Using prior knowledge to accelerate online least-squares policy iteration

Lucian Buşoniu, Bart De Schutter, Robert Babuška, Damien Ernst

Abstract— **Reinforcement learning (RL) is a promising paradigm for learning optimal control. Although RL is generally envisioned as working without any prior knowledge about the system, such knowledge is often available and can be exploited to great advantage. In this paper, we consider prior knowledge about the monotonicity of the control policy with respect to the system states, and we introduce an approach that exploits this type of prior knowledge to accelerate a state-of-the-art RL algorithm called online least-squares policy iteration (LSPI). Monotonic policies are appropriate for important classes of systems appearing in control applications. LSPI is a data-efficient RL algorithm that we previously extended to online learning, but that did not provide until now a way to use prior knowledge about the policy. In an empirical evaluation, online LSPI with prior knowledge learns much faster and more reliably than the original online LSPI.**

I. Introduction

Reinforcement learning (RL) can address important problems from a variety of fields, including automatic control, computer science, operations research, and economics [1]–[3]. In automatic control, RL algorithms can in principle solve nonlinear, stochastic optimal control problems without using a model. Rather than relying on the model, a RL controller learns how to control the system from data. The immediate performance is measured by a scalar reward, and the goal is to find an optimal control policy that maximizes the value function, i.e., the cumulative reward as a function of the system state and possibly of the control action. For systems with continuous or large discrete state-action spaces, RL solutions cannot be represented exactly, but must be approximated. State-of-the-art algorithms for approximate RL use weighted summations of basis functions to represent the value function, and least-squares techniques to find the weights [4]–[8].

One such algorithm is least-squares policy iteration (LSPI) [6]. At every iteration, LSPI evaluates the current policy, by computing its approximate value function from transition samples, and then finds a new, improved policy from this value function. LSPI can efficiently use transition data collected in any manner, but works offline. In [9], we have introduced an online variant of LSPI, which collects its own data by interacting with the system, and performs policy improvements "optimistically" [3], [10], without waiting until an accurate evaluation of the current policy is completed. Such policy improvements allow online LSPI to learn fast, i.e., to achieve good performance after interacting with the system for only a short interval of time.

This research was financially supported by the BSIK-ICIS project (grant no. BSIK03024). Lucian Buşoniu, Bart De Schutter, and Robert Babuška are with the Delft Center for Systems and Control, Delft University of Technology, the Netherlands (email: i.l.busoniu@tudelft.nl, b.deschutter@tudelft.nl, r.babuska@tudelft.nl). Damien Ernst is a Research Associate of the Belgian FNRS; he is affiliated with the Systems and Modeling Unit of the University of Liège, Belgium (email: dernst@ulg.ac.be).

In this paper, we present a method to further accelerate online LSPI by exploiting prior knowledge. Although RL is usually envisioned as working without any prior knowledge, exploiting such knowledge can be highly beneficial, if it is available. We consider prior knowledge in the form of the monotonicity of the control policy with respect to the state variables. Monotonic policies are suitable for controlling e.g., (nearly) linear systems, nonlinear systems in neighborhoods of equilibria where they are nearly linear, as well as some linear systems with monotonic input nonlinearities (such as saturation or dead-zone nonlinearities). We employ a policy representation for which monotonicity can be ensured by imposing linear inequality constraints on the policy parameters. This allows policy improvements to be performed efficiently, using quadratic programming. A speedup of the learning process is expected, because online LSPI restricts its focus to the class of monotonic policies, and no longer invests valuable learning time in trying other, unsuitable policies.

Several other online RL algorithms based on policy iteration and least-squares techniques have been proposed. For instance, [11] investigated a version of LSPI with online sample collection, focusing on the issue of exploration. This version does not perform optimistic policy updates, but fully executes offline LSPI between consecutive sample-collection episodes. An algorithm related to LSPI, called least-squares policy evaluation [5], was studied in the optimistic context in [12]. However, these existing techniques do not exploit prior knowledge about the solution.

The remainder of this paper is organized as follows. The necessary theoretical background on RL and (offline and online) LSPI is described in Section II. Then, in Section III, we introduce our procedure to integrate prior knowledge into online LSPI. Section IV provides an empirical evaluation of the resulting online LSPI with prior knowledge, for a simulated DC motor example. Section V concludes the paper.

II. The RL problem. Online LSPI

This section first introduces the RL problem in the framework of Markov decision processes, following [2], [3]. Then, offline LSPI [6] and online LSPI [9] are described.

Consider a Markov decision process with state space X and action space U. Assume for now that X and U are countable. The probability that the next state x_{k+1} is reached after action u_k is taken in state x_k is $f(x_k,u_k,x_{k+1})$, where $f : X \times U \times X \to [0,1]$ is the transition probability function. After the transition to x_{k+1}, a reward $r_{k+1} = \rho(x_k,u_k,x_{k+1})$

is received, where $\rho : X \times U \times X \to \mathbb{R}$ is the reward function. The symbol k denotes the discrete time index. The expected infinite-horizon discounted return of initial state x_0 under a control policy $h : X \to U$ is:

$$R^h(x_0) = \lim_{K\to\infty} \mathrm{E}_{x_{k+1}\sim f(x_k,h(x_k),\cdot)} \left\{ \sum_{k=0}^{K} \gamma^k r_{k+1} \right\} \qquad (1)$$

where $r_{k+1} = \rho(x_k, u_k, x_{k+1})$, $\gamma \in [0,1)$ is the discount factor, and the notation $x_{k+1} \sim f(x_k, h(x_k), \cdot)$ means that x_{k+1} is drawn from the distribution $f(x_k, h(x_k), \cdot)$. The goal is to find an optimal policy h^* that maximizes the return (1) from every $x_0 \in X$. RL algorithms aim to find h^* from transition and reward data, without using the functions f and ρ. Moreover, *online* RL algorithms collect their own data, by interacting with the system while they learn.

The classical policy iteration algorithm starts with some initial policy h_0. At every iteration $\ell \geq 0$, the algorithm first *evaluates* the current policy h_ℓ by computing its Q-function $Q_\ell : X \times U \to \mathbb{R}$, which gives for every pair (x,u) the expected return when starting in x, applying u, and following h_ℓ thereafter. This Q-function is the unique solution of the Bellman equation:

$$Q_\ell(x,u) = \mathrm{E}_{x'\sim f(x,u,\cdot)} \left\{ \rho(x,u,x') + \gamma Q_\ell(x', h(x')) \right\} \qquad (2)$$

Once Q_ℓ is available, an *improved* policy is determined:

$$h_{\ell+1}(x) = \arg\max_u Q_\ell(x,u) \qquad (3)$$

and the algorithm continues with this policy at the next iteration. Policy iteration is guaranteed to converge to h^*, see, e.g., Section 4.3 of [2].

When the state-action space is very large, Q-functions cannot be represented exactly, but must be approximated. Here, linearly parameterized approximators are considered:

$$\widehat{Q}(x,u) = \sum_{l=1}^{n} \phi_l(x,u)\theta_l = \phi^{\mathrm{T}}(x,u)\theta \qquad (4)$$

where $\phi(x,u)$ is a vector of n basis functions (BFs), $\phi(x,u) = [\phi_1(x,u), \ldots, \phi_n(x,u)]^{\mathrm{T}}$, and $\theta \in \mathbb{R}^n$ is a parameter vector. Given this approximator, the policy evaluation problem at the ℓth iteration, for the policy h_ℓ, boils down to finding θ_ℓ so that $\widehat{Q}_\ell \approx Q_\ell$, where $\widehat{Q}_\ell(x,u) = \phi^{\mathrm{T}}(x,u)\theta_\ell$.

LSPI [6] is an originally offline RL algorithm that finds θ_ℓ by solving the linear system:

$$\Gamma\theta_\ell = z \qquad (5)$$

where the matrix $\Gamma \in \mathbb{R}^{n\times n}$ and the vector $z \in \mathbb{R}^n$ are computed from samples. Specifically, consider a set of n_s samples $\{(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}, x'_{l_\mathrm{s}} \sim f(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}, \cdot), r_{l_\mathrm{s}} = \rho(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}, x'_{l_\mathrm{s}})) \,|\, l_\mathrm{s} = 1, \ldots, n_\mathrm{s}\}$. The matrix Γ and the vector z are initialized to zeros and updated for every sample $l_\mathrm{s} = 1, \ldots, n_\mathrm{s}$ as follows:

$$\begin{aligned} \Gamma &\leftarrow \Gamma + \phi(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})\phi^{\mathrm{T}}(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}) \\ &\quad - \gamma\phi(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})\phi^{\mathrm{T}}(x'_{l_\mathrm{s}}, h(x'_{l_\mathrm{s}})) \\ z &\leftarrow z + \phi(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}) r_{l_\mathrm{s}} \end{aligned} \qquad (6)$$

Once θ_ℓ is available, the approximate Q-function $\widehat{Q}_\ell$ is plugged into (3) to obtain an improved policy, and the algorithm continues with this policy at the next iteration.

Note that although for the derivation above we assumed that X and U are countable, LSPI can also be applied in uncountable spaces, such as the continuous spaces found in most automatic control problems. Also note that the algorithm is called "least-squares" because the system (5) is, in a certain sense, a least-squares approximation of the Bellman equation (2). For a more detailed description of LSPI, see [6].

In this paper, we consider an *online* variant of LSPI, which collects its own transition samples by interacting with the system [9]. This online variant is shown in Algorithm 1. Note that an idealized, infinite-time setting is considered, in which the algorithm runs forever and its result is the performance improvement achieved while interacting with the system. In practice, the algorithm is of course stopped after a finite time.

Algorithm 1 Online LSPI with ε-greedy exploration.

Input: BFs $\phi_1, \ldots, \phi_n$; γ; K_θ; $\{\varepsilon_k\}_{k\geq 0}$; δ

1: $\ell \leftarrow 0$; initialize policy h_0
2: $\Gamma \leftarrow \delta I_{n\times n}$; $z \leftarrow 0_n$
3: measure initial state x_0
4: **for** each time step $k \geq 0$ **do**
5: $\quad u_k \leftarrow \begin{cases} h_\ell(x_k) & \text{w.p. } 1-\varepsilon_k \\ \text{a uniform random action} & \text{w.p. } \varepsilon_k \end{cases}$
6: $\quad$ apply u_k, measure next state x_{k+1} and reward r_{k+1}
7: $\quad \Gamma \leftarrow \Gamma + \phi(x_k,u_k)\phi^{\mathrm{T}}(x_k,u_k) - \gamma\phi(x_k,u_k)\phi^{\mathrm{T}}(x_{k+1},h_\ell(x_{k+1}))$
8: $\quad z \leftarrow z + \phi(x_k,u_k)\, r_{k+1}$
9: $\quad$ **if** $k = (\ell+1)K_\theta$ **then**
10: $\quad\quad$ find θ_ℓ by solving $\Gamma\theta_\ell = z$
11: $\quad\quad h_{\ell+1}(x) \leftarrow \arg\max_u \phi^{\mathrm{T}}(x,u)\theta_\ell$
12: $\quad\quad \ell \leftarrow \ell + 1$
13: $\quad$ **end if**
14: **end for**

To ensure fast learning, online LSPI performs policy improvements once every few transitions, without waiting until an accurate evaluation of the current policy is completed (unlike the offline algorithm, which processes all the samples at every iteration, to obtain an accurate policy evaluation). Such a variant of policy iteration is called "optimistic" [3], [10]. The integer $K_\theta \geq 1$ is the number of transitions between two consecutive policy improvements.

To collect informative data, online LSPI must also *explore*, i.e., try other actions than those given by the current policy. In Algorithm 1, ε-greedy exploration is employed, which applies at every step k a uniformly random exploratory action with probability $\varepsilon_k \in [0,1]$, and the maximizing (also called greedy) action with probability $1-\varepsilon_k$, see, e.g., Section 2.2 of [2]. Typically, ε_k decreases over time, so that the algorithm increasingly *exploits* the current policy. Furthermore, to ensure the invertibility of Γ in the early stages of the learning process, this matrix is initialized to a small multiple of the identity matrix, using the parameter $\delta > 0$.

Note that, in practice, online LSPI does not have to compute and store *complete* improved policies (line 11). Indeed, such a procedure would be problematic in large

and continuous state spaces. Fortunately, improved actions can instead be found by applying the formula at line 11 on demand, only for the states where such actions are actually necessary.

III. Online LSPI with prior knowledge

RL is usually envisioned as working without any prior knowledge about the system or the solution. However, in practice, prior knowledge is often available, and using it can offer great benefits. We propose to exploit prior knowledge about the policy, since this is often easier to obtain than knowledge about the value function. Policy knowledge can generally be described by defining constraints. The main benefit of constraining policies is a speedup of the learning process, expected because the algorithm restricts its focus to the constrained class of policies, and no longer invests valuable learning time in trying other, unsuitable policies. We do not focus on accelerating *computation*, but rather on using *experience* more efficiently: an algorithm is fast if it performs well after a observing a small number of transitions. This measure of learning speed is crucial in practice, because obtaining data is costly (in terms of energy consumption, wear-and-tear, and possibly economic profit), whereas computation is relatively cheap.

We develop an online LSPI variant for *globally monotonic policies*. Such policies are monotonic with respect to any state variable, if the other state variables are held constant. Monotonic policies are suitable for controlling important classes of systems, including, e.g., (nearly) linear systems, or nonlinear systems in neighborhoods of equilibria where they are nearly linear. Monotonic policies also work well for some linear systems with monotonic input nonlinearities (such as saturation or dead-zone nonlinearities), for which the policy may be strongly nonlinear, but still monotonic.

A. Globally monotonic policies

Consider a system with a D-dimensional, continuous state space $X \subset \mathbb{R}^D$. We assume that X is a hyperbox:

$$X = [x_{\min,1}, x_{\max,1}] \times \cdots \times [x_{\min,D}, x_{\max,D}] \tag{7}$$

where $x_{\min,d} \in \mathbb{R}$, $x_{\max,d} \in \mathbb{R}$, and $x_{\min,d} < x_{\max,d}$, for $d = 1, \ldots, D$. For simplicity, we also assume that u is scalar, but the entire derivation in the sequel can easily be extended to multiple action variables.

A policy h is monotonic along the dth dimension of the state space if and only if, for any pair $(x, \bar{x}) \in X \times X$ of states that fulfill:

$$x_d \leq \bar{x}_d; \quad \text{and } x_{d'} = \bar{x}_{d'} \ \forall d' \neq d$$

the policy satisfies:

$$\delta_{\text{mon},d} \cdot h(x) \leq \delta_{\text{mon},d} \cdot h(\bar{x}) \tag{8}$$

where $\delta_{\text{mon},d} \in \{-1, 1\}$ specifies the monotonicity direction: if $\delta_{\text{mon},d}$ is -1 then h is decreasing along the dth dimension, and if it is 1 then h is increasing. A policy is (globally) monotonic if it is monotonic along every dimension d. The monotonicity directions are collected in a vector $\delta_{\text{mon}} = [\delta_{\text{mon},1}, \ldots, \delta_{\text{mon},D}]^{\mathrm{T}} \in \{-1, 1\}^D$, which encodes the prior knowledge about the policy monotonicity.

B. Enforcing monotonicity

To efficiently enforce policy monotonicity, two choices are made. The first choice is to represent the policy explicitly, rather than implicitly via the Q-function, as in the original online LSPI. This frees us from translating the monotonicity constraints into Q-function constraints – generally a daunting task. Since continuous-state policies cannot be represented exactly in general, the explicit representation comes at the expense of introducing policy approximation errors.

The second choice is to employ a linear policy parametrization:[1]

$$\widehat{h}(x) = \sum_{i=1}^{\mathscr{N}} \varphi_i(x)\vartheta_i = \varphi^{\mathrm{T}}(x)\vartheta \tag{9}$$

where $\varphi(x) = [\varphi_1(x), \ldots, \varphi_{\mathscr{N}}(x)]^{\mathrm{T}}$ are axis-aligned, normalized radial basis functions (RBFs) with their centers arranged on a grid and having identical widths.[2] The first and last grid points are placed at the boundaries of the hyperbox state space (7), and the grid spacing is equidistant along each dimension. With this specific policy approximator, in order to satisfy (8) it suffices to enforce a proper ordering of the parameters corresponding to each sequence of RBFs, along all the grid lines and in every dimension of the state space. We have verified the sufficiency of this condition using extensive experimentation, for many RBF configurations and parameter values, and we conjecture that it is also sufficient in general — although, to our knowledge, this has not been formally proven yet.

To develop a mathematical notation for this condition, denote the grid sizes along each dimension by, respectively, $\mathscr{N}_1, \ldots, \mathscr{N}_D$; there are $\mathscr{N} = \prod_{d=1}^{D} \mathscr{N}_d$ RBFs in total. Furthermore, denote by $\varphi_{i_1,\ldots,i_D}$ the RBF located at grid indices $i_1, \ldots, i_D$, and by $\vartheta_{i_1,\ldots,i_D}$ the parameter that multiplies this RBF in (9). We will use these D-dimensional indices interchangeably with the single-dimensional indices that appear in (9). Choosing any bijective mapping between the D-dimensional indices and the single-dimensional ones suffices to make these two indexing conventions equivalent.

The monotonicity conditions on the parameters can now be written in the following, linear form:

$$\begin{aligned}
&\delta_{\text{mon},1} \cdot \vartheta_{1,i_2,i_3,\ldots,i_D} \leq \delta_{\text{mon},1} \cdot \vartheta_{2,i_2,i_3,\ldots,i_D} \leq \ldots \\
&\quad \ldots \leq \delta_{\text{mon},1} \cdot \vartheta_{\mathscr{N}_1,i_2,\ldots,i_D} \quad \text{for all } i_2, i_3, \ldots, i_D, \\
&\delta_{\text{mon},2} \cdot \vartheta_{i_1,1,i_3,\ldots,i_D} \leq \delta_{\text{mon},2} \cdot \vartheta_{i_1,2,i_3,\ldots,i_D} \leq \ldots \\
&\quad \ldots \leq \delta_{\text{mon},2} \cdot \vartheta_{i_1,\mathscr{N}_2,i_3,\ldots,i_D} \quad \text{for all } i_1, i_3, \ldots, i_D, \\
&\qquad\qquad \cdots \qquad \cdots \qquad \cdots \\
&\delta_{\text{mon},D} \cdot \vartheta_{i_1,i_2,i_3,\ldots,1} \leq \delta_{\text{mon},D} \cdot \vartheta_{i_1,i_2,i_3,\ldots,2} \leq \ldots \\
&\quad \ldots \leq \delta_{\text{mon},D} \cdot \vartheta_{i_1,i_2,i_3,\ldots,\mathscr{N}_D} \quad \text{for all } i_1, i_2, \ldots, i_{D-1}
\end{aligned} \tag{10}$$

[1] We use calligraphic notation to differentiate mathematical objects related to policy approximation from those related to value function approximation (e.g., the policy parameters are denoted by ϑ, whereas the value function parameters are denoted by θ).

[2] The formula to compute the ith (nonnormalized) RBF is $\bar{\varphi}_i(x) = \exp\left[-\sum_{d=1}^{D}(x_d - c_{i,d})^2 / b_{i,d}^2\right]$ where $c_{i,d}$ is the center coordinate along the dth dimension, and $b_{i,d}$ is the width along this dimension.

The total number of inequalities in (10) is:

$$\sum_{d=1}^{D}\left((\mathcal{N}_d-1)\prod_{d'=1,d'\neq d}^{D}\mathcal{N}_{d'}\right)$$

This type of constraints is easier to understand in the two-dimensional case. For instance, an ordering corresponding to a 3×3 grid of RBFs could be:

$$\begin{array}{ccccc} \vartheta_{1,1} & \leq & \vartheta_{1,2} & \leq & \vartheta_{1,3} \\ \geq & & \geq & & \geq \\ \vartheta_{2,1} & \leq & \vartheta_{2,2} & \leq & \vartheta_{2,3} \\ \geq & & \geq & & \geq \\ \vartheta_{3,1} & \leq & \vartheta_{3,2} & \leq & \vartheta_{3,3} \end{array} \tag{11}$$

in which case the policy would be decreasing along the first dimension of X – vertically in (11) – and increasing along the second dimension – horizontally in (11).

C. Online LSPI with monotonic policies

The prior knowledge about policy monotonicity is employed in online LSPI by replacing the unconstrained policy improvement (line 11 of Algorithm 1) with the constrained least-squares problem:

$$\vartheta_{\ell+1}=\underset{\vartheta \text{ satisfying (10)}}{\arg\min}\sum_{i_s=1}^{\mathcal{N}_s}\left(\varphi^{\mathrm{T}}(x_{i_s})\vartheta-u_{i_s}\right)^2 \quad \text{where } u_{i_s}\in\arg\max_u \phi^{\mathrm{T}}(x_{i_s},u)\theta_\ell \tag{12}$$

Here, $\{x_1,\ldots,x_{\mathcal{N}_s}\}$ is an arbitrary set of state samples to be used for policy improvement. Since the constraints (10) are linear, the problem (12) can be efficiently solved using quadratic programming. The parameter vector $\vartheta_{\ell+1}$ leads to a monotonic and improved approximate policy $\hat{h}_{\ell+1}(x)=\varphi^{\mathrm{T}}(x)\vartheta_{\ell+1}$, which is used instead of the unconstrained policy to choose actions and in the updates of Γ.

For completeness, Algorithm 2 summarizes online LSPI with monotonic policies, a general linear parametrization of the Q-function, and ε-greedy exploration.

Algorithm 2 Online LSPI with monotonic policies.

Input: Q-function BFs $\phi_1,\ldots,\phi_n$, policy BFs $\varphi_1,\ldots,\varphi_{\mathcal{N}}$; set of samples $\{x_1,\ldots,x_{\mathcal{N}_s}\}$; γ; K_θ; $\{\varepsilon_k\}_{k=0}^{\infty}$; δ

1: $\ell\leftarrow 0$; initialize policy parameter ϑ_0
2: $\Gamma\leftarrow\delta I_{n\times n}$; $z\leftarrow 0$
3: measure initial state x_0
4: **for** each time step $k\geq 0$ **do**
5: $\quad u_k\leftarrow\begin{cases}\varphi^{\mathrm{T}}(x_k)\vartheta_\ell & \text{w.p. } 1-\varepsilon_k\\ \text{a uniform random action in } U & \text{w.p. } \varepsilon_k\end{cases}$
6: $\quad$ apply u_k, measure next state x_{k+1} and reward r_{k+1}
7: $\quad\Gamma\leftarrow\Gamma+\phi(x_k,u_k)\phi^{\mathrm{T}}(x_k,u_k)-\gamma\phi(x_k,u_k)\phi^{\mathrm{T}}(x_{k+1},\varphi^{\mathrm{T}}(x_{k+1})\vartheta_\ell)$
8: $\quad z\leftarrow z+\phi(x_k,u_k)\,r_{k+1}$
9: $\quad$ **if** $k=(\ell+1)K_\theta$ **then**
10: $\quad\quad$ find θ_ℓ by solving $\Gamma\theta_\ell=z$
11: $\quad\quad$ find $\vartheta_{\ell+1}$ by solving (12)
12: $\quad\quad\ell\leftarrow\ell+1$
13: $\quad$ **end if**
14: **end for**

To generalize this framework to multiple action variables, a distinct policy parameter vector should be used for every action variable, and the monotonicity constraints should be enforced separately, for each of these parameter vectors. Different monotonicity directions can be imposed for different action variables.

IV. Experimental study

In this section, we investigate the effects of using prior knowledge in online LSPI. To this end, in a simulation example involving the stabilization of a DC motor, we compare the learning performance of online LSPI with prior knowledge (Algorithm 2), with the performance of the original online LSPI (Algorithm 1), which does not use prior knowledge.

A. DC motor problem

The DC motor is described by the discrete-time dynamics:

$$f(x,u)=Ax+Bu$$
$$A=\begin{bmatrix}1 & 0.0049\\ 0 & 0.9540\end{bmatrix},\quad B=\begin{bmatrix}0.0021\\ 0.8505\end{bmatrix}$$

where $x_1=\alpha\in[-\pi,\pi]$ rad is the shaft angle, $x_2=\dot{\alpha}\in[-16\pi,16\pi]$ rad/s is the angular velocity, and $u\in[-10,10]$ V is the control input (voltage). The state variables are restricted to their domains using saturation. The goal is to stabilize the system around $x=0$, and is described by the quadratic reward function:

$$\rho(x,u)=-x^{\mathrm{T}}Q_{\mathrm{rew}}x-R_{\mathrm{rew}}u^2$$
$$Q_{\mathrm{rew}}=\begin{bmatrix}5 & 0\\ 0 & 0.01\end{bmatrix},\quad R_{\mathrm{rew}}=0.01$$

with discount factor $\gamma=0.95$.

Because the dynamics are linear and the reward function is quadratic, the optimal policy would be a linear state feedback, of the form $h(x)=L^{\mathrm{T}}x$, if the constraints on the state and action variables were disregarded. The gain vector L can be computed from f and ρ, using an extension of linear quadratic control to the discounted case, as explained, e.g., in, Section 3.2 of [3]. The result is $L=[-12.92,-0.68]^{\mathrm{T}}$, corresponding to a policy that monotonically decreases along both axes of the state space. This monotonicity property will be used in the sequel. Note that only prior knowledge about the *signs* of the feedback gains is required to establish the policy monotonicity directions, and the actual *values* of these gains are not needed.

B. Policy and Q-function approximators, parameter settings, and performance criterion

To apply online LSPI with monotonicity constraints, the policy is represented using a grid of RBFs, as described in Section III-B. The grid contains 9×9 RBFs, so the policy has 81 parameters. The RBF width along each dimension is identical to the distance between two adjacent RBFs along that dimension (the grid step). To perform the policy improvements (12), $\mathcal{N}_s=1000$ uniformly distributed, random state samples are employed.

The Q-function approximator relies on the same grid of state-dependent RBFs as the policy approximator, and on a discretization of the action space into 3 discrete values: $\{-10, 0, 10\}$. (Of course, in general the Q-functions BFs can be chosen independently from the policy BFs.) To obtain the state-action BFs required for Q-function approximation (4), the RBFs are replicated for every discrete action, obtaining a total of $81 \cdot 3 = 243$ BFs. When computing approximate Q-values, all the BFs that do not correspond to the current discrete action are taken equal to 0, i.e., the vector of Q-function BFs is $\phi(x,u) = [\mathscr{I}(u=-10)\cdot\varphi^{\mathrm{T}}(x), \mathscr{I}(u=0)\cdot\varphi^{\mathrm{T}}(x), \mathscr{I}(u=10)\cdot\varphi^{\mathrm{T}}(x)]^{\mathrm{T}}$, where the indicator function $\mathscr{I}$ is 1 when its argument is true, and 0 otherwise.

Note that, although the parameterized policy produces continuous actions, the Q-function approximator only works for the discrete actions considered. Therefore, the continuous actions produced by the policy must be discretized during learning, at lines 5 and 7 of Algorithm 2.

The learning experiment has a length of 600 s and is divided into 1.5 s learning trials. The initial state of each trial is chosen randomly from a uniform distribution over the state space. The policy is improved once every $K_\theta = 100$ transitions. An exponentially decaying exploration schedule is used that starts from an initial probability $\varepsilon_0 = 1$, and decays so that after $t = 200$ s, ε becomes 0.1. The parameter δ is set to 0.001. The initial policy parameters, together with the resulting policy, are identically zero.

The original online LSPI employs the same Q-function approximator and settings as online LSPI with prior knowledge, but does not approximate the policy or enforce monotonicity constraints. Instead, it computes greedy actions on demand, by maximizing the Q-function (see Section II). The initial policy chooses the first discrete action (-10) for any state.

After each online LSPI experiment is completed, snapshots of the policy taken at increasing moments of time are evaluated. This produces a curve recording the control performance of the policy over time. During performance evaluation, learning and exploration are turned off. Policies are evaluated using simulation, by estimating their average return (score) over the grid of initial states $X_0 = \{-\pi, -\pi/2, 0, \pi/2, \pi\} \times \{-10\pi, -5\pi, -2\pi, -\pi, 0, \pi, 2\pi, 5\pi, 10\pi\}$. The return from each state on this grid is estimated by simulating only the first K steps of the controlled trajectory, with K chosen large enough to guarantee the estimate is within a 0.1 distance of the true return.

C. Results and discussion

Figure 1 shows the learning performance of online LSPI with monotonic policies, in comparison to the performance of the original online LSPI algorithm. Mean values across 20 independent runs are reported, together with 95% confidence intervals on these means.

Using prior knowledge leads to much faster and more reliable learning: the score reliably converges in around 50 s of simulation time, during which 10 000 samples are observed. In contrast, online LSPI without prior knowledge requires more than 300 s (60 000 samples) to reach a near-optimal performance, and has a larger variation in performance across the 20 runs, which can be seen in the wider 95% confidence intervals.

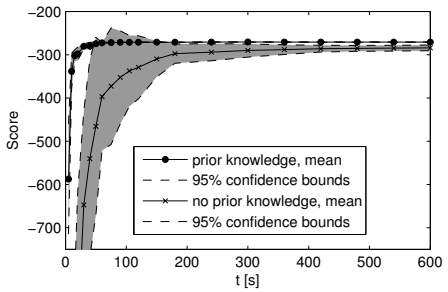
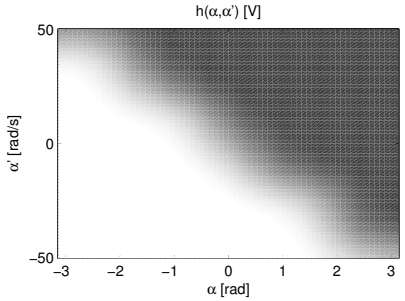


Fig. 1. Performance comparison between online LSPI with prior knowledge and the original online LSPI algorithm. The horizontal axis shows the time spent interacting with the system (simulation time).

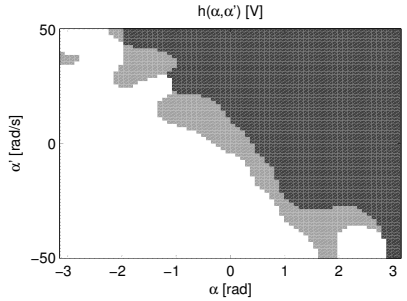
Figure 2 compares a representative solution obtained using prior knowledge with one obtained by the original online LSPI. The policy of Figure 2(b), obtained without using prior knowledge, violates monotonicity in several areas. The control performance of the monotonic policy – Figure 2(c) – is better than the performance of the policy found without prior knowledge – Figure 2(d). This difference appears mainly because the monotonic policy outputs continuous actions. Recall however from Section IV-B that this advantage cannot be exploited *during* learning, when the actions must be discretized to make them compatible with the Q-function approximator.

The mean execution time of online LSPI with prior knowledge is 1046.5 s, with a 95% confidence interval of $[1024.2, 1068.9]$ s. For the original online LSPI algorithm, the mean execution time is 87.7 s with a confidence interval of $[81.8, 93.6]$ s. These execution times were recorded while running the algorithms in MATLAB 7 on a PC with an Intel Core 2 Duo E6550 2.33 GHz CPU and with 3 GB of RAM.

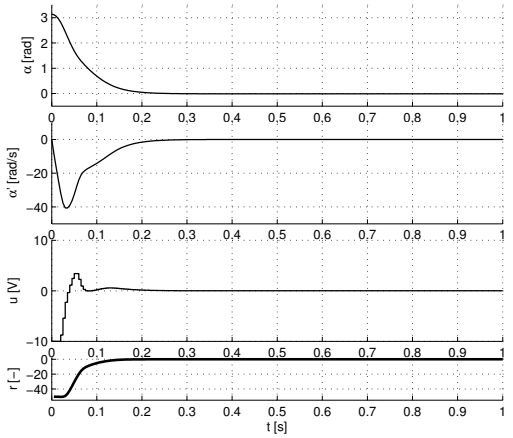
So, although online LSPI with prior knowledge learns faster in terms of *simulation* time (number of transition samples observed), its *execution* time is larger. This is mainly because the constrained policy improvements (12) are more computationally demanding than the original policy improvements (3). In particular, solving (12) takes much longer than a sampling period (around 0.75 s, whereas $T_\mathrm{s} = 0.005$ s), which means that the algorithm cannot be directly applied in real-time. To address this difficulty, besides the obvious solution of optimizing the implementation (e.g., by switching from MATLAB to C, which should provide a significant boost in execution speed), another possibility is to perform the policy improvements *asynchronously*, on a different thread than the one responsible with controlling the system. This thread could run on another CPU core or even on another computer. While executing policy improvement, the system should be controlled with the previously available policy, possibly collecting transition samples for later use in evaluating the new policy.



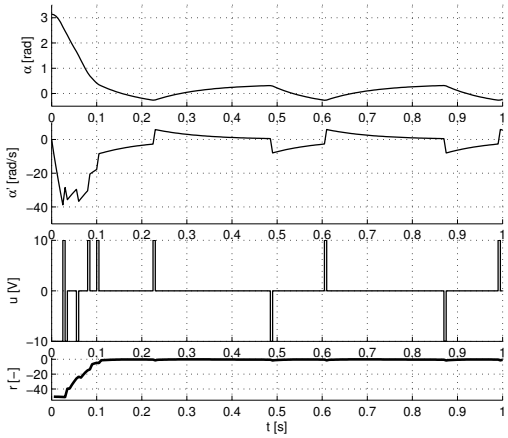
(a) Policy found using prior knowledge.



(b) Policy found without prior knowledge.



(c) Trajectory controlled by the policy found using prior knowledge.



(d) Trajectory controlled by the policy found without prior knowledge.

Fig. 2. Representative solutions found using prior knowledge (left) and without prior knowledge (right).

V. Conclusions

In this paper, we have introduced an approach to integrate prior knowledge into online least-squares policy iteration. In particular, we have considered problems in which a (near-)optimal policy is known to be monotonic in the state variables. For an example involving the control of a DC motor, using this type of prior knowledge has led to much faster (in terms of time spent interacting with the system) and more reliable learning.

While the global monotonicity requirement is restrictive in general, our approach can easily be extended to handle other types of monotonicity restrictions. For instance, the policy could be monotonic only with respect to a subset of state variables, or only over a subregion of the state space, such as in a neighborhood of an equilibrium. Multiple monotonicity regions can also be considered. Another research direction is representing prior knowledge using inequality constraints of the form $g_{\text{in}}(x, h(x)) \leq 0$, and equality constraints of the form $g_{\text{eq}}(x, h(x)) = 0$. Unlike the monotonicity property, such constraints can be exploited without representing the policy explicitly. They can be enforced while computing improved actions, separately for every state x where such actions are necessary.

References

[1] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1996.

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

[3] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed. Athena Scientific, 2007, vol. 2.

[4] J. Boyan, "Technical update: Least-squares temporal difference learning," *Machine Learning*, vol. 49, pp. 233–246, 2002.

[5] A. Nedić and D. P. Bertsekas, "Least-squares policy evaluation algorithms with linear function approximation," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 13, no. 1–2, pp. 79–110, 2003.

[6] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[7] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, vol. 6, pp. 503–556, 2005.

[8] H. Yu and D. P. Bertsekas, "Convergence results for some temporal difference methods based on least squares," *IEEE Transactions on Automatic Control*, vol. 54, no. 7, pp. 1515–1531, 2009.

[9] L. Buşoniu, D. Ernst, B. D. Schutter, and R. Babuška, "Online least-squares policy iteration for reinforcement learning control," in *Proceedings 2010 American Control Conference (ACC-10)*, Baltimore, US, 30 June – 2 July 2010, accepted for publication.

[10] R. S. Sutton, "Learning to predict by the method of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[11] L. Li, M. L. Littman, and C. R. Mansley, "Online exploration in least-squares policy iteration," in *Proceedings 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-09)*, vol. 2, Budapest, Hungary, 10–15 May 2009, pp. 733–739.

[12] T. Jung and D. Polani, "Kernelizing LSPE(λ)," in *Proceedings 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL-07)*, Honolulu, US, 1–5 April 2007, pp. 338–345.