Reinforcement Learning with Kernel Recursive Least-Squares Support Vector Machine

Hitesh Shah and M. Gopal

***Abstract*—A reinforcement learning system based on the kernel recursive least-squares algorithm for continuous state-space is proposed in this paper. A kernel recursive least-squares- support vector machine is used to realized a mapping from state-action pair to *Q*-value function. An online sparsification process that permits the addition of training sample into the *Q*-function approximation only if it is approximately linearly independent of the preceding training samples. Simulation result of two-link robot manipulator show that the proposed method has high learning efficiency – better accuracy measured in terms of mean square error, and lesser computation time compare to the least-squares support vector machine.**

***Index Terms*—Kernel methods, least-squares support vector machine, recursive least squares, reinforcement learning.**

I. Introduction

Support vector machine (SVM), which is based on Vapnik's structural risk minimization (SRM) [1], has become one of the most popular methods in solving classification and regression problems. Conventional SVMs have properties of global optimization, and good adaptability. However, the optimal solutions are obtained by solving standard quadratic programming which results in high computational cost. In order to reduce the computational cost, a least square support vector (LS-SVM) was proposed in [2] by converting inequality constraints to linear equations. LS-SVM has been successfully applied to reinforcement learning (RL) problems [3]. A RL problem is converted into a regression problem, wherein the observed states and actions are considered as inputs and *Q*-value functions as output. All the training samples may be support vectors in LS-SVM, and thus the support vectors are no longer sparse. It may lead to poor generalization. In addition, with the number of the input training pairs increasing, the number of equations will increase which may result in higher computational cost.

Our focus in this paper is on reinforcement learning problems. The objective in hand is to explore the use of a support vector machine with sparse support vectors, low computational cost and satisfactory accuracy. The kernel recursive least-squares (KRLS) – SVM [4], is a strong candidate to achieve this objective. We develop a KRLS-SVM algorithm for reinforcement learning and demonstrate its potential through case study – two-link robot manipulator. The mean square error accuracy, computational cost and robustness properties of this scheme are compared with the scheme based on LS-SVM.

The paper is organized as follows. Section II presents architecture of Q-learning system based on KRLS-SVM. Section III gives details of on-line KRLS-SVM learning algorithm. Section IV compares and discusses the empirical performance study on the basis of simulation results. Additionally this section highlights the features of KRLS-SVM in comparison with LS-SVM. Finally in Section V, the conclusions are presented.

Manuscript received July 9, 2012; revised August 8, 2012.

The authors are with Department of Electrical Engineering, Indian Institute of Technology – Delhi, New Delhi, India (e-mail: iitd.hitesh@gmail.com; mgopal@ee.iitd.ac.in).

II. *Q*-Learning System Based on KRLS-SVM

In reinforcement learning paradigm, an agent (controller) must learn from interaction with its environment (plant) in order to achieve certain goals. The goal of RL agent is to estimate the optimal policy or optimal value function for Markov Decision Process (MDP) without knowing its model. In order to do so, most current RL techniques estimate the value of actions, i.e., the future reward one can expect as a result of executing an action, using recursive estimation techniques. One of the most well-known RL approaches is *Q*-learning [5]. A *Q*-learning system observes the state transition s^t to s^{t+1} with an execution of an action a_t at each time step *t*, and receives an immediate reward r_t. Watkins (1989) proposed a procedure to iteratively update *Q*-values that does not require a system model, and is given by:

$$Q(s^t, a_t) \leftarrow Q(s^t, a_t) + \eta(r_t + \gamma V(s^{t+1}) - Q(s^t, a_t)) \quad (1)$$

The learning rate parameter $0 \le \eta \le 1$ is reduced to a small value at a suitable rate, and γ is the discount factor used to determine the proportion of the delay to the future rewards. The recursive algorithm represented by (1) is guaranteed to converge to the true value function if certain assumptions are respected [6]. One of the assumptions is that *Q*-values are stored in a lookup table, with single entry for each state-action pair.

Support vector machine function approximation (SVM*Q*-learning) is one of the RL frameworks to deal with continuous space problem. Conventional SVMs have properties of global optimization, and good adaptability. However, the optimal solutions are obtained by solving standard quadratic programming which is numerically inefficient. A Least Squares Support Vector Machine (LS-SVM) has good generalization property, and is used to approximate the *Q*-values of state-action pairs online by taking the advantage of not falling into the trap of local minima. On the other hand, LS-SVM suffers from poor sparsification of support vectors and results in higher

computational cost when number of training pairs increases. A new *Q*-learning method on continuous state domains based on Kernel Recursive Least-Squares Support Vector Machine (KRLS-SVM) is proposed in this paper. Fig.1 shows the architecture of the *Q*-learning system based on KRLS-SVM.

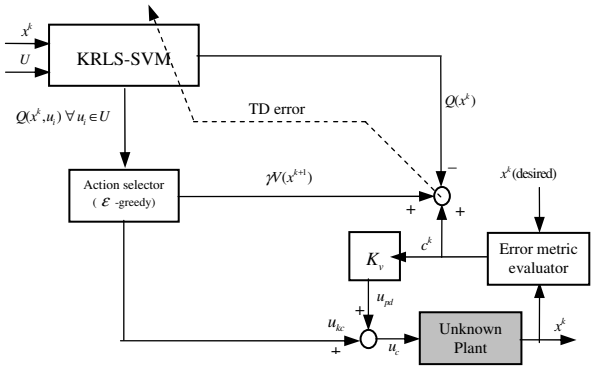

Fig. 1. KRLS-SVM architecture

In Fig.1, control action set is denoted as $U=\{u_k\};\ k=1,\ldots,m$, where m is the number of possible discrete control actions. A series of state-action pairs $\left(x^k,u_k\right)$ comprised of each action u_k in action set and the current system state $x^k=\left\{x_1^k,x_2^k,\ldots,x_n^k\right\}$ can be constituted. The input of KRLS-SVM is the state-action pair (x^k,u_k), while the output is the estimated *Q*-value corresponding to (x^k,u_k). Training samples of KRLS-SVM should be obtained during interaction between the learning system and its environment. In specific, control actions are selected using an exploration/exploitation policy (EEP) in order to explore the set of possible actions and acquire experience through the RL signals [7]. We use a pseudo-stochastic exploration as in [8]. In pseudo-stochastic exploration, we gradually reduce the exploration (determined by the ε parameter) according to some schedule; we have reduced ε to its 90 percent value after every 50 iterations. The lower limit of parameter ε has been fixed at 0.002 (to maintain exploration).

III. Online KRLS-SVM Learning

The Kernel Recursive Least Squares (KRLS) algorithm was introduced in [4] and has a conceptual foundation related to Principal Component Analysis (PCA) and Support Vector Machines (SVM). KRLS algorithm produces much sparser solutions with higher robustness to noise. Moreover KRLS is a fully online algorithm designed to operate in real-time environment where data became available one sample at a time. As a matter of fact, KRLS seems to be the best choice as a function approximator in reinforcement learning algorithm.

In our setting, KRLS is presented with input-output pairs i.e., state-action pair with estimated *Q*-value, arising from an unknown mapping. The standard recursive least-square (RLS) algorithm is used to recursively train a linear regression model [9], which can be expressed in the following parametric form:

$$Q=\hat{f}(x)=\left\langle w,\phi(x)\right\rangle+b;\quad w\in\mathbb{R}^n,b\in\mathbb{R} \tag{2}$$

where, $\phi(x)$ is a fixed, finite dimensional nonlinear mapping from input space to some high-dimensional feature space, $\langle .,. \rangle$ denotes inner products, and w is a weight vector of parameters that can be adjusted in a manner such that the bias term b becomes zero. In this case, regression model reduces to the simpler form as:

$$Q=\hat{f}(x)=\left\langle w,\phi(x)\right\rangle=\phi(x)^T w\quad ;w\in\mathbb{R}^n \tag{3}$$

The objective of learning algorithm is to minimize, $g(w)=\sum_{i=1}^{n}(y_i-\hat{f}(x_i))^2$ with respect to the w weight vector. The optimal weight vector can be expressed as, $w=\sum_{i=1}^{n}\alpha_i\phi(x_i)$ and the regression model becomes,

$$Q=\hat{f}(x)=\phi(x)^T\phi(x)\alpha \tag{4}$$

Kernel methods present an alternative to the parametric approach. KRLS attempts to learn an approximation to the mapping $\hat{f}(x)$ in the form of a weighted linear sum of the kernels $k(x_i,x)=\left\langle\phi(x_i),\phi(x)\right\rangle$, where $\{x_i\}_{i=1}^{t}$ are the training data points up to time t. This leads to

$$Q=\hat{f}(x)=\sum_{i=1}^{t}\alpha_i k(x_i,x) \tag{5}$$

The vector x_i associated with coefficients $\alpha_i>0$ are called Support Vectors and only these contribute to minimizing the cost function. In general, Radial Basis kernel function is the most common choice for nonlinear system study.

In order to reduce the number of adjustable parameters in (5), KRLS-SVM employs a form of *online sparsification*. By making use of online sparsification, the training data can be stored in a compact form, i.e., only a fraction of training data will be actually used for training purpose. The sparsification methodology permits the addition of training sample into the approximation (5) only if it is approximately linearly independent of the preceding training samples. In the sparsification procedure, the linearly independent training data points will be stored in a Dictionary Set. To prove the linear dependency of new data vector on the dictionary vectors, the *approximate linear dependences* (ALD) test is

$$\delta_t=\min_a\left\|\sum_{j=1}^{t-1}a_j\phi(x_j)-\phi(x_t)\right\|^2<\upsilon \tag{6}$$

where υ is the dictionary-inclusion threshold, which is an important tuning parameter that determines the accuracy of approximation (5). Solving for optimality of the cost function, we can get: $a_t=\tilde{\mathrm{K}}_{t-1}^{-1}\tilde{\mathrm{k}}_{t-1}(x_t)$ and $\delta_t=k_{tt}-\tilde{\mathrm{k}}_{t-1}(x_t)^{\mathrm{T}}a_t$, where $[\tilde{\mathrm{K}}_{t-1}]_{i,j}=k(\tilde{x}_i,\tilde{x}_j)$, $[\tilde{\mathrm{k}}_{t-1}(x_t)]_i=k(\tilde{x}_i,x_j)$ and $k_{tt}=k(\tilde{x}_t,\tilde{x}_t)$. While updating the weight vector α online, if $\delta_i<\upsilon$, the new training data point will not be added in the

dictionary set. But if $\delta_t > \upsilon$ then new training data point will be added in the dictionary set. As a result, the weight vector $\alpha_t = (\alpha_1, \alpha_2, \ldots \alpha_{m_{t-1}})$ are learned by KRLS over time through successive minimization of the approximation error in the least-squares sense. We use KRLS algorithm as proposed in [4] for reinforcement learning. Q-learning method based on KRLS-SVM can be summarized as follows:

Step (1) Initialize the KRLS-SVM model with the kernel function, variance of Gaussian, and linear dependence threshold.

Step (2) Start the simulation to construct a series of state-action pairs (x^k, u) comprising of each action u in action set U and current state x^k.

Step (3) Add this training set into dictionary set and compute the kernel weight vector (alpha).

Step (4) Obtain Q-values $Q(x^k, u_k)$ corresponding to (x^k, u_k) for each action u_k in action set U by solving regression model of the KRLS-SVM, send them to a ε-greedy action selector and obtain the actual action.

Step (5) Perform actual action and obtain reward and successor state x^{k+1}.

Step (6) Update the Q-value according to the Eq. (1), to obtain the target Q-value.

Step (7) With the next sample of training set, perform the approximate linear dependence (ALD) test for it.

Step (8) If ALD test error is less than the threshold value, go to step (10).

Step (9) Add the new sample to dictionary set. Update the kernel weight vector, and go to step (11).

Step (10) Keep dictionary set unchanged. Update the kernel weight vector.

Step (11) If training set has any element left, go to step (7).

Train the KRLS-SVM model and assign $x^k = x^{k+1}$. Repeat the procedure for on-line learning.

IV. Simulation Studies and Analysis

To verify the proposed KRLS-SVM learning approach, we use two-link robot manipulator tracking control problem as the standard bench mark.

In robot-manipulator tracking control problem, we try to train the kernel recursive least square support vector machine so that its outputs can track those of an unknown dynamic system over the time interval [0, T]. The dynamical model of the two-link robotic manipulator and parameters as specified in [10], have been used in this simulation study.

A. Controller Learning Details

Simulation parameters and learning details for KRLS-SVM value function approximator in reinforcement learning control structure are as follows:

We define tracking error vector as: $e^k = \theta_d^k - \theta^k$ and cost function $c^k = \dot{e}^k + \Lambda e^k, \Lambda = \Lambda^T > 0$ with $\Lambda = \text{diag}\{30,20\}$. Maximum limit of error is taken as 0.2 rad for both the links (10% of peak-to-peak of reference trajectory). System state space (continuous) has four variables, i.e., $x^k = [\theta_1 \;\; \theta_2 \;\; \dot{\theta}_1 \;\; \dot{\theta}_2]^T = [x_1 \;\; x_2 \;\; \dot{x}_1 \;\; \dot{x}_2]^T$. Controller action sets for link1 and link2 are $|U|(1) = [-20 \;\; 0 \;\; 20]$ Nm, and $|U|(2) = [-2 \;\; 0 \;\; 2]$ Nm, respectively. Exploration level ε decays from $0.5 \rightarrow 0.002$ over the iterations. The discount factor γ is set to 0.8; learning-rate parameter η is set to 0.2, and PD gain matrix $K_v = \text{diag}\{20, 20\}$. We deliberately introduce deterministic noise of ±1% in control effort with a probability of (1/3), for stochastic simulation.

A KRLS-SVM is used to realize mapping from state-action pair to Q-value function. For simplicity, the controller uses two function approximators, one each for the two links. The training samples of KRLS-SVM are obtained during the interaction between the controller and the environment. The Gaussian kernel $K(x, y) = e^{(-((x-y)/\rho)^2)}$ is chosen for our simulation studies. The variance of Gaussian kernel is set to 0.1, and linear dependence threshold to 0.1.

In controller implementations, we have used controller structure with an inner PD loop. Control action to the robot manipulator is a combination of an action generated by an adaptive learning RL signal through KRLS-SVM and a fixed gain PD controller signal. The PD loop will maintain stability until KRLS-SVM controller learns, starting with zero initialized Q-values. The controller, thus, requires no offline learning.

B. Simulation Results

In order to study the learning performance, and robustness against uncertainties, KRLS-SVM learning approach has been simulated on two-link robot manipulator control problem. MATLAB 7.4.0 (R2007a) has been used as simulation tool. To analyze the KRLS-SVM algorithm for computational cost, accuracy, and robustness, we compare the proposed approach with LS-SVM reinforcement learning approach.

C. Learning Performance Study

The physical system has been simulated for a single run of 10 sec using fourth-order Runge-Kutta method, with fixed time step of 10 msec over a single episode. The the output tracking error (both the links) and control torque (both the links) for LS-SVM and KRLS-SVM learning algorithms are shown in Figure 2 and Figure 3, respectively. Table I tabulates the mean square error, absolute maximum error (max $|e(t)|$), and absolute maximum control effort (max $|\tau|$) under nominal operating conditions.

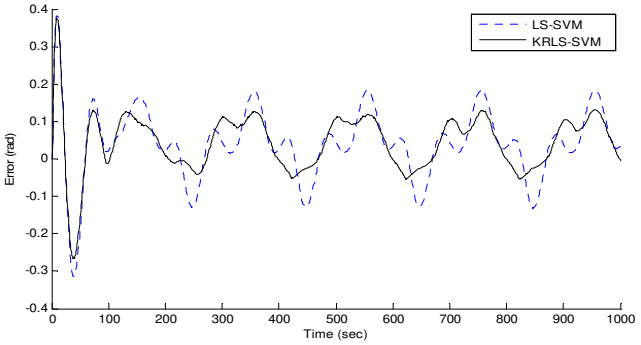

Fig. 2.(a) Output tracking error (link1)

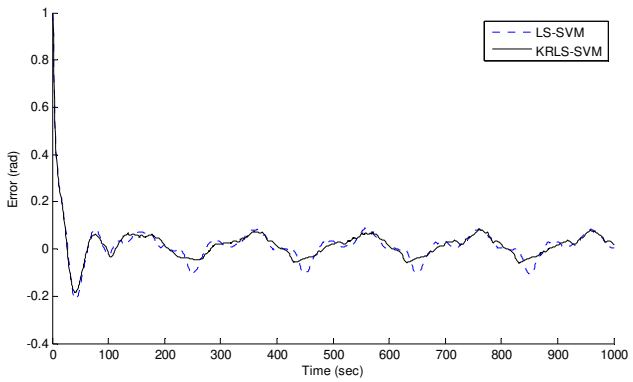

Fig. 2.(b) Output tracking error (link2)

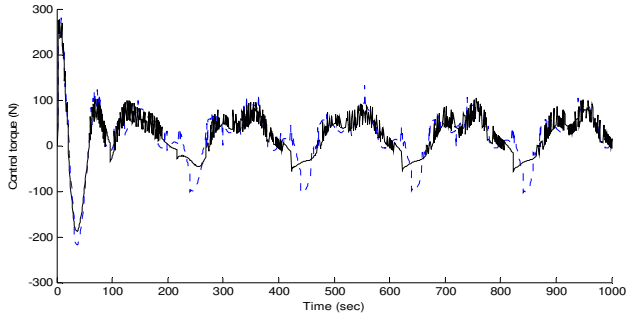

Fig. 3.(a) Control torque (link1)

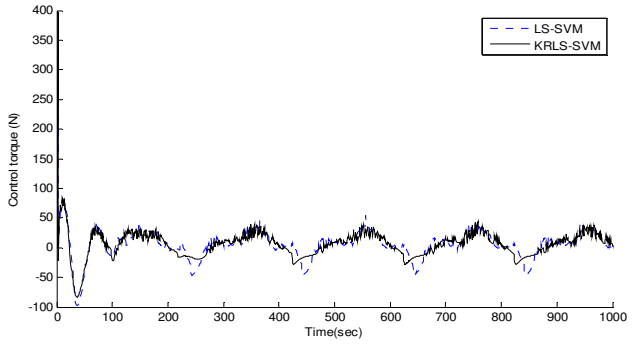

Fig. 3.(b) Control torque (link2)

TABLE I: COMPARISON OF CONTROLLERS

Controller	MSE (rad)		max $\lvert e(t)\rvert$ (rad)		max $\lvert \tau \rvert$ (Nm)		Training Time (sec)
	Link 1	Link 2	Link 1	Link 2	Link 1	Link 2	------
LS-SVM	0.0110	0.0071	0.1848	0.1042	133.28	54.75	32.607
KRLS-SVM	0.0080	0.0065	0.1322	0.0843	104.65	44.90	14.908

D. Robustness Study

In the following, we compare the performance of LS-SVM and KRLS-SVM controllers under uncertainties. For this study, we trained the controller for 20 episodes, and then evaluated the performance for two cases:

Effect of payload variations: The end-effector mass is varied with time, which corresponds to the robotic arm picking up and releasing payloads having different masses. The mass is varied as: a) $t < 2$ s ; $m_2 = 1$ kg b) $2 \le t < 3.5$ s ; $m_2 = 2.5$ kg c) $3.5 \le t < 4.5$ s ; $m_2 = 1$ kg d) $4.5 \le t < 6$ s ; $m_2 = 4$ kg e) $6 \le t < 7.5$ s ; $m_2 = 1$ kg f) $7.5 \le t < 9$ s ; $m_2 = 2$ kg g) $9 \le t < 10$ s ; $m_2 = 1$ kg . Figs. 4(a) and (b) show the output tracking errors (both the links) and Table II tabulates the mean square error, absolute maximum error (max $\lvert e(t)\rvert$), and absolute maximum control effort (max $\lvert \tau \rvert$) at payload variation with time.

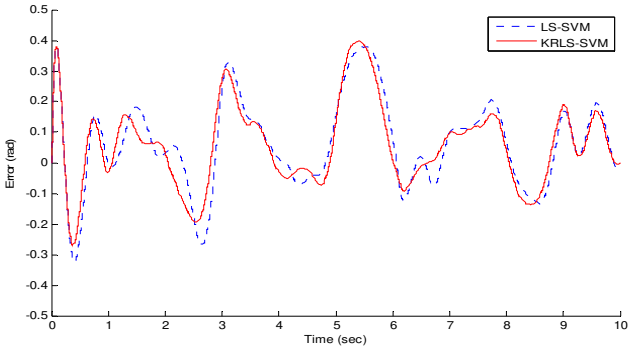

Fig. 4.(a) Output tracking error (link1)

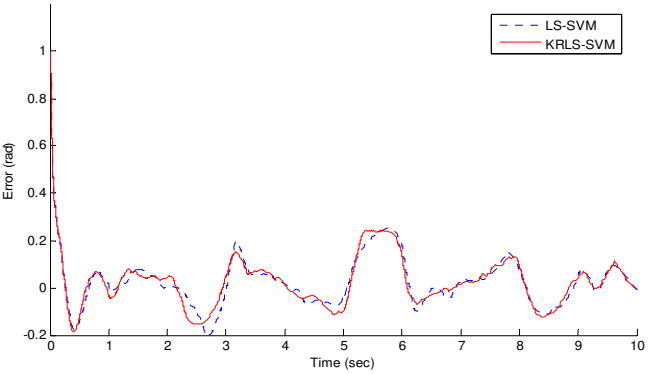

Fig. 4.(b) Output tracking error (link2)

TABLE II: COMPARISON OF CONTROLLERS

Controller	MSE (rad)		max $\lvert e(t)\rvert$ (rad)		max $\lvert \tau \rvert$ (Nm)	
	Link 1	Link 2	Link 1	Link 2	Link 1	Link 2
LS-SVM	0.0254	0.0121	0.3793	0.9070	281.87	402.01
KRLS-SVM	0.0220	0.0122	0.3984	0.9075	265.60	400.00

E. Effects of External Disturbances

A torque disturbance with a sinusoidal variation of frequency 2π rad/sec, was added to the model with time. The magnitude of torque disturbance is expressed as a percentage of control effort. The magnitude is varied as: a) $t < 2$ s ; 0% b) $2 \le t < 3.5$ s ; 0.2% c) $3.5 \le t < 4.5$ s ; 0% d) $4.5 \le t < 6$ s ; 0.8% e) $6 \le t < 7.5$ s ; 0% f) $7.5 \le t < 9$ s ; -0.2% g) $9 \le t < 10$ s ; 0%

Figs. 5(a) and (b) show the output tracking errors (both the links) and Table III tabulates the mean square error, absolute maximum error (max $\lvert e(t)\rvert$), and absolute maximum control effort (max $\lvert \tau \rvert$) for torque disturbances added to the model variation with time.

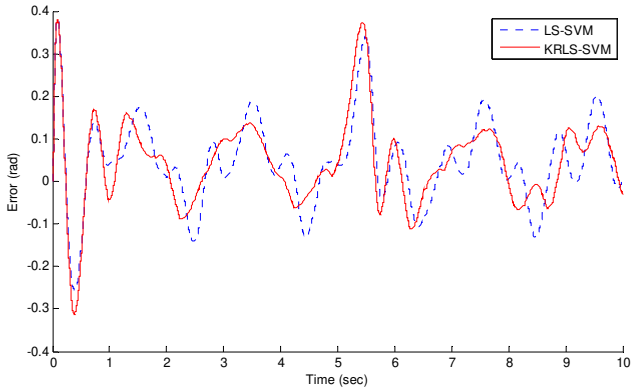

Fig. 5.(a) Output tracking error (link1)

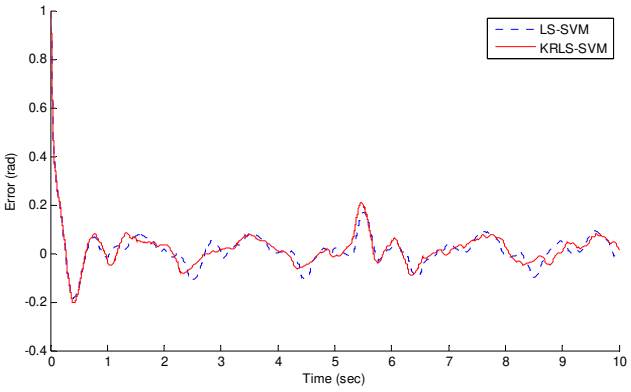

Fig. 5.(b) Output tracking error (link2)

TABLE III: COMPARISON OF CONTROLLERS

Controller	MSE (rad)		max $\|e(t)\|$ (rad)		max $\|\tau\|$ (Nm)	
	Link 1	Link 2	Link 1	Link 2	Link 1	Link 2
LS-SVM	0.0128	0.0064	0.3793	0.9080	285.05	397.99
KRLS-SVM	0.0125	0.0066	0.3805	0.9075	265.59	400.00

V. CONCLUSION

As an important machine learning method reinforcement learning has a difficulty scaling to challenges in large-scale space problems. A *Q*-learning method is proposed in this paper by taking advantage of good generalization ability with low computational cost of KRLS-SVM. An online sparsification process that permits the addition of training sample into the *Q*-function approximation only if it is approximately linearly independent of the preceding training samples.

Simulation results of two-link robot manipulator show that the proposed *Q*-learning method is suitable for working in large-scale continuous state-space; in fact the proposed scheme gives better accuracy, lower computational cost and better robustness property compare to the scheme based on LS-SVM.

REFERENCES

[1] V. Vapnik, The nature of statistical learning theory, Springer Verlag, 1995.

[2] J. A. K. Suykens and J. Vandewalle, "Least square support vector machine classifiers", *Neural processing letters*, vol. 9, 1999, pp. 293-300.

[3] X. Wang, X. Tian, and Y. Cheng, "Value approximation with least square support vector machines in reinforcement learning system," *Journal of Computational and Theoretical Nanoscience*, vol. 4, no. 7-8, 2007, pp. 1290-1294.

[4] Yaakov Engel, Shie Mannor, , and Ron Meir, "The Kernel Recursive Least-Squares Algorithm," *IEEE Transations on Signal Processing*, Vol. 52, No. 8, 2004.

[5] C. H. Watkins, "Learning from delayed rewards," *Thesis*, University of Cambridge, England, 1989.

[6] Singh S., Jaakkola T., Littman M., Szpesvari C. (2000) Convergence results for single step on-policy reinforcement learning algorithms, *Machine learning*, Vol. 38, pp.287-308.

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

[8] P. Y. Glorennec and L. Jouffe, "Fuzzy *Q*-learning," in *Proc. 6th IEEE Conf. Fuzzy Systems*, Barcelona, Spain, vol. 2, 1997, pp.659–662.

[9] R. H. Myers, Classical and Modern regression with Applications, 2nd Edn., Boston, 1994.

[10] Green A., Sasiadek J. Z., Dynamics and trajectory tracking control of a two-link robot manipulator, *J. Vibration Control* 2004; 10(10), 1415–1440.