Least Absolute Policy Iteration for Robust Value Function Approximation

Masashi Sugiyama, Hirotaka Hachiya, Hisashi Kashima, and Tetsuro Morimura

Abstract— **Least-squares policy iteration is a useful reinforcement learning method in robotics due to its computational efficiency. However, it tends to be sensitive to outliers in observed rewards. In this paper, we propose an alternative method that employs the absolute loss for enhancing robustness and reliability. The proposed method is formulated as a linear programming problem which can be solved efficiently by standard optimization software, so the computational advantage is not sacrificed for gaining robustness and reliability. We demonstrate the usefulness of the proposed approach through simulated robot-control tasks.**

I. Introduction

One of the popular reinforcement learning frameworks for obtaining the optimal policy is *policy iteration*, which iteratively performs policy evaluation and improvement steps [1], [2]. The computational cost of a naive implementation of policy iteration is dominated by the number of states and actions, so it is not scalable to real-world robotics problems with large state/action spaces. To cope with this problem, an alternative method called *least squares policy iteration (LSPI)* has been proposed [3]. In LSPI, value functions of policies are approximated using a linear architecture, so its computational cost is governed by the number of parameters in the linear model. Thus, if the number of parameters is kept reasonably small, LSPI is applicable to large-scale robot-control tasks.

A basic idea of LSPI is to learn the parameters of the linear model so that the *temporal-difference (TD)* error is minimized under the squared loss. On the other hand, in this paper, we propose minimizing the TD error under the absolute loss (see Fig.1). This is just replacement of the loss function, but we argue that this modification brings about very useful advantages in practical robotics problems. More specifically, the rationale behind the use of the absolute loss lies in *robustness* and *reliability*:

Robustness: In many robotics applications, immediate rewards are obtained through physical measurement such as distance sensors or computer vision. Due to intrinsic measurement noise or recognition error, the obtained rewards are often deviated from the true value; in particular, the rewards occasionally contain *outliers*, which are significantly different from regular values.

A demo movie is available from '`http://sugiyama-www.cs.titech.ac.jp/~sugi/2009/LAPIvsLSPI.mp4`'.

MS and HH are with Department of Computer Science, Tokyo Institute of Technology, Japan, and HK and TM are with IBM Research, Tokyo Research Laboratory, Japan. `sugi@cs.titech.ac.jp` `hachiya@sg.cs.titech.ac.jp` `hkashima@jp.ibm.com` `tetsuro@jp.ibm.com`

Residual minimization under the squared loss amounts to obtaining the *mean*:

$$\underset{c}{\mathrm{argmin}}\left[\sum_{i=1}^{m}(x_i-c)^2\right]=\mathrm{mean}(\{x_i\}_{i=1}^{m})=\frac{1}{m}\sum_{i=1}^{m}x_i.$$

If one of the values, say x_j, is very large, the mean would be strongly affected by this. Thus *all* the values $\{x_i\}_{i=1}^{m}$ are responsible for the mean and therefore even a *single* outlier observation can significantly damage the learned result.

On the other hand, residual minimization under the absolute loss amounts to obtaining the *median*.

$$\underset{c}{\mathrm{argmin}}\left[\sum_{i=1}^{2n+1}|x_i-c|\right]=\mathrm{median}(\{x_i\}_{i=1}^{2n+1})=x_n,$$

where $x_1\leq x_2\leq\cdots\leq x_{2n+1}$. The median is influenced not by the *magnitude* of the values $\{x_i\}_{i=1}^{2n+1}$, but only by their *order*. Thus, as long as the order is kept the same, the median is not affected by outliers—in fact, the median is known to be the most robust estimator in the light of *breakdown-point analysis* [4], [5].

Therefore, the use of the absolute loss would remedy the problem of robustness in policy iteration.

Reliability: In practical robot-control tasks, we often want to attain a stable performance, rather than to achieve a "dream" performance with very low chance; for example, in the acquisition of a humanoid gait, we may want the robot to walk forward in a stable manner with high probability of success, rather than to rush very fast in a chance level.

On the other hand, we do not want to be too conservative when training robots—if we are afraid of unrealistic failure too much, no practically useful control policies can be obtained. For example, any robots can be broken in principle if activated for long time. However, if we fear this fact too much, we may end up in a control policy that does not activate the robots at all—obviously this is non-sense in practice.

Since the squared-loss solution is not robust against outliers, it is sensitive to rare events with either positively or negatively very large immediate rewards. Consequently, the squared loss prefers an extraordinarily successful motion even if the success probability is very low; similarly it dislikes an unrealistic trouble even if such a terrible event may not happen in practice. On the other hand, the absolute-loss solution is not easily affected by such rare events due to robustness. Therefore, the use of the absolute loss would

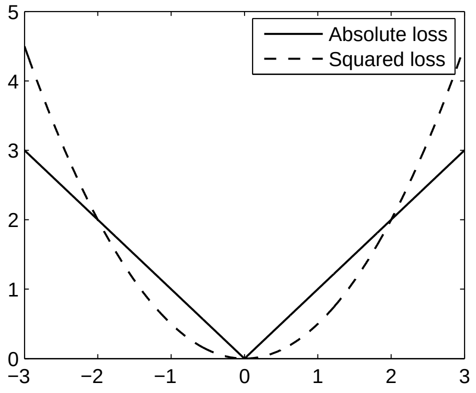


Fig. 1. The absolute and squared loss functions for reducing the temporal-difference error.

produce a reliable control policy even in the presence of such extreme events.

As shown above, the use of the absolute loss in value function approximation would bring about robustness and reliability, which are preferable properties in real-world robotics problems. This modification is very simple, but to the best of our knowledge, such an idea has never been incorporated in value function approximation.

Another important advantage of the proposed approach is that scalability to massive data is not sacrificed for enhancing robustness and reliability. Indeed, the absolute-loss solution can be obtained by solving a *linear programming* problem; this can be carried out very efficiently using a standard optimization software. We demonstrate the usefulness of the absolute-loss approach through robotics simulations.

Related Work: In the seminal paper [6], the α*-value criterion* was introduced as an alternative to the expected discounted reward. This criterion is essentially identical to the *value-at-risk* of the discounted reward, which is a popular risk measure in finance [7]. However, the resulted optimization problem is not convex and therefore it is difficult to obtain a good solution efficiently. A *soft risk aversion method* [8], emphasizes actions whose rewards are less than expected. Although the idea of this approach is intuitive, it is rather heuristic and does not have a clear interpretation as risk minimization. In the paper [9], an approach that optimizes a linear combination of the mean and the variance of discounted rewards was proposed. This approach is based on the *mean-variance model*, which is also a popular modeling in finance [10]. The assumption behind the mean-variance model is that the discounted rewards follow the Gaussian distribution, which may not be true in practice. On the other hand, the method proposed in this paper has a clear interpretation as *median* risk minimization and no strong assumption is imposed. Furthermore, the resulting optimization problem is a linear program, which is convex and can be solved efficiently using a standard optimization software.

In the area of optimal control, *robust control theory* was used to design stable controllers [11], [12], [13]. Although this approach is also sometimes referred to as *robust* reinforcement learning, its aim is different from the current paper—robust control aims at enhancing robustness against *uncertainties in the environment*, while our goal is to enhance robustness against *outliers*.

II. Problem Formulation

In this section, we formulate the reinforcement learning problem using a Markov decision process (MDP) and briefly review the core ideas of policy iteration and value function approximation.

A. Markov Decision Process

Let us consider an MDP specified by

$$(\mathcal{S}, \mathcal{A}, P_{\mathrm{T}}, R, \gamma),$$

where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $P_{\mathrm{T}}(s'|s,a)$ ($\in [0,1]$) is the conditional transition probability-density from state s to next state s' when action a is taken, $R(s,a,s')$ ($\in \mathbb{R}$) is a reward for transition from s to s' by taking action a, and $\gamma \in (0,1]$ is the discount factor for future rewards.

Let $\pi(a|s) \in [0,1]$ be a stochastic policy which is the conditional probability density of taking action a given state s. The state-action value function $Q(s,a) \in \mathbb{R}$ for policy π is the expected discounted sum of rewards the agent will receive when taking action a in state s and following policy π thereafter, i.e.,

$$Q(s,a) \equiv \mathop{\mathbb{E}}_{\pi, P_{\mathrm{T}}} \left[\sum_{n=1}^{\infty} \gamma^{n-1} R(s_n, a_n, s_{n+1}) \;\Big|\; s_1 = s, a_1 = a \right],$$

where $\mathbb{E}_{\pi,P_{\mathrm{T}}}$ denotes the expectation over $\{s_n, a_n\}_{n=1}^{\infty}$ following $\pi(a_n|s_n)$ and $P_{\mathrm{T}}(s_{n+1}|s_n,a_n)$. The goal of reinforcement learning is to obtain the policy that maximizes the discounted sum of future rewards.

Computing the value function $Q(s,a)$ is called *policy evaluation* since this corresponds to evaluating the value of policy π. Using $Q(s,a)$, we can find a better policy as

$$\pi(a|s) \leftarrow \delta(a - \operatorname*{argmax}_{a'} Q(s,a')),$$

where $\delta(\cdot)$ is the delta function. This is called *policy improvement*. It is known that repeating policy evaluation and policy improvement leads to the optimal policy [1]. This entire process is called *policy iteration*.

B. Value Function Approximation

Although policy iteration is guaranteed to produce the optimal policy, it is computationally intractable when the number of state-action pairs $|\mathcal{S}| \times |\mathcal{A}|$ is very large; $|\mathcal{S}|$ or $|\mathcal{A}|$ becomes infinity when the state space or action space is continuous. To overcome this problem, the state-action value function $Q(s,a)$ may be approximated using the following linear model:

$$\widehat{Q}(s,a) \equiv \sum_{b=1}^{B} \theta_b \phi_b(s,a) = \boldsymbol{\theta}^{\top} \boldsymbol{\phi}(s,a),$$

where

$$\boldsymbol{\phi}(s,a) = (\phi_1(s,a), \phi_2(s,a), \ldots, \phi_B(s,a))^\top$$

are the fixed basis functions, $\top$ denotes the transpose, B is the number of basis functions, and

$$\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_B)^\top$$

are model parameters. Note that B is usually chosen to be much smaller than $|\mathcal{S}| \times |\mathcal{A}|$.

Suppose we have an N-step data sample, i.e., the agent initially starts from a randomly selected state s_1 following the initial-state probability density $P_\mathrm{I}(s_1)$ and chooses an action based on the current policy $\pi(a_n|s_n)$. Then the agent makes a transition following $P_\mathrm{T}(s_{n+1}|s_n, a_n)$ and receives an immediate reward r_n $(= R(s_n, a_n, s_{n+1}))$—thus the training dataset $\mathcal{D}$ is expressed as

$$\mathcal{D} \equiv \{(s_n, a_n, r_n, s_{n+1})\}_{n=1}^N.$$

The *temporal-difference (TD)* error for the n-th sample is defined by

$$\boldsymbol{\theta}^\top \widehat{\boldsymbol{\psi}}(s_n, a_n) - r_n, \tag{1}$$

where $\widehat{\boldsymbol{\psi}}(s,a)$ is a B-dimensional column vector defined by

$$\widehat{\boldsymbol{\psi}}(s,a) \equiv \boldsymbol{\phi}(s,a) - \frac{\gamma}{|\mathcal{D}_{(s,a)}|} \sum_{s' \in \mathcal{D}_{(s,a)}} \mathop{\mathbb{E}}_{\pi(a'|s')} [\boldsymbol{\phi}(s', a')] .$$

$\mathcal{D}_{(s,a)}$ is a set of 4-tuple elements containing state s and action a in the training data $\mathcal{D}$, $\sum_{s' \in \mathcal{D}_{(s,a)}}$ denotes the summation over s' in the set $\mathcal{D}_{(s,a)}$, and $\mathbb{E}_{\pi(a'|s')}$ denotes the conditional expectation with respect to a' over $\pi(a'|s')$ given s'.

The issue we would like to address in this paper is the choice of the loss function when evaluating the TD error (1); more specifically, we argue that the use of the absolute loss is more advantageous than the squared loss. We note that our results could be easily extended to various settings—for example, multiple sequences of episodic training samples can be employed without essentially changing the framework. The *off-policy* scenarios where the sampling policy is different from the evaluation policy can also be incorporated by applying *importance-weighting* techniques [1], [14], [15]. However, we do not go into the detail of such generalization for keeping the presentation of the current paper simple.

III. Loss Functions for TD-error Minimization

In this section, we first review a squared-loss method for TD-error minimization and then introduce an absolute-loss method.

A. Least Squares Policy Iteration (LSPI)

A standard choice of the loss function for minimizing the residual error would be the squared loss [1], [16], [3]. The least-squares TD-error solution $\widehat{\boldsymbol{\theta}}$ is defined by

$$\widehat{\boldsymbol{\theta}}_\mathrm{LS} \equiv \mathop{\mathrm{argmin}}_{\boldsymbol{\theta}} \left[\frac{1}{2} \sum_{n=1}^N \left(\boldsymbol{\theta}^\top \widehat{\boldsymbol{\psi}}(s_n, a_n) - r_n \right)^2 \right].$$

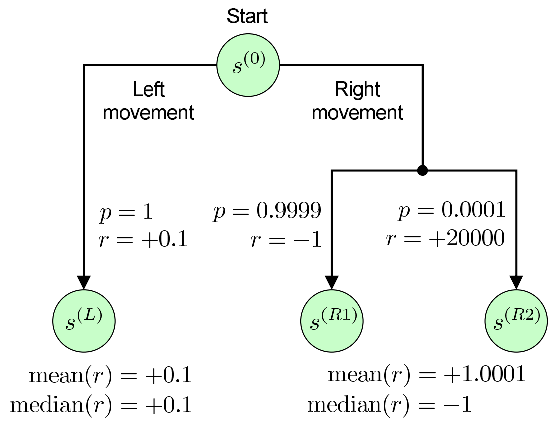


Fig. 2. Illustrative MDP problem.

The solution $\widehat{\boldsymbol{\theta}}_\mathrm{LS}$ can be analytically computed as

$$\widehat{\boldsymbol{\theta}}_\mathrm{LS} = \left(\sum_{n=1}^N \widehat{\boldsymbol{\psi}}(s_n, a_n) \widehat{\boldsymbol{\psi}}(s_n, a_n)^\top \right)^{-1} \sum_{n=1}^N r_n \widehat{\boldsymbol{\psi}}(s_n, a_n).$$

The value-function approximation method based on the above least-squares formulation is called *least squares TDQ (LSTDQ)* and the policy iteration method based on LSTDQ is called *least squares policy iteration (LSPI)* [3].

B. Least Absolute Policy Iteration (LAPI)

As explained in the introduction, LSPI suffers from excessive sensitivity to outliers and less reliability. Here, we introduce an alternative approach to value function approximation, which we refer to as *least absolute TDQ (LATDQ)*—we propose employing the absolute loss instead of the squared loss (Fig. 1):

$$\widehat{\boldsymbol{\theta}}_\mathrm{LA} \equiv \mathop{\mathrm{argmin}}_{\boldsymbol{\theta}} \left[\sum_{n=1}^N \left| \boldsymbol{\theta}^\top \widehat{\boldsymbol{\psi}}(s_n, a_n) - r_n \right| \right]. \tag{2}$$

This minimization problem looks cumbersome due to the absolute value operator which is non-differentiable, but the following mathematical trick mitigates this issue.

Proposition 1: [17]

$$|x| = \min_b b \quad \text{subject to } -b \leq x \leq b. \tag{3}$$

Since Eq.(3) is a linear programming problem, it can be solved very efficiently using a standard optimization software. Using this proposition, the minimization problem (2) is reduced to the following linear programming problem:

$$\begin{cases} \displaystyle\min_{\boldsymbol{\theta}, \{b_n\}_{n=1}^N} & \displaystyle\sum_{n=1}^N b_n \\ \text{subject to} & -b_n \leq \boldsymbol{\theta}^\top \widehat{\boldsymbol{\psi}}(s_n, a_n) - r_n \leq b_n, \ \forall n. \end{cases}$$

The number of constraints is N in the above linear program. When N is large, we may employ sophisticated optimization techniques such as *column generation* [18] for efficiently solving the linear programming problem.

We refer to the policy iteration method based on LATDQ as *least absolute policy iteration (LAPI)*.

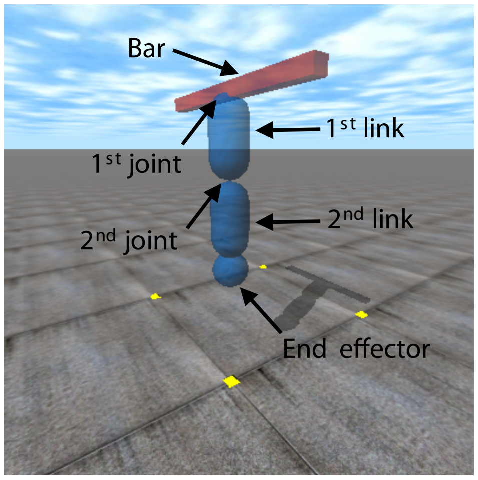


Fig. 3. Illustration of the acrobot. The goal is to swing up the end effector by only controlling the second joint.

C. Numerical Examples of LATDQ

For illustration purposes, let us consider the 4-state MDP problem described in Fig. 2. The agent is initially located at state $s^{(0)}$ and the actions allowed to take is moving to the left or right states. If the left-movement action is chosen, the agent always receives small positive reward $+0.1$ at $s^{(L)}$. On the other hand, if the right-movement action is chosen, the agent receives negative reward -1 with probability 0.9999 at $s^{(R1)}$ or it receives very large positive reward $+20000$ with probability 0.0001 at $s^{(R2)}$. The mean and median rewards for left movement are both $+0.1$, while the mean and median rewards for right movement are $+1.0001$ and -1, respectively.

If $Q(s^{(0)}, \text{'Left'})$ and $Q(s^{(0)}, \text{'Right'})$ are approximated by LSTDQ, it returns the mean rewards, i.e., $+0.1$ and $+1.0001$, respectively. Thus LSTDQ prefers right movement, which is a 'gambling' policy that negative reward -1 is almost always obtained at $s^{(R1)}$, but there is a chance to obtain very high reward $+20000$ with a very small probability at $s^{(R2)}$. On the other hand, if $Q(s^{(0)}, \text{'Left'})$ and $Q(s^{(0)}, \text{'Right'})$ are approximated by LATDQ, it returns the median rewards, i.e., $+0.1$ and -1, respectively. Thus LATDQ prefers left movement, which is a stable policy that the agent can always receive small positive reward $+0.1$ at $s^{(L)}$.

If all the rewards in Fig. 2 are negated, the value functions are also negated and we obtain a different interpretation: LSTDQ is afraid of the risk of receiving very large negative reward -20000 at $s^{(R2)}$ with a very low probability, and consequently it ends up in a very conservative policy that the agent always receives negative reward -0.1 at $s^{(L)}$. On the other hand, LATDQ tries to receive positive reward $+1$ at $s^{(R1)}$ without being afraid of visiting $s^{(R2)}$ too much.

As illustrated above, LATDQ tends to provide qualitatively different solutions from LSTDQ. We argue that the robust and reliable behavior of LATDQ would be more preferable in practical robotics tasks, as discussed in the introduction. In the next section, we experimentally show the usefulness of the proposed method in robot-control tasks.

IV. Experimental Evaluation

In this section, we apply LAPI to simulated robot-control problems and evaluate its practical performance.

A. Acrobot Swing-up

Here, we use an *acrobot* illustrated in Fig. 3. The acrobot is an under-actuated system and consists of two links, two joints, and an end effector. The length of each link is 0.3 $[m]$, and the diameter of each joint is 0.15 $[m]$. The diameter of the end effector is 0.10 $[m]$ and the height of the horizontal bar is 1.2 $[m]$. The first joint connects the first link to the horizontal bar and is *not* controllable. The second joint connects the first link to the second link and is controllable. The end effector is attached to the tip of the second link. The control command (action) we can choose is applying positive torque $+30$ $[N \cdot m]$, no torque 0 $[N \cdot m]$, or negative torque -30 $[N \cdot m]$ to the second joint. Note that the acrobot moves only within a plane orthogonal to the horizontal bar.

The goal is to acquire a control policy such that the end effector is swung up as high as possible. The state space consists of the angle θ_i $[rad]$ and angular velocity $\dot{\theta}_i$ $[rad/s]$ of the first and second joints ($i = 1, 2$). The immediate reward is given according to the height h of the center of the end effector—if $h > 1.3$, h itself is given as the reward; otherwise -0.01 is given as the reward. Note that $0.55 \leq h \leq 1.85$ in the current setting.

Here, we suppose that the length of the links is unknown; thus the height h cannot be directly computed from state information. The height of the end effector is supposed to be estimated from an image taken by a camera—the end effector is detected in the image and then its vertical coordinate is computed. Due to recognition error, the estimated height is highly noisy and could contain outliers.

In each policy iteration step, 10 episodic training samples of length 200 are gathered, and the performance of the obtained policy is evaluated using 100 episodic test samples of length 200. The policies are updated in a *soft-max* manner:

$$\pi(a|s) \leftarrow \frac{\exp(Q(s,a)/\eta)}{\sum_{a' \in \mathcal{A}} \exp(Q(s,a')/\eta)}, \tag{4}$$

where $\eta = 10 - t + 1$ with t being the iteration number. The discounted factor is set to $\gamma = 1$, i.e., no discount. As basis functions for value function approximation, we use the Gaussian kernel with standard deviation π—the Gaussian centers are located on all combinations of

$$\theta_1, \theta_2 \in \{-\pi, -\tfrac{\pi}{2}, 0, \tfrac{\pi}{2}, \pi\},$$
$$(\dot{\theta}_1, \dot{\theta}_2) \in \{(-\pi, -\pi), (0, 0), (\pi, \pi)\}.$$

The above 75 $(= 5 \times 5 \times 3)$ Gaussian kernels are defined for each of the three actions; thus 225 $(= 75 \times 3)$ kernels are used in total. In our implementation, the matrix inverse included in LSPI is computed by MATLAB®, and the linear programming problem included in LAPI is solved by CPLEX®.

We consider two noise environments; one is no noise and the other is Laplacian noise with variance 1. Note that the tail

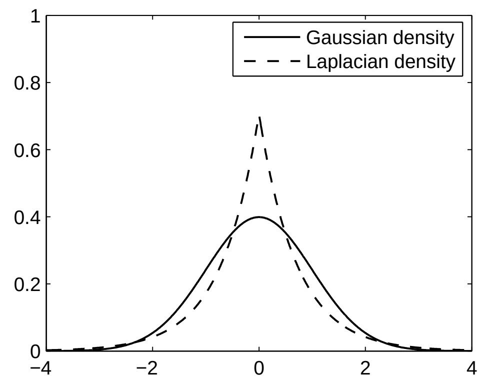
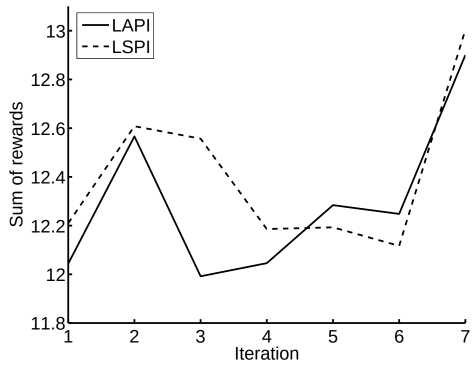
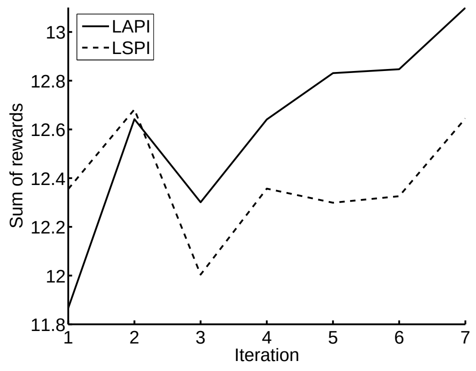


Fig. 4. Probability density functions of Gaussian and Laplacian distributions.



(a) No noise



(b) Laplacian noise

Fig. 5. The sum of future rewards averaged over 20 runs for the acrobot swinging-up simulation.

of the Laplacian density is heavier than that of the Gaussian density (see Fig. 4), implying that a small number of outliers tend to be included in the Laplacian noise environment. For each noise environment, the experiment is repeated 20 times with different random seed and the average sum of future rewards obtained by LAPI and LSPI are summarized in Fig. 5. In the noise-less case (see Fig. 5(a)), both LAPI and LSPI improve the performance over iterations in a comparable way. On the other hand, in the noisy case (see Fig. 5(b)), the performance of LSPI is not improved much due to outliers, while LAPI still produces a good control policy.

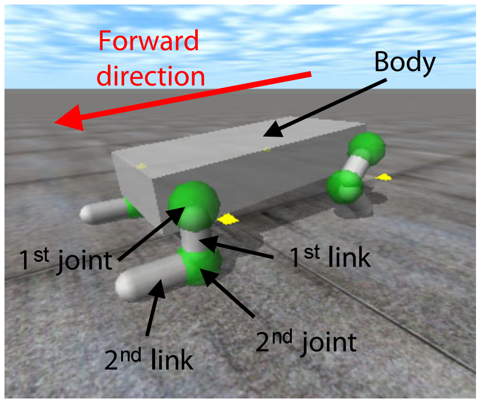


Fig. 6. Illustration of the four-legged robot. The goal is to let the robot walk forward by controlling the leg joints.

A demo movie[1] of the robot motion obtained by LAPI and LSPI under the noisy environment shows the superior performance of LAPI.

B. *Four-legged Robot Walking*

Next, we apply LAPI and LSPI to a four-legged robot walking task (see Fig. 6).

The robot consists of a body and four legs, and each leg is composed of two links and two joints. The first joint connects the first link to the body, and the second joint connects the first link to the second link. The control command (action) we can choose is applying positive torque $+100$ $[N \cdot m]$ or negative torque -100 $[N \cdot m]$ to the first joints and positive torque $+70$ $[N \cdot m]$ or negative torque -70 $[N \cdot m]$ to the second joints; we can apply different torques to the front and hind joints, but the same torque to the left and right joints. Thus, the number of possible actions is 16 $(= 2 \times 2 \times 2 \times 2)$.

The goal is to acquire a control policy such that the robot walks forward. The state space consists of the angle θ_i $[rad]$ of the i-th joint of the front legs $(i = 1, 2)$, the angle η_i $[rad]$ of the i-th joint of the hind legs $(i = 1, 2)$, the height h of the body, and the forward-backward angle ϕ of the body. Note that the behavior of the left and right legs are identical in the current setup, so the dimension of the state space is 6. We impose the following restriction on the joint angles:

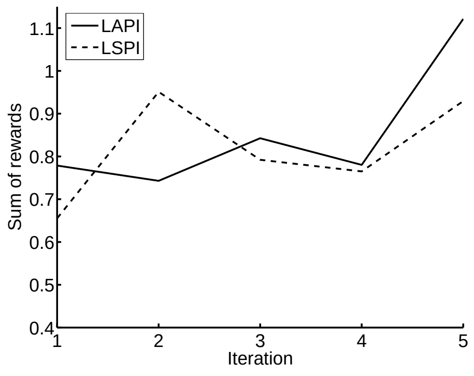
$$-\frac{\pi}{4} \leq \theta_1, \theta_2 \leq \frac{\pi}{4}, \;\; 0 \leq \eta_1 \leq \frac{\pi}{4}, \;\; \text{and} \;\; -\frac{\pi}{4} \leq \eta_2 \leq 0.$$

Thus, the second joints of the front legs are bent backward and the second joints of the hind legs are bent forward (see Fig. 6 again).

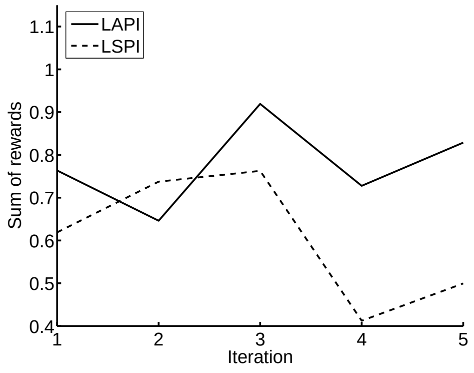
The immediate reward given in each time step is d, where d is the walking distance. Since the walking distance is not easily computable from state information, we suppose it is measured by a distance sensor. Thus the reward is highly noisy and could contain outliers.

In each policy iteration step, 20 episodic training samples of length 50 are gathered, and the performance of the obtained policy is evaluated using 100 episodic test samples of length 50. The policies are updated by *soft-max* (4) with

[1]'`http://sugiyama-www.cs.titech.ac.jp/~sugi/2009/LAPIvsLSPI.mp4`'.



(a) No noise



(b) Laplacian noise

Fig. 7. The sum of future rewards averaged over 20 runs for the four-legged robot walking simulation.

$\eta = 10 - t + 1$, where t is the iteration number. The discounted factor is set to $\gamma = 1$, i.e., no discount. As basis functions for value function approximation, we use the Gaussian kernel with standard deviation 1; the Gaussian centers are located on all combinations of

$$\theta_1, \eta_1 \in \{-\tfrac{\pi}{4}, 0, \tfrac{\pi}{4}\}, \ \ \theta_2 \in \{0, \tfrac{\pi}{4}\},$$
$$\eta_2 \in \{-\tfrac{\pi}{4}, 0\}, \ \ h \in \{1\}, \text{ and } \phi \in \{0\}.$$

The above 36 $(= 3 \times 3 \times 2 \times 2 \times 1 \times 1)$ Gaussian kernels are defined for each of the 16 actions and therefore 5761 $(= 36 \times 16)$ kernels are used in total. We again use MATLAB® and CPLEX® for computing the solution of LSPI and LAPI.

We consider two noise environments; one is no noise and the other is Laplacian noise with standard deviation 0.5. For each noise environment, the experiment is repeated 20 times with different random seed and the average sum of future rewards obtained by LAPI and LSPI are summarized in Fig. 7. This shows that LAPI and LSPI are comparable in the noise-less case, and LAPI tends to outperform LSPI in the noisy case.

A demo movie[2] of the robot motion obtained by LAPI and LSPI under the noisy environment illustrates the usefulness of LAPI.

[2]'`http://sugiyama-www.cs.titech.ac.jp/~sugi/2009/LAPIvsLSPI.mp4`'.

V. Conclusions

In this paper, we proposed using the absolute loss in value function approximation for enhancing robustness and reliability. The change of loss function resulted in a linear programming formulation which can be solved efficiently by a standard optimization software. We experimentally investigated the usefulness of the proposed method, LAPI, in simulated robot-control tasks and confirmed advantages of LAPI; the good performance of the existing method, LSPI, is maintained in the noise-less cases and higher tolerance to outliers than LSPI is exhibited in the noisy cases. Furthermore, the computation time of LAPI and LSPI is comparable—meaning that the computational advantage of LSPI is not sacrificed for improving robustness and reliability.

References

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

[2] B. P. D. and J. Tsitsiklis, *Neuro-Dynamic Programming*. NH, USA: Athena Scientific, 1996.

[3] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of Machine Learning Research*, vol. 4, no. Dec, pp. 1107–1149, 2003.

[4] P. J. Huber, *Robust Statistics*. New York: Wiley, 1981.

[5] P. J. Rousseeuw and A. M. Leroy, *Robust Regression and Outlier Detection*. New York: Wiley, 1987.

[6] M. Herger, "Considering of risk in reinforcement learning," in *Proceedings of the 11th International Conference on Machine Learning*, 1994, pp. 105–111.

[7] R. T. Rockafellar and S. Uryasev, "Conditional value-at-risk for general loss distributions," *Journal of Banking & Finance*, vol. 26, no. 7, pp. 1443–1472, 2002.

[8] O. Mihatsch and R. Neuneier, "Risk sensitive reinforcement learning," *Machine Learning*, vol. 49, no. 2-3, pp. 267–290, 2002.

[9] M. Sato, H. Kimura, and S. Kobayashi, "TD algorithm for the variance of return and mean-variance reinforcement learning," *Journal of Japanese Society of Artificial Intelligence*, vol. 16, no. 3, pp. 353–362, 2001.

[10] H. M. Markovitz, "Portfolio selection," *Journal of Finance*, vol. 7, no. 1, pp. 77–91, 1952.

[11] R. M. Kretchmar, P. M. Young, C. W. Anderson, D. C. Hittle, M. L. Anderson, and C. C. Delnero, "Robust reinforcement learning control with static and dynamic stability," *International Journal of Robust and Nonlinear Control*, vol. 11, no. 15, pp. 1469–1500, 2001.

[12] J. Morimoto and K. Doya, "Robust reinforcement learning," *Neural Computation*, vol. 17, no. 2, pp. 335–359, 2005.

[13] C. W. Anderson, P. M. Young, J. N. Buehner, M. R. Knight, H. A. Bush, and D. C. Hittle, "Robust reinforcement learning control using integral quadratic constraints for recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 18, no. 4, pp. 993–1002, 2007.

[14] D. Precup, R. S. Sutton, and S. Singh, "Eligibility traces for off-policy policy evaluation," in *Proceedings of the Seventeenth International Conference on Machine Learning*, Morgan Kaufmann, 2000, pp. 759–766.

[15] H. Hachiya, T. Akiyama, M. Sugiyama, and J. Peters, "Adaptive importance sampling for value function approximation in off-policy reinforcement learning," *Neural Networks*, 2009, to appear.

[16] D. Precup, R. S. Sutton, and S. Dasgupta, "Off-policy temporal-difference learning with function approximation," in *Proceedings of International Conference on Machine Learning*, 2001, pp. 417–424.

[17] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge: Cambridge University Press, 2004.

[18] A. Demiriz, K. P. Bennett, and J. Shawe-Taylor, "Linear programming boosting via column generation," *Machine Learning*, vol. 46, no. 1/3, p. 225, 2002.