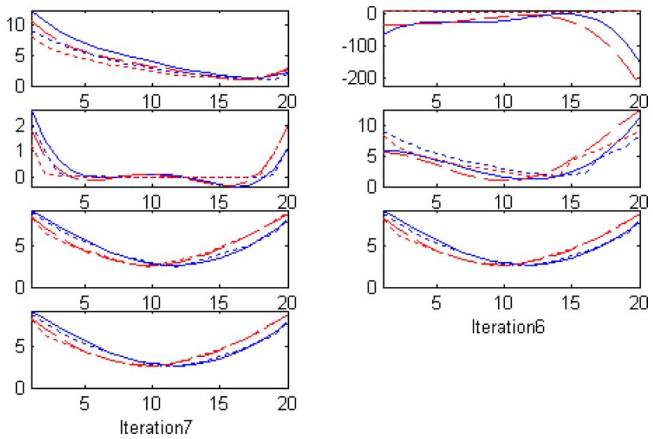

Fig. 6. State-action value function $Q(s, \pi(s))$ of the policy being evaluated in each iteration (LSPI approximation—solid line; exact values—dotted line). Top left—iteration 1. Bottom left—iteration 7. Others—iterations 2–6.

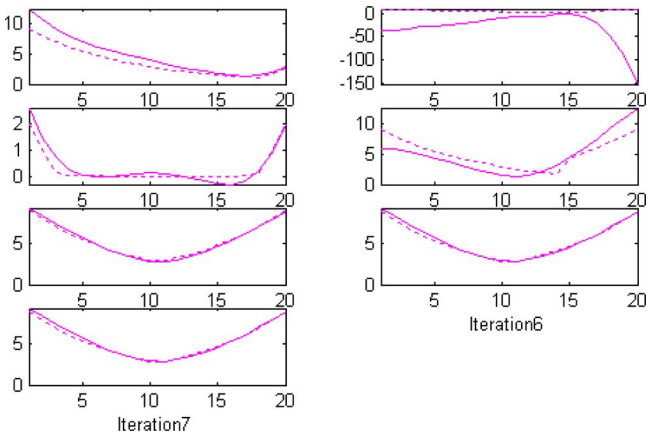

Fig. 8. State value function $V(s)$ of the policy being evaluated in each iteration (LSPI approximation—solid line; exact values—dotted line). Top left—iteration 1. Bottom left—iteration 7. Others—iterations 2–6.

policy evaluation using KLSTD-Q. This can be illustrated in Figs. 5–8, where the approximated value functions in each iteration as well as the exact values are plotted for both KLSPI and LSPI. In Fig. 5, it is shown that by using KLSTD-Q, KLSPI can approximate the exact state-action value functions with high precision so that it converges to the optimal policy in fewer iterations. In Fig. 6, it can be seen that there are relatively larger approximation errors in LSPI using LSTD-Q with manually selected basis functions, especially in the first four iterations. As the errors become smaller, LSPI also converges to the optimal policy but more iterations are needed.

In the experiments, the state-action pairs of the 20-state MDP have a total number of 40 since there are two actions available for each state, and every state-action pair was originally represented as a 2-D vector $[s_t, a_t]$, where the elements were normalized by a positive constant, i.e., $s_t \in \{1/20, 2/20, \ldots, 20/20\}$ and $a_t \in \{0.1, 0.2\}$. After the kernel sparsification process, a kernel-based feature vector, which has the dimension of 21,

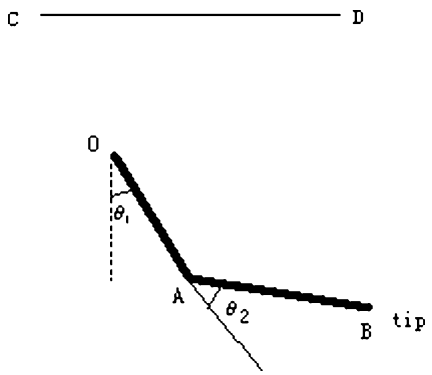

Fig. 9. Acrobot.

was automatically obtained from the 5000 training samples. Although the feature dimension of KLSPI $(d = 17)$ is larger than that of LSPI $(d = 10)$, they are both compact representations of the original state-action space $(d = 40)$ and the increase of computational costs in KLSPI is not significant when compared with the benefits of better approximation accuracy and better convergence behavior. Moreover, the features in KLSPI are automatically produced and optimized by the kernel sparsification procedure.

Figs. 7 and 8 make comparisons of the state value functions $V(s)$ approximated by KLSPI and LSPI. It is also clearly shown that KLSPI can approximate the true state value functions with smaller errors and converge within fewer iterations than LSPI.

B. Swing-Up Control of an Underactuated Double-Link Pendulum

In this section, a more difficult learning control problem, which is the swing-up control of an underactuated double-link pendulum in minimum time, is studied to evaluate the effectiveness of KLSPI. The underactuated double-link pendulum is also called an acrobot, which is a class of underactuated robots that has been widely studied in control engineering [40]–[42]. As shown in Fig. 9, the acrobot is a double-link pendulum (link OA and AB) moving on a vertical plane with only one actuator at the elbow. It has two equilibrium points, which are the stable straight-down equilibrium point and the unstable straight-up equilibrium point. The control objective is to swing up the acrobot from the stable equilibrium point to the neighborhood of the unstable equilibrium and balance it there. Because of the complexity of the problem, the control of the acrobot is usually divided into two phases which are the swing-up and the balancing control phases. In this paper, we will only consider the time-optimal swing-up control of the acrobot.

Until now, several approaches have been proposed for the controller design of the acrobot. In [40] and [41], nonlinear approximation and pseudolinearization methods were presented for the balancing control of the acrobot. In [42], a partial feedback linearization method was proposed for the swing-up control of an acrobot, together with a linear quadratic regulator (LQR) method to balance it. However, the previous methods greatly rely upon the dynamics model of the acrobot and it may take a long time to swing up the acrobot. The learning control of the acrobot is to realize time-optimal swing-up motion of the acrobot without knowing the exact dynamics of the system and it has been studied by researchers from the RL community. In [43], a SARSA-learning algorithm was applied to the problem, where a function approximator based on cerebellar model articulation controller (CMAC) was used. In [20], an improved RL algorithm using VFA has been proposed for the acrobot problem. However, all the aforementioned results lack rigorous convergence proofs and it still takes many learning episodes to obtain good performance. In the following, both LSPI and KLSPI will be applied to the swing-up control of the acrobot and their performance will be compared.

In Fig. 9, the first joint O is fixed to a bar and it cannot exert torques. The only control torque is applied at the second joint A. The control aim is to swing the acrobot up so that the tip point B is above the bar by an amount equal to one of the links. The time-optimal learning control problem is to swing up the acrobot in minimal time without knowing the dynamics of the system.

The dynamics model of the acrobot system is described by the following equations, which are only used for simulation:

$$\ddot{\theta}_1 = -(d_2\ddot{\theta}_2 + \phi_1)/d_1 \tag{67}$$

$$\ddot{\theta}_2 = (\tau + d_2\phi_1/d_1 - \phi_2) \tag{68}$$

where

$$d_1 = m_1 l_{c1}^2 + m_2\left(l_1^2 + l_{c2}^2 + 2l_1 l_{c2}\cos\theta_2\right) + I_1 + I_2 \tag{69}$$

$$d_2 = m_2\left(l_{c2}^2 + l_1 l_{c2}\cos\theta_2\right) + I_2 \tag{70}$$

$$\begin{aligned}\phi_1 = &-m_2 l_1 l_{c2}\dot{\theta}_2^2\sin\theta_2 - 2m_2 l_1 l_{c2}\dot{\theta}_1\dot{\theta}_2\sin\theta_2 \\ &+ (m_1 l_{c1} + m_2 l_1)g\cos(\theta_1 - \pi/2) + \phi_2\end{aligned} \tag{71}$$

$$\phi_2 = m_2 l_{c2} g\cos(\theta_1 + \theta_2 - \pi/2). \tag{72}$$

In (69)–(72), the parameters θ_i, $\dot{\theta}_i$, m_i, l_i, I_i, and l_{ci} are the angle, the angle velocity, the mass, the length, the moment of inertia, and the length of the center of mass for link $i(i = 1, 2)$, respectively.

Let s_T denote the goal state of the swing-up control. Since the control aim is to swing up the acrobot in minimum time, the reward function r_t is defined as

$$r_t = \begin{cases} 0, & \text{if } s = s_T \\ -1, & \text{else} \end{cases}. \tag{73}$$

In the simulation, the parameters for the acrobot are chosen as $m_1 = m_2 = 1$ kg, $I_1 = I_2 = 1$ kg·m^2, $l_{c1} = l_{c2} = 0.5$ m, $l_1 = l_2 = 1$ m, and $g = 9.8$ m/s^2. The control torque τ has three discrete values $[-1N, 0N, 1N]$. The time step for simulation is 0.05 s and the time interval for learning control is 0.2 s. A learning episode is defined as the period that starts from the stable equilibrium, i.e., $\theta_1 = \theta_2 = 0$, and ends when the goal state is reached or a maximum time step is accumulated. The performance of the algorithms is evaluated based on the time steps needed to swing up the acrobot.

As discussed in [20] and [43], the swing-up control problem of the acrobot can be modeled as an MDP with four-state variables θ_1, θ_2, $\dot{\theta}_1$, and $\dot{\theta}_2$. In our simulations, both LSPI and KLSPI are applied to learn a near-optimal policy to swing up

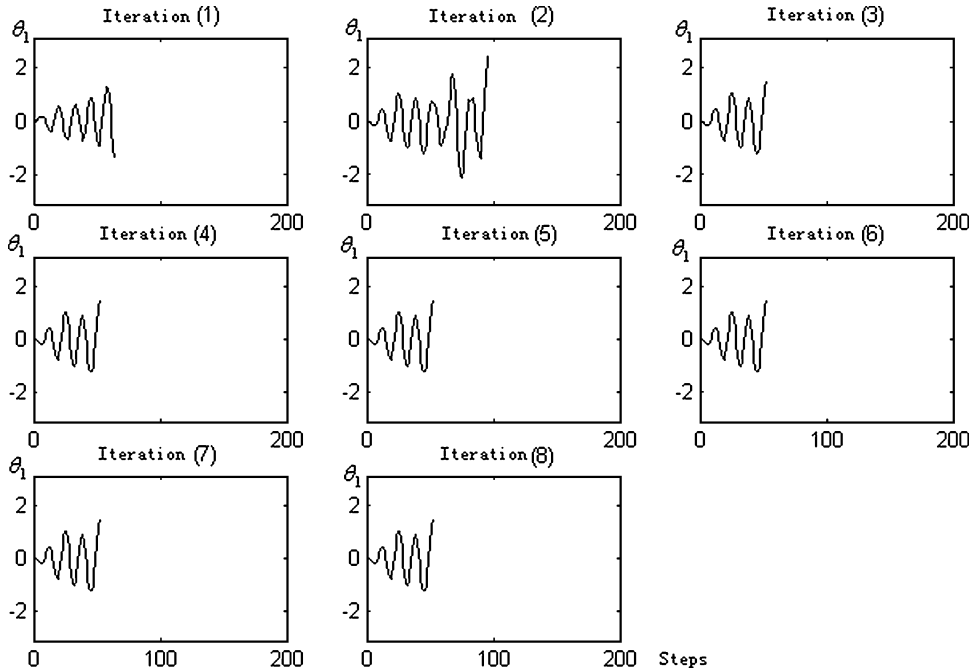

Fig. 10. Variations of θ_1 for the acrobot controlled by the policies of KLSPI at each iteration (the three subfigures at the first row are curves of θ_1 at iterations 1–3; the second row corresponds to iterations 4–6; and curves of θ_1 at iterations 7 and 8 are shown at the last row).

the acrobot as fast as possible. The initial training samples were generated by a random control policy, where 12 episodes of data were simulated and each episode had a maximum time step of 250. Thus, the maximum initial training samples may have $N = 3000$ samples and every sample has the form of $(x_i, a_i, r_i, x_{i+1}, a_{i+1})$, where $xi = (\theta_{1i}, \theta_{2i}, \dot{\theta}_{1i}, \dot{\theta}_{2i})$, $1 \leq i \leq N$. Each iteration of LSPI or KLSPI is composed of two stages, i.e., a data collecting and policy evaluation stage and a policy improvement stage. Although the initial policy can be constructed by an *a priori* controller to improve performance and speed of convergence, in the acrobot example, only random initial policies were consider to study the complete convergence behavior of LSPI and KLSPI. In the next example of tanker ship steering control, an initial controller using prior knowledge will be used to improve the online performance of KLSPI.

In the implementation of KLSPI, RBF kernel functions are used and the width parameter for RBF kernel is selected as $\sigma = 10$. The threshold parameter for ALD-based sparsification is set as $\mu = 0.001$. The discount factor is chosen as $\gamma = 0.9$. The simulation experiments were conducted for several independent runs with random initial policies. For every run of KLSPI, the learning control process of KLSPI consists of eight iterations. The simulation results show that KLSPI can obtain a very good control policy after very few iterations and it can stabilize or converge to a near-optimal policy very soon. A typical run of KLSPI with eight iterations is shown in Figs. 10 and 11, where the variations of the acrobot angles θ_1 and θ_2 are depicted, respectively. From Figs. 10 and 11, it is clearly shown that KLSPI converges to a near-optimal policy after four iterations and the obtained policy can swing up the acrobot with only 52 time steps, which is better than the near-optimal policies (around 70 steps) found by previous RL algorithms [20], [43].

To compare the performance between KLSPI and LSPI, learning control experiments of the acrobot were also conducted by making use of the LSPI method. A total number of 243 RBF basis functions of LSPI were selected, where there were three RBF functions for each dimension of state variables and action variables. The learning control process of LSPI also has several independent runs, where each run has ten iterations. In the simulations, the policies of LSPI usually cannot converge to a near-optimal policy, which can be seen in Fig. 12. The reason is mainly due to the improperly selected basis functions which have limited generalization ability. However, to manually select a set of good basis functions for LSPI in high-dimensional domains is difficult. For comparison, in the learning control process of Figs. 10 and 11, KLSPI can automatically construct a kernel-based feature vector of 131 dimensions.

As an adaptive learning control method based on approximate dynamic programming, KLSPI can be used to optimize controller performance without much *a priori* knowledge about the dynamics of control plants. In addition, since KLSPI can converge to a near-optimal policy with relatively small number of iterations or training episodes, it will be beneficial to use KLSPI for control systems with changing dynamics. The following Fig. 13 shows the variations of θ_1 for the acrobot when the mass of pole 1 changes from 1 kg to $m_1 = 1.5$ kg. It can be seen that KLSPI can quickly optimize the controller performance (after four iterations, the converged policy can swing up the acrobot within 49 steps) without knowing the exact change of plant dynamics, even if a randomly generated initial policy is used. However, to ensure online performance, especially the performance in the initial stage, a suitable controller can be used as the initial policy of KLSPI. This problem will be further studied in the following tanker ship heading control task,

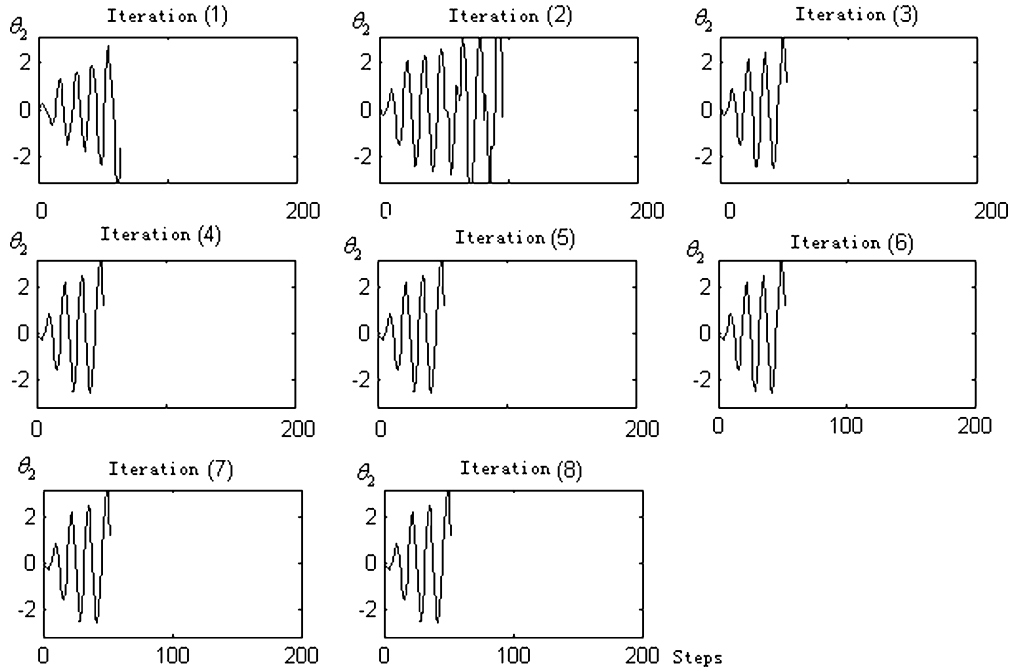

Fig. 11. Variations of θ_2 for the acrobot controlled by the policies of KLSPI at each iteration (the three subfigures at the first row are curves of θ_2 at iterations 1–3; the second row corresponds to iterations 4–6; and curves of θ_2 at iterations 7 and 8 are shown at the last row).

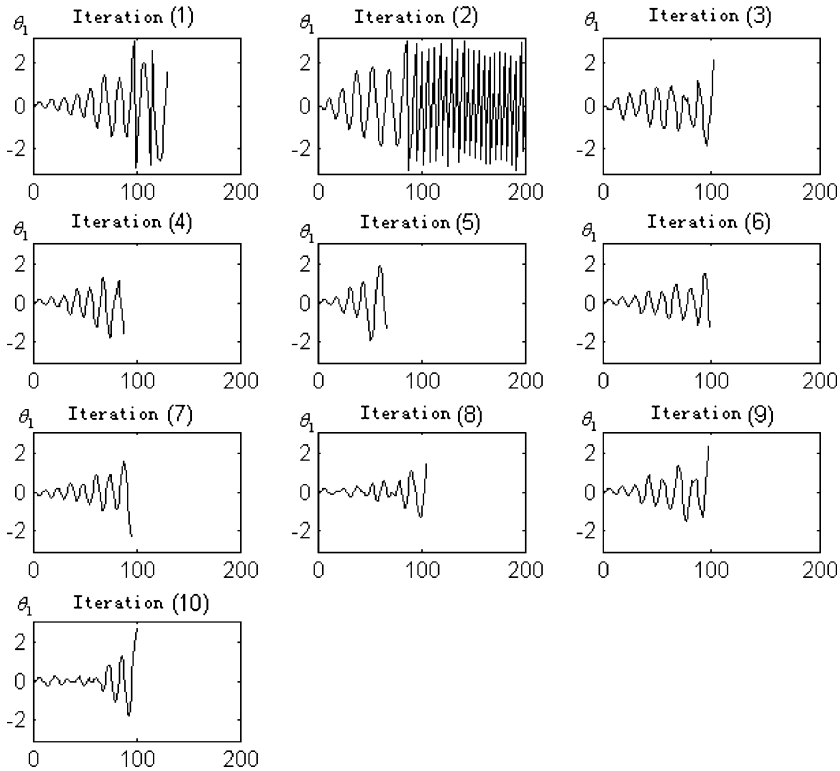

Fig. 12. Variations of θ_1 for the acrobot controlled by the policies of LSPI at each iteration (the three subfigures at the first row are curves of θ_1 at iterations 1–3; the second and third rows correspond to iterations 4– 6 and iterations 7–9, respectively; and the curve of θ_1 at iteration 10 is shown at the last row).

where an initial PD control structure will be used to improve online performance.

C. Adaptive Steering Control of a Tanker Ship

To improve fuel efficiency and reduce wear on ship components, autosteering systems have been developed and implemented for controlling the directional heading of ships. Traditional steering control systems of ships often make use of simple control schemes such as PID control. However, in many cases, the parameters of the ship steering controller have to be continually optimized. Such continual optimization is necessary since the dynamics of a ship change with the speed, trim, and loading

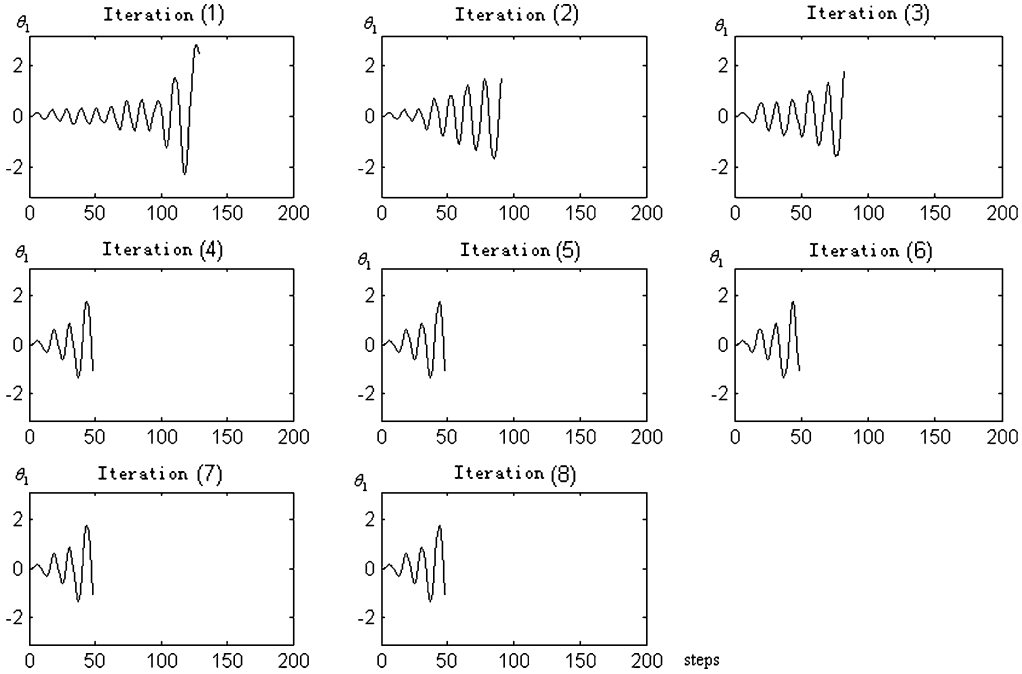

Fig. 13. Variations of θ_1 for the acrobot (the mass of pole 1 changes to $m_1 = 1.5$ kg) controlled by the policies of KLSPI at each iteration (the three subfigures at the first row are curves of θ_1 at iterations 1–3; the second row corresponds to iterations 4–6; and curves of θ_1 at iterations 7 and 8 are shown at the last row).

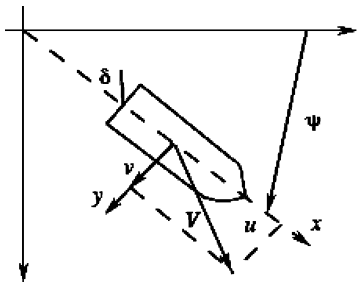

Fig. 14. Steering control of a tanker ship [46].

of the ship, and adaptive control law must be developed to compensate for large disturbances resulting from variations in the wind, waves, current, and water depth. Manual optimization of the controller parameters is often a burden on the crew and poor adjustment may be caused by human error. As a result, it is of great significance to develop a method for automatically optimizing or modifying the underlying controller for ship steering or heading control.

In the following, the adaptive steering control problem of a heavy tanker ship will be considered, which has been studied by many researchers from the control engineering community [44]–[46]. The tanker ship dynamics are usually modeled by applying Newton's laws of motion to the ship. For very large tanker ships, the motion in the vertical plane may be neglected since the vertical motion is very small for large tanker vessels. The motion of the ship is generally described by a coordinate system that is fixed to the ship [45], [46], which is shown in Fig. 14.

In Fig. 14, u is the forward velocity, v is the lateral velocity, ψ is the heading of the ship, and δ is the rudder angle, which serves as the control signal. Although a simplified linear model of the tanker ship may be used, it will be impractical for rudder angles that are larger than 5°. Therefore, the same nonlinear model studied in [45] was used in the simulation, which is given by

$$\dddot{\psi}(t) + \left(\frac{1}{\tau_1} + \frac{1}{\tau_2}\right)\ddot{\psi}(t) + \left(\frac{1}{\tau_1\tau_2}\right)H\left(\dot{\psi}(t)\right) = \frac{K}{\tau_1\tau_2}\left(\tau_3\dot{\delta}(t) + \delta(t)\right) \quad (74)$$

where K, τ_1, τ_2, and τ_3 are parameters that are a function of the ship's constant forward velocity u and its length l, and $H(\dot{\psi})$ is a nonlinear function of $\dot{\psi}$.

As discussed in [45], the nonlinear function $H(\dot{\psi})$ can be given by

$$H(\dot{\psi}) = \overline{a}\dot{\psi}^3 + \overline{b}\dot{\psi} \quad (75)$$

where $\overline{a}$ and $\overline{b}$ are real-valued constants such that $\overline{a}$ is always positive.

The relationships between K, $\tau_i (i = 1, 2, 3)$, u, and l are described as follows:

$$K = K_0\left(\frac{u}{l}\right) \quad (76)$$

$$\tau = \tau_{i0}\left(\frac{l}{u}\right), \qquad i = 1, 2, 3. \quad (77)$$

The previously described dynamics model of a tanker ship was further studied in [47], where a fuzzy control method and a neural control method based on RBF neural networks have been designed for the ship heading control problem. In this paper, the same tanker ship dynamics will be used for simulation but the KLSPI algorithm does not need model information as *a priori* and it can optimize controller performance only by collecting

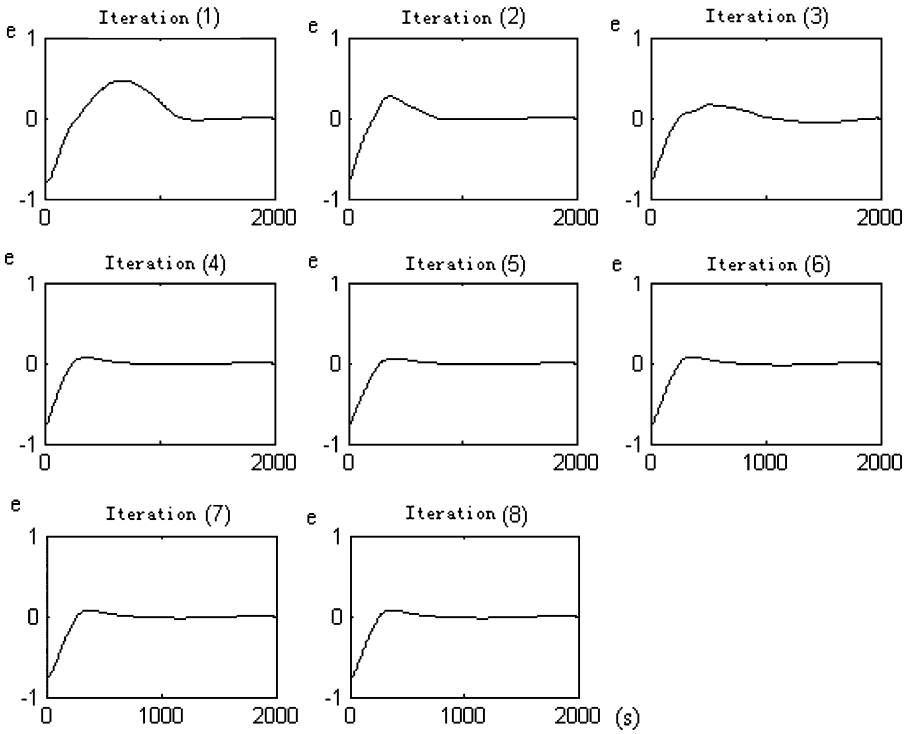

Fig. 15. Variations of heading error for the tanker ship controlled by the policies of KLSPI at each iteration (the three subfigures at the first row are error curves at iterations 1–3; the second row corresponds to iterations 4–6; and curves of heading error at iterations 7 and 8 are shown at the last row).

observation data samples. Thus, KLSPI needs little work on the modeling of the tanker ship and it will have advantages in realizing adaptive learning control of tanker ships under different conditions. The remaining problem is to ensure the online performance during the learning control process of KLSPI. In this paper, we propose to combine the KLSPI method with a PD control structure and make use of KLSPI to automatically select the PD parameters. Moreover, the initial PD parameters or candidate PD parameters to be selected can be designed using other conventional method and KLSPI is employed to improve the controller's performance further so that the online learning control process will have good performance.

Based on the aforementioned idea, the KLSPI controller has the following form:

$$\delta = z(e + W \cdot ec) \tag{78}$$

where e and ec are the heading error and the derivative of heading error, respectively, z is the action output of the KLSPI algorithm, and W is a constant. For the implementation of KLSPI, a 2-D vector (e and ec) is used as the state vector, and the action z is selected from three values $0.2, 1, 2$. Although continuous actions of KLSPI can also be considered by using a searching process for greedy action selection, in the simulation, we only focus on KLSPI with discrete actions since the convergence behavior and generalization performance are our main concerns. Furthermore, by combining KLSPI with a PD control structure, good controller performance can still be obtained for KLSPI with discrete actions.

In the simulation, an episode is defined as a period that starts from an initial state (0, 0) at $t = 0$ s and ends at $t = 200$ s. A desired heading angle of 45° is used during all the simulations. Therefore, the initial heading error (in radians) is $e = -45^{*}\mathrm{PI}/180 = -0.78$. The initial training samples were generated by a random control policy of z combined with the PD control structure in (78), where five episodes of data were simulated and each episode has a time step of 200. Thus, the maximum initial training samples have $N = 1000$ samples and every sample has the form of $(x_i, a_i, r_i, x_{i+1}, a_{i+1})$, where $x_i = (e_i, e_{ci})$, $1 \le i \le N$ and e_i and e_{ci} are the heading error and the derivative of heading error at time step t, respectively. An iteration of KLSPI is composed of a data collecting and policy evaluation stage, and a policy improvement stage. The simulation parameters are chosen as $\overline{a} = 1$, $\overline{b} = 1$, $K_0 = 5.88$, $\tau_{10} = -16.91$, $\tau_{20} = 0.45$, $\tau_{30} = 1.43$, and $l = 350$ m. The ship is assumed to travel in the x direction at a constant velocity of 5 m/s. The parameters for KLSPI include the RBF kernel width $\sigma = 0.25$, and the constant for PD control structure $W = 100$.

A typical run of KLSPI with eight iterations is shown in Figs. 15 and 16, where the variations of the ship heading error and its derivatives are depicted, respectively. From Figs. 15 and 16, it is demonstrated that KLSPI converges to a near-optimal policy after very few iterations and the obtained policy is much better than the initial policy provided.

To compare the performance of KLSPI with other methods, the tanker ship heading control problem was also simulated by applying the fuzzy control method and neural feedback controller studied in [47], as well as a conventional PD controller with constant PD parameters, which has the form of (78). The parameters of the fuzzy and RBF controllers are the same as those in the Matlab simulation code used in [47] and the PD control parameters are $z = 1$ and $W = 100$.

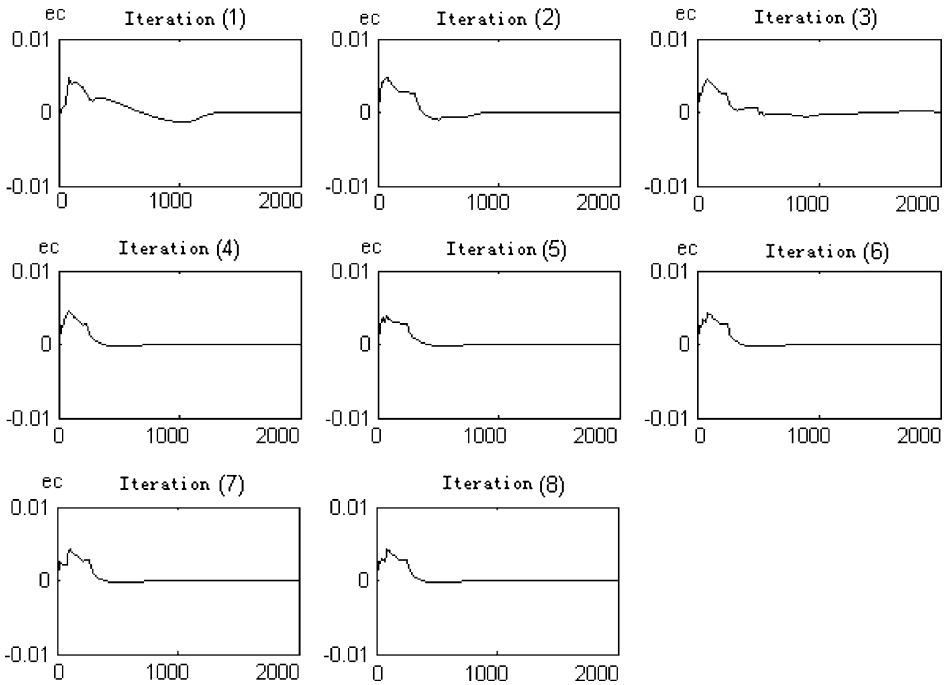

Fig. 16. Variations of the derivatives of heading errors for the tanker ship controlled by the policies of KLSPI at each iteration (the three subfigures at the first row are error derivative curves at iterations 1–3; the second row corresponds to iterations 4–6; and curves of heading error derivatives at iterations 7 and 8 are shown at the last row).

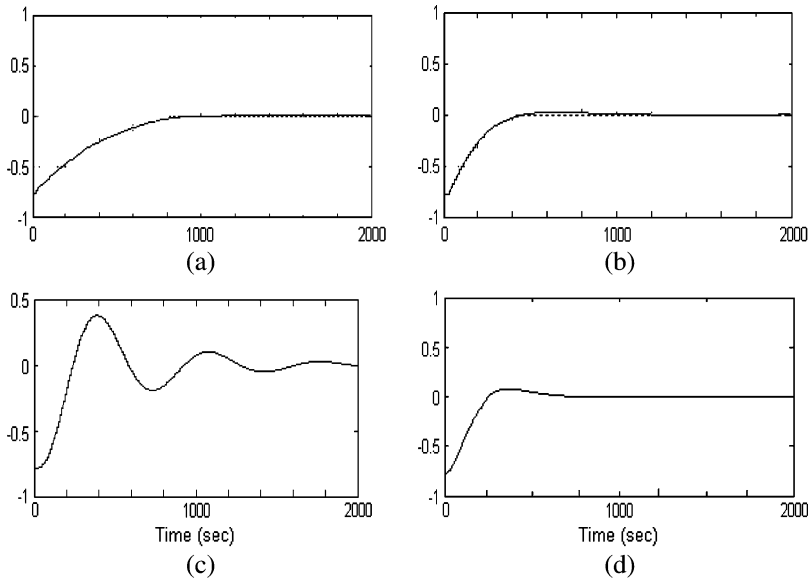

Fig. 17. Performance comparison between KLSPI combined with a PD control structure and other controllers for the ship heading control problem. (a) Fuzzy control. (b) RBF control. (c) PD control. (d) KLSPI+PD.

Fig. 17 shows the curves of the ship heading error under control by different methods. It can be seen from the results that the performance of the proposed KLSPI method is comparable to the fuzzy controller and the RBF controller studied in [47] and it is much better than a simple PD controller with manually selected parameters. Nevertheless, the main advantage of KLSPI is the adaptive learning ability with good convergence and generalization performance.

V. Related Work and Discussion

Recently, kernel machines are popularly studied to realize nonlinear and nonparametric versions of supervised or unsupervised learning algorithms. The main idea behind kernel machines is that an inner product in a high-dimensional feature space can be represented as a Mercer kernel function so that existing supervised or unsupervised learning algorithms in linear spaces can be transformed to nonlinear algorithms without explicitly computing the inner products in high-dimensional feature spaces. This idea, which is usually called the "kernel trick," has been widely used in supervised and unsupervised learning problems. In supervised learning, the most popular kernel machines include the SVMs and Gaussian processes (GPs), which have been successfully applied in many classification and regression problems and in most cases, kernel machines can ob-

tain better results or even the state-of-the-art performance. In unsupervised learning, KPCA was also addressed in many papers. Comprehensive reviews on kernel machines can be found in [28]–[30].

The combination of kernel methods with RL did not receive much attention several years ago since the regression or function approximation problem is more difficult in RL than supervised learning. One of the earliest works in this direction was published in 2002 [34], where kernel-based locally weighted averaging was used to approximate the state value functions of MDPs. However, as indicated in [24], by working directly on value-function space and performing the Bellman update and projection without any kind of model, LSPI as well as KLPSI make better use of function approximation than the kernel-based locally weighted averaging method. In recent years, there were studies done on applying GPs or SVMs to RL problems, such as GPs in TD(0) learning [35], SVMs for RL [36], and GPs in model-based policy iteration learning [37]. The work on GPs in TD learning (GPTD) [35] has some similarities with KLSTD-Q. Both GPTD and KLSTD-Q make use of kernel sparsification procedure based on ALD analysis. However, 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. KLSTD-Q is based on the least squares approach to approximate the state-action value functions and it can be effectively combined with policy improvement to produce highly efficient policy iteration algorithms with proven convergence. Furthermore, GPTD was only combined with the optimistic policy iteration (OPI) method and no convergence guarantee has been given. In [36], 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 GPs method in policy iteration learning [37] is aimed at realizing a model-based RL by approximating the plant dynamics and the state value functions of MDPs using two GP regression model. The GP-based policy iteration method uses support points, which are usually selected by manual discretization of the state spaces, for generalization in large or continuous spaces and the policy evaluation is performed using the state transition model approximated by a GP model. Thus, the analysis on the convergence of the GP-based policy iteration method is complex and has not been solved yet.

The most related work to this paper is the LSPI algorithm studied in [24]. The relationship between LSPI and KLSPI can be summarized as follows. On one aspect, the KLSPI algorithm can be viewed as a substantial extension of LSPI by introducing kernel-based features that are constructed by an ALD-based kernel sparsification procedure. This extension from manually selected features of LSPI to automatically constructed features eliminates one of the main obstacles to successful applications of LSPI. On the other hand, by using KLSTD-Q, the policy evaluation in KLSPI can efficiently approximate the state-action value functions with high precision; therefore, the convergence as well as the policy optimality of KLPSI is better guaranteed than LSPI.

VI. Conclusion and Future Work

As a class of model-free approximate dynamic programming methods, RL is very promising when applied in many complex sequential decision problems where conventional mathematical programming and supervised learning methods cannot be used, either due to the lack of model information or the model being too complex. However, there are several difficulties for wide applications of current RL theory and algorithms, e.g., generalization and convergence in large-scale MDPs. Despite the recent research work that has been devoted to solving these problems, especially the LSPI algorithm in [24] and kernel methods in RL [35]–[37], there still remain two difficulties. One difficulty is the convergence guarantee to the optimal or near-optimal policies when various function approximators are employed. The other difficulty is the feature selection for improving generalization and learning efficiency. The KLSPI algorithm proposed in this paper makes two improvements to eliminate these two difficulties. One is the convergence and (near) optimality guarantee based on the KLSTD-Q algorithm for policy evaluation with high precision. The other is the automatic feature selection using the ALD-based kernel sparsification method. Experimental results demonstrate that KLSPI can converge to the optimal or near-optimal policies within fewer iterations and with smaller size of sample sets than the previous LSPI. Thus, KLSPI provides a general RL method with generalization performance and convergence guarantee for large-scale MDPs.

For applying KLSPI to feedback control of dynamic systems, where online learning and optimization with performance guarantees are usually required, it is not necessary for the random data gathering stages in which random/intermediate controls are applied to the system. Instead, a suitable initial policy or controller such as a PD controller can be used, and KLSPI is able to optimize the controller performance incrementally based on the data gathered from the initial policy. Therefore, KLSPI can also be used in online learning control problems when an initial controller is provided to guarantee the online performance and stability. The simulation experiments on the heading control of a tanker ship have shown that the online learning control process of KLSPI can be realized in a periodical way, i.e., data gathering and learning optimization are both performed online but alternatively. Compared to other neural network methods for feedback control such as ACDs [38], KLSPI uses kernel methods and VFA to implicitly establish the behavior model of uncertain dynamic systems and the controller performance is optimized via approximate dynamic programming based on policy iteration. Although the results are very encouraging, the applications of KLSPI in more real-world problems, as well as comprehensive comparisons between KLSPI and ACDs, are to be studied in future work, not only to provide optimized solutions of complex sequential decision problems but also for further verification and improvement of the KLSPI algorithm.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions, which greatly improved the quality of this paper. They would also like to thank Dr. M. G. Lagoudakis, Dr. R. Parr, and Prof. K. M. Passino for their publicly available simulation codes for LSPI and tanker ship heading control.

REFERENCES

[1] R. Sutton and A. Barto, *Reinforcement Learning, an Introduction*. Cambridge, MA: MIT Press, 1998.

[2] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, 1996.

[3] R. Sutton, "Learning to predict by the method of temporal differences," *Mach. Learn.*, vol. 3, no. 1, pp. 9–44, 1988.

[4] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, pp. 279–292, 1992.

[5] J. N. Tsitsiklis, "Asynchronous stochastic approximation and Q-learning," *Mach. Learn.*, vol. 16, pp. 185–202, 1994.

[6] P. Dayan and T. J. Sejnowski, " TD(λ) converges with probability 1," *Mach. Learn.*, vol. 14, pp. 295–301, 1994.

[7] J. N. Tsitsiklis and B. V. Roy, "An analysis of temporal difference learning with function approximation," *IEEE Trans. Autom. Control*, vol. 42, no. 5, pp. 674–690, May 1997.

[8] S. P. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvari, "Convergence results for single-step on-policy reinforcement-learning algorithms," *Mach. Learn.*, vol. 38, pp. 287–308, 2000.

[9] D. P. Bertsekas and J. N. Tsitsiklis, *Neurodynamic Programming*. Belmont, MA: Athena Scientific, 1996.

[10] J. Moody and M. Saffell, "Learning to trade via direct reinforcement," *IEEE Trans. Neural Netw.*, vol. 12, no. 4, pp. 875–889, Jul. 2001.

[11] J. Baxter and P. L. Bartlett, "Infinite-horizon policy-gradient estimation," *J. Artif. Intell. Res.*, vol. 15, pp. 319–350, 2001.

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

[13] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press, 2000, vol. 12.

[14] X. Xu, H. G. He, and D. W. Hu, "Efficient reinforcement learning using recursive least-squares methods," *J. Artif. Intell. Res.*, vol. 16, pp. 259–292, 2002.

[15] J. Boyan, "Technical update: Least-squares temporal difference learning," *Mach. Learn.*, vol. 49, no. 2–3, pp. 233–246, 2002.

[16] R. H. Crites and A. G. Barto, "Elevator group control using multiple reinforcement learning agents," *Mach. Learn.*, vol. 33, no. 2–3, pp. 235–262, 1998.

[17] G. Tesauro, "TD-Gammon, a self-teaching backgammon program, achieves master-level play," *Neural Comput.*, vol. 6, pp. 215–219, 1994.

[18] W. Zhang and T. Dietterich, "A reinforcement learning approach to job-shop scheduling," in *Proc. 14th Int. Joint Conf. Artif. Intell.*, 1995, pp. 1114–1120.

[19] L. Baird, "Residual algorithms: Reinforcement learning with function approximation," in *Proc. 12th Int. Conf. Mach. Learn.*, San Francisco, CA, Jul. 9–12, 1995, pp. 30–37.

[20] X. Xu and H. G. He, "Residual-gradient-based neural reinforcement learning for the optimal control of an acrobot," in *Proc. IEEE Int. Symp. Intell. Control*, Vancouver, BC, Canada, Oct. 2002, pp. 758–763.

[21] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, pp. 229–256, 1992.

[22] L. Baird and A. Moore, "Gradient descent for general reinforcement learning," in *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press, 1999, vol. 11.

[23] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press, 2000, vol. 12.

[24] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *J. Mach. Learn. Res.*, vol. 4, pp. 1107–1149, 2003.

[25] M. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: Wiley, 1994.

[26] J. N. Tsitsiklis, "Asynchronous stochastic approximation and Q-learning," *Mach. Learn.*, vol. 16, pp. 185–202, 1994.

[27] X. Xu, T. Xie, D. W. Hu, and X. C. Lu, "Kernel least-squares temporal difference learning," *Int. J. Inf. Technol.*, vol. 11, no. 9, pp. 54–63, 2005.

[28] V. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.

[29] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*. Cambridge, U.K.: Cambridge Univ. Press, 2000.

[30] B. Schölkopf and A. Smola, *Learning With Kernels*. Cambridge, MA: MIT Press, 2002.

[31] Y. Engel, S. Mannor, and R. Meir, "Sparse online greedy support vector regression," in *Proc. 13th Eur. Conf. Mach. Learn.*, 2002, pp. 84–96.

[32] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least-squares algorithm," *IEEE Trans. Signal Process.*, vol. 52, no. 8, pp. 2275–2285, Aug. 2004.

[33] D. Koller and R. Parr, "Policy iteration for factored MDPs," in *Proc. 16th Conf. Uncertainty Artif. Intell.*, Stanford, CA, 2000, pp. 326–334.

[34] D. Ormoneit and S. Sen, "Kernel-based reinforcement learning," *Mach. Learn.*, vol. 49, no. 2–3, pp. 161–178, 2002.

[35] Y. Engel, S. Mannor, and R. Meir, "Bayes meets Bellman: The Gaussian process approach to temporal difference learning," in *Proc. Int. Conf. Mach. Learn.*, 2003, pp. 154–161.

[36] T. G. Dietterich and X. Wang, "Batch value function approximation via support vectors," in *Advances in Neural Information Processing Systems 14*. Cambridge, MA: MIT Press, 2002, pp. 1491–1498.

[37] C. E. Rasmussen and M. Kuss, "Gaussian processes in reinforcement learning," in *Advances in Neural Information Processing Systems 16*, S. Thrun, L. K. Saul, and B. Schölkopf, Eds. Cambridge, MA: MIT Press, 2004, pp. 751–759.

[38] D. V. Prokhorov and D. C. Wunsch, "Adaptive critic designs," *IEEE Trans. Neural Netw.*, vol. 8, no. 5, pp. 997–1007, Sep. 1997.

[39] G. K. Venayagamoorthy, R. G. Harley, and D. C. Wunsch, "Comparison of heuristic dynamic programming and dual heuristic programming adaptive critics for neurocontrol of a turbogenerator," *IEEE Trans. Neural Netw.*, vol. 13, no. 3, pp. 764–773, May 2002.

[40] J. Hauser and R. M. Murray, "Nonlinear controllers for non-integratable systems: The acrobot example," in *Proc. Amer. Control Conf.*, San Diego, CA, 1990, pp. 669–671.

[41] S. Bortoff and M. W. Spong, "Pseudolinearization of the acrobot using spline functions," in *Proc. IEEE Conf. Decision Control*, Tucson, AZ, 1992, pp. 593–598.

[42] M. W. Spong, "The swing up control problem for the acrobot," *IEEE Control Syst. Mag.*, vol. 15, no. 1, pp. 49–55, Feb. 1995.

[43] R. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Advances in Neural Information Processing Systems 8*. Cambridge, MA: MIT Press, 1996, pp. 1038–1044.

[44] K. M. Passino, "Intelligent control: an overview of techniques," in *Perspectives in Control Engineering: Technologies, Applications, and New Directions*, T. Samad, Ed. New York: IEEE Press, 2001, pp. 104–133.

[45] K. M. Passino and S. Yurkovich, *Fuzzy Control*. Reading, MA: Addison-Wesley, 1998.

[46] K. S. Narendra and A. M. Annaswamy, *Stable Adaptive Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[47] K. M. Passino, *Biomimicry for Optimization, Control, and Automation*. London, U.K.: Springer-Verlag, 2005.

Xin Xu was born in Hubei, P. R. China, in 1974. He received the B.S. degree in control engineering from the Department of Automatic Control, National University of Defense Technology (NUDT), Changsha, P. R. China, in 1996 and the Ph.D. degree in electrical engineering from the College of Mechantronics and Automation (CMEA), NUDT.

From 2003 to 2004, he was a Postdoctoral Fellow at School of Computer, NUDT. Currently, he is an Associate Professor at the Institute of Automation, CMEA, NUDT. He has coauthored four books and published more than 40 papers in international journals and conferences. His research interests include reinforcement learning, data mining, learning control, robotics, autonomic computing, and computer security.

Dr. Xu received the excellent Ph.D. dissertation award from Hunan Province, P. R. China, in 2004. He has served as a Session Chair in many international conferences, and currently, he is a reviewer for many journals.

Dewen Hu (M'03–SM'06) was born in Hunan, P. R. China, in 1963. He received the B.Sc. and M.Sc. degrees from Xi'an Jiaotong University, Xi'an, China, in 1983 and 1986, respectively, and the Ph.D. degree from the National University of Defense Technology, Changsha, P. R. China, in 1999, all in automatic control.

From 1986, he was with the National University of Defense Technology. From October 1995 to October 1996, he was a Visiting Scholar at the University of Sheffield, Sheffield, U.K. He was promoted Professor in 1996. He published over 180 papers and three monographs in the areas of his interests. His research interests include image processing, system identification and control, neural networks, and cognitive science.

Dr. Hu is the joint recipient of more than a dozen academic prizes for his research on neurocontrol, intelligent robot, and image processing. Currently, he is an action editor of *Neural Networks*.

Xicheng Lu was born in Jiangsu, P. R. China, in 1946. He received the B.Sc. degree in computer science from Harbin Military Engineering Institute, China, in 1970.

From September 1982 to September 1984, he was a Visiting Scholar at the University of Massachusetts. Since 1978, he has been with the School of Computer, National University of Defense Technology, Changsha, P. R. China. He became a Professor in 1992. His research interests include distributed computing, computer networks, high-performance computing, etc. He has over 120 papers and three books in the areas of his interests.

Prof. Lu is an academician of Chinese Academy of Engineering and he is the joint recipient of four First Class National Scientific and Technological Progress Prize in his research area.