Batch Reinforcement Learning for Controlling a Mobile Wheeled Pendulum Robot

Andrea Bonarini, Claudio Caccia, Alessandro Lazaric, and Marcello Restelli

Abstract In this paper we present an application of Reinforcement Learning (RL) methods in the field of robot control. The main objective is to analyze the behavior of batch RL algorithms when applied to a mobile robot of the kind called *Mobile Wheeled Pendulum* (MWP). In this paper we focus on the common problem in classical control theory of following a reference state (e.g., position set point) and try to solve it by RL. In this case, the state space of the robot has one more dimension, in order to represent the desired variable state, while the cost function is evaluated considering the difference between the state and the reference. Within this framework some interesting aspects arise, like the ability of the RL algorithm to generalize to reference points never considered during the training phase. The performance of the learning method has been empirically analyzed and, when possible, compared to a classic control algorithm, namely linear quadratic optimal control (LQR).

1 Introduction

This paper is about the application of Reinforcement Learning (RL) methods [10] in the field of robot control. To achieve optimal performance, many feedback control techniques (e.g., PID, direct pole placement, optimal control, etc.) generally require very accurate models of the dynamics of the robot and of its interaction with the

Andrea Bonarini, Alessandro Lazaric, Marcello Restelli
Dept. of Electronics and Information
Politecnico di Milano
Piazza Leonardo da Vinci 32, I-20133 Milan, Italy
e-mail: {bonarini,lazaric,restelli}@elet.polimi.it

Claudio Caccia
Dept. of Informatics, Systems and Communication
Università degli Studi di Milano - Bicocca
Viale Sarca 336, I-20126 Milan, Italy e-mail: caccia@disco.unimib.it

Please use the following format when citing this chapter:

Bonarini, A., Caccia, C., Lazaric, A. and Restelli, M., 2008, in IFIP International Federation for Information Processing, Volume 276; *Artificial Intelligence and Practice II*; Max Bramer; (Boston: Springer), pp. 151–60.

surrounding environment, which is often infeasible in many real situations. Using traditional RL techniques (e.g., Q-learning), a robot can learn the optimal control policy by directly interacting with the environment, without knowing any model in advance. On the other hand, collecting data through direct interaction is a very long process when real robots are considered. Furthermore, RL algorithms are typically used to solve single-task problems such as balancing an inverted pendulum, driving the robot to a goal state, or learning to reach a given set point. This implies that every time the goal changes the learning process must restart from scratch, thus making infeasible to cope with a common problem in control theory like following a continuously changing reference point (e.g., position or speed set points). To face this class of problems, the state space of the problem needs to be enlarged by adding a new variable to represent the desired state, while the cost function can be defined on the error signal (i.e., the distance between the current state and the desired one).

In this paper, we propose to use *fitted Q-iteration* algorithms [6, 9, 2], i.e., batch algorithms that decompose the original RL problem into a sequence of supervised problems defined on a set of samples of state transitions. Since the value of the reference state does not affect the transition model, but only the reward function, using a batch approach allows to reuse the same set of transition samples to train the controller for different values of the reference state thus reducing the time of direct interaction with the environment. Within this framework some interesting aspects arise, for example the ability of the RL algorithm to generalize to reference points never seen during the training phase. The experimental activity has been carried on using a model of a mobile robot of the kind called Mobile Wheeled Pendulum. We experimentally evaluate batch RL algorithms using different function-approximation techniques, and compare their accuracies when following a given angle profile.

The rest of the paper is organized as follows: next section briefly motivates the use of batch RL algorithms and reviews the main state-of-the-art approaches. Section 3 describes the dynamic model of the robot Tilty used in the experiments. In Section 4 we describe how to collect data and how to train batch RL algorithms to build automatic controllers, and in Section 5 we show the results obtained using both neural networks and extra-randomized trees. Section 6 draws conclusions and proposes new directions for future research.

2 Batch Reinforcement Learning

In reinforcement learning [10] problems, an agent interacts with an unknown environment. At each time step, the agent observes the *state*, takes an *action*, and receives a *reward*. The goal is to learn a *policy* (i.e., a mapping from states to actions) that maximizes the long-term return. An RL problem can be modeled as a Markov Decision Process (MDP) defined by a tuple $\langle \mathscr{S}, \mathscr{A}, \mathscr{T}, \mathscr{R}, \gamma \rangle$, where $\mathscr{S}$ is the set of states, $\mathscr{A}(s)$ is the set of actions available in state s, $\mathscr{T} : \mathscr{S} \times \mathscr{A} \times \mathscr{S} \to [0,1]$ is a transition distribution that specifies the probability of observing a certain state after taking a given action in a given state, $\mathscr{R} : \mathscr{S} \times \mathscr{A} \to \mathbb{R}$ is a reward

function that specifies the instantaneous reward when taking a given action in a given state, and $\gamma \in [0,1]$ is a discount factor. The policy of an agent is characterized by a probability distribution $\pi : \mathscr{S} \times \mathscr{A} \rightarrow [0,1]$ that specifies the probability of taking a certain action in a given state. The utility of taking action a in state s and following a policy π thereafter is formalized by the action-value function $Q^{\pi}(s,a) = E\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t | s = s_1, a = a_1, \pi\right]$, where $r_1 = \mathscr{R}(s,a)$. RL approaches aim at learning the policy that maximizes the action-value function in each state. The optimal action-value function is the solution of the Bellman equation: $Q^*(s,a) = R(s,a) + \gamma \sum_{s'} \mathscr{T}(s,a,s') \max_{a'} Q^*(s',a')$. The optimal policy $\pi^*(\cdot,\cdot)$ takes in each state the action with the highest utility.

Temporal Difference algorithms [10] allow the computation of $Q^*(s,a)$ directly interacting with the environment with a trial-and-error process. Given the tuple $\langle s_t, a_t, r_t, s_{t+1} \rangle$ (experience made by the agent), at each step, action values may be estimated by online algorithms, such as Q-learning [10], whose update rule is: $Q(s_t,a_t) \leftarrow (1-\alpha) Q(s_t,a_t) + \alpha (r_t + \gamma \max_{a'} Q(s_{t+1},a'))$:$\alpha \in [0,1]$ learning rate.

RL has proven to be an effective approach to solve finite MDPs. On the other hand, using RL techniques in robotic and control applications rises several difficulties. Since state and action spaces are high-dimensional and continuous, the value function cannot be represented by means of tabular approaches, but function-approximation techniques are required. Despite some successful applications, coupling function approximation with *online* RL algorithms can lead to oscillatory behaviors or even to divergence [1]. The reason for this is that, unlike in the supervised case, in RL we cannot sample from the target function, and the training samples depend on the function approximator itself. Recently, several studies have focused on developing *batch* RL algorithms. While in online learning the agent modifies its control policy at each time step according to the experience gathered from the environment, the batch approach aims at determining the best control policy given a set of experience samples $\langle s_t, a_t, r_t, s_{t+1} \rangle$ previously collected by the agent following a given sampling strategy. In particular, good results have been achieved by *fitted Q-iteration* algorithms derived from the *fitted value iteration* approach [3]. The idea is to reformulate the RL problem as a sequence of supervised learning problems. Given the dataset, in the first iteration of the algorithm, for each tuple $\langle s_i, a_i, r_i, s_i' \rangle$, the corresponding training pair is set to $(s_i, a_i) \rightarrow r_i$, and the goal is to approximate the expected immediate reward $Q_1(s,a) = E[\mathscr{R}(s_t,a_t) | s_t = s, a_t = a]$.

The second iteration, based on the approximation of the Q_1-function, extends the optimization horizon one step further: $Q_2(s,a) = \mathscr{R}(s,a) + \gamma \max_{a'} \hat{Q}_1(s',a')$. At the N^{th} iteration, using the approximation of the Q_{N-1}-function, we can compute an approximation of the optimal action-value function at horizon N.

The batch approach allows to use any regression algorithm, and not only parametric function approximators as happens for stochastic approximation algorithms. Several studies have reported very good results with a wide range of approximation techniques: kernel-based regressors [6], tree-based regressors [2], and neural networks [9]. All these works show how batch mode RL algorithms allow to effectively exploit the information contained in the collected samples, so that, even using small datasets, very good performances can be achieved. The size of the dataset is

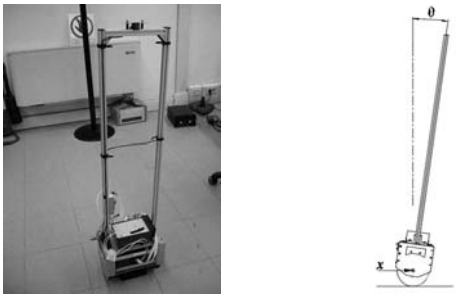

Fig. 1 The MWP robot *Tilty* and its degrees of freedom

a key factor for robotic applications, since collecting a large amount of data with real robots may be expensive and dangerous. As shown in [9], it is possible to solve simple control problems, such as balancing a pole, with a dataset obtained by performing random actions for a few minutes.

In this paper, we study the problem of controlling a mobile wheeled pendulum to follow a time-dependent set point. As we will explain in Section 5, fitted Q-iteration algorithms are well-suited to cope with this class of problems and we will show the results achieved with different function approximators.

3 The Robot: *Tilty*

The mobile wheeled inverted pendulum robot named *Tilty* has been considered for the tests. Tilty has an aluminum frame, a pair of wheels on the same axis connected to a DC motor each, batteries and a programmable drive. The robot structure is represented in Figure 1. The system has two types of sensors onboard: encoders on motors and a dual-axis accelerometer.

3.1 Dynamical Model

The first analysis consists in the study of the equations that describe the dynamics of the robot. In our case the model describing *Tilty* is non-linear: i.e., the system cannot be described by a system of equations of the form $\dot{x}(t) = \mathbf{A} \cdot x(t) + \mathbf{B} \cdot u(t)$, with $\mathbf{A}$ and $\mathbf{B}$ constant matrices, but by the general form $\dot{x} = f(x(t), u(t))$.

For mechanical or dynamical models, a common way to obtain such non-linear equations is to use the *Lagrange equations* [4], [8]:

$$\frac{d}{dt}\left(\frac{\partial \mathbf{T}}{\partial \dot{q}}\right) - \frac{\partial \mathbf{T}}{\partial q} - \frac{\partial \mathbf{V}}{\partial q} = \tau, \tag{1}$$

where q is the generic variable describing the pose of the system (i.e., degree of freedom), $\mathbf{T}$ is the kinetic energy, $\mathbf{V}$ is the potential energy, and τ represents the generalized force acting on the system. *Tilty* is described by two degrees of freedom when considering a linear motion (see Figure 1): the position of the center of the wheels x, and the angle ϑ between the pole and the vertical axis, while the input is represented by the motor torque $\mathbf{C}$ acting between the motors and the pendulum.

The equations for the kinetic and potential energy of the system are:

$$\begin{cases} \mathbf{T}(x,\vartheta) = \frac{1}{2}\cdot M_{tot}\dot{x}^2 + \frac{1}{2}\cdot J_{tot}\dot{\vartheta}^2 + H_{tot}\cdot\cos(\vartheta)\dot{\vartheta}\dot{x} \\ \mathbf{V}(x,\vartheta) = -H_{tot}(1-cos(\vartheta))\cdot g \end{cases} \tag{2}$$

where $M_{tot} = \sum m_i$ is the total mass of the system, $J_{tot} = \sum J_i + \sum m_i \cdot d_i^2$ is the moment of inertia w.r.t. the wheel axis, and $H_{tot} = \sum m_i \cdot d_i$ is the first order moment.

In order to calculate the generalized forces τ in (1), the virtual work of acting forces has to be determined: $\tau_\mathbf{i} = \frac{\delta^* L}{\partial q_i}$ with $\delta^* L = 2C \cdot \left(\frac{\delta^* x}{R_{wheel}} - \delta^* \vartheta\right)$. Where C is the motor torque acting on wheels. Solving the equations in (1) with the expressions in (2) brings to the following system of equations:

$$\begin{cases} M_{tot}\ddot{x} + H_{tot}\cos(\vartheta)\cdot\ddot{\vartheta} - H_{tot}\sin(\vartheta)\cdot\dot{\vartheta}^2 = \frac{2C}{R_{wheel}} \\ J_{tot}\ddot{\vartheta} - H_{tot}\cos(\vartheta)\cdot\ddot{x} - H_{tot}\sin(\vartheta)\cdot g = -2C \end{cases} \tag{3}$$

which represents the non-linear dynamics of *Tilty*. The system in (3) can be rearranged in the form:

$$\left\{\ddot{x},\dot{\vartheta},\ddot{\vartheta}\right\}' = \left[\mathbf{A}\left(\vartheta,\dot{\vartheta}\right)\right]\cdot\left\{\dot{x},\vartheta,\dot{\vartheta}\right\}' + \left\{\mathbf{B}\left(\vartheta\right)\right\}\cdot C \tag{4}$$

The matrix $\mathbf{A}$ and vector $\mathbf{B}$ in (4) are the state space representation of the system's dynamics.

3.2 Design of the controller

The system needs a regulator able to keep it in equilibrium. We developed a regulator of the kind LQR (linear quadratic regulator), obtained by optimal control theory [5]. Optimal control finds a regulator able to give the best performance with respect to a specific measure of the performance itself. LQR procedure is interesting because it allows to minimize the cost functional of the following equation (5) giving stable controllers and because it is applicable to Multi-Input Multi-Output (MIMO) systems.

$$\mathbf{J} = \int_0^\infty \left(x'\cdot\tilde{\mathbf{Q}}\cdot x + u'\cdot\tilde{\mathbf{R}}\cdot u\right)dt, \tag{5}$$

where x is the state of the system, u is the input variable, $\tilde{\mathbf{Q}}$ and $\tilde{\mathbf{R}}$ are weight matrices for state variables and actions. The values $\tilde{\mathbf{Q}}$ and $\tilde{\mathbf{R}}$ have been chosen to optimize the system response in the conditions of the experiments. The feedback control law that

minimizes the value of the cost is $\mathbf{u} = -K \cdot \mathbf{x}$. This feedback law is determined by solving the *Riccati Equation* [5]. Therefore, it is necessary to linearize (4) around an equilibrium point, so that matrix **A** and vector **B** become state independent. This approach gives controllers with good behavior, but are heavily dependent on the model of the system: it is necessary to describe exactly the geometry and the dynamics and assume that the linear approximation is good. RL methods, proposed here, do not need any prior knowledge about the system, so appear to be interesting when the model is strongly non-linear or when its parameters can be hardly estimated.

3.3 Simulation of the dynamics

The behavior of the controlled system has been simulated as accurately as possible, considering, among others, the following aspects:

- control frequency of 50Hz,
- the robot inclination ϑ is available by reading the output of the 2-axis accelerometer, obtaining the inclination by comparing the two data,
- the robot angular velocity $\dot{\vartheta}$ is not directly available by sensor reading, so its value is determined by means of a reduced observer block.

The model described here has been used to gather all the data used to apply batch RL algorithms and to compare the performance of the LQR controller with the policy determined by the learning algorithms.

4 Experimental Settings

The application of RL methods requires the agent to interact with the environment in order to learn the optimal policy. The interaction can be direct (on-line) or indirect (off-line). In the first case, the agent itself chooses the action based on what it has learned until then, and the policy (the *Q-function*) is estimated progressively. In the second case, the policy update is done in a batch fashion. Rather than choosing actions based on a policy, the agent observes state transitions due to actions externally determined. On the basis of the whole dataset, the optimal policy is computed. Therefore, the first step in batch RL methods consists in collecting samples $\langle s, a, r, s' \rangle$.

4.1 Data collection

The kind of raw data needed for training is made of tuples $(\mathbf{s}, a, \mathbf{s}')$, where $\mathbf{s} = \{\dot{x}, \vartheta, \dot{\vartheta}\}$ is the present state of the robot, a is the torque $\mathbf{C}$ applied, and $\mathbf{s}' =$

$\{\dot{x}', \vartheta', \dot{\vartheta}'\}$ is the next state reached from state $\mathbf{s}$ when $\mathbf{C}$ is applied. The whole dataset is composed by seven-tuple samples: $\left\{\dot{x}, \vartheta, \dot{\vartheta}, \mathbf{C}, \dot{x}', \vartheta', \dot{\vartheta}'\right\}$.

Each sample represents a Markovian transition. Using model-free RL algorithms, it is not required a priori knowledge about system dynamics, since it can be implicitly inferred by the samples obtained through direct interaction with the real robot.

In our approach, we consider the dynamical model described in Section 3.1. The model is initialized with a random state defined by the vector $[0, \vartheta_{in}, 0]'$, with ϑ_{in} varying in the range of ± 0.3rad. A random motor torque uniformly distributed in $\pm C_{max} = 7.6\ Nm$ is then applied to the system at frequency of 50Hz and the sequence of the states reached is collected with the same frequency. When the system reaches dangerous conditions (i.e., $|\vartheta| \geq 0.5$rad), the simulation is stopped and the system is initialized again. All the experiments have been carried out using datasets with $1,000$, $3,000$, and $5,000$ samples, that correspond, respectively, to $20s$, $60s$, and $100s$ of training in real time. During the phase of data collection no reference value is considered. Then, we have considered angular references by adding two values to each sample, thus obtaining the input vector of the training set: $\left\{\dot{x}, \vartheta, \dot{\vartheta}, \vartheta_{ref}, \mathbf{C}, \dot{x}', \vartheta', \dot{\vartheta}', \vartheta'_{ref}\right\}$. We made the assumption that the reference varies slowly w.r.t. the frequency of data collection, so that it can be considered constant during a single transition ($\vartheta'_{ref} = \vartheta_{ref}$). We consider the following set of reference values: $\vartheta_{ref} \in \{-0.1, 0, 0.1\}$rad. Since the reference value does not affect the dynamics of the system, we simply replicate the data previously collected for each reference value.

4.2 Training

Once the input data have been collected, we need to compute the output values. The first step is to consider the instantaneous rewards. The reward function is:

$$\mathscr{R}(s_t, a_t) = \begin{cases} -\left|\vartheta_t - \vartheta_{ref}\right| & \text{when } s_{t+1} \text{ not final} \\ -1 & \text{when } s_{t+1} \text{ final} \end{cases} \tag{6}$$

where a final state is the one in which the magnitude of angle ϑ exceeds 0.5 rad. As described in Section 2, fitted Q-iteration algorithms iteratively extend the time horizon by approximating Q-functions defined according to the following equation:

$$Q_k(s,a) = \mathscr{R}(s,a) + \gamma \cdot \max_b Q_{k-1}\left(s'_t, b\right). \tag{7}$$

On the basis of the approximation of the Q_{k-1}-function, it is possible to build the training set in order to get an approximation of the Q_k-function.

The first *Q-function*, Q_1, is the approximation of the direct rewards as calculated in (6). The training values of the following functions (Q_k) are determined by (7), using direct rewards and the approximation given in the previous step (Q_{k-1}). These values are the output values of Q_k used by the approximator.

For each experiment, we have approximated the Q-functions from Q_0 to Q_{50} using two kinds of function approximators: neural networks and extra-trees, which are briefly described in the following. For more details refer to [9, 2].

4.2.1 Training with neural networks

The training of neural networks follows the approach used for the *NFQ* algorithm [9]. The *Q-function* at the k^{th} step is represented by a neural network whose input is the tuple $(\dot{x}, \vartheta, \dot{\vartheta}, \vartheta_{ref}, C)$. The model considered for the network uses 2 hidden layers composed of 5 neurons each and an output layer composed of one neuron. The activation function is sigmoidal for the inner layers and linear for the output layer. The training method used to determine the set of weights and biases is Levenberg-Marquardt [7].

4.2.2 Training with Extra-Trees

Besides neural networks, we have performed experiments with extra-trees, a particular kind of regression tree ensemble. Each tree is built so that each test at a node is determined by selecting K candidate tests at random, and choosing the one with the highest score. The parameters used in our experiments are those proposed in [2]: 50 trees, 5 candidate tests, and each node must contain at least two samples.

5 Simulation Results

In this section, we present and discuss some of the results obtained with the fitted Q-iteration algorithm using neural networks (NN) and with extra-trees. To give an idea of the performances achievable by the learned controllers, in each graph we report three simulations, which correspond to controllers learned using datasets with different sizes. To compare the results, we show simulations starting from a fixed angular position: 0.2 rad.

Figure 2 compares the behavior of the LQR control with the behavior of the learned controllers when the angular set point is fixed to 0. It can be noticed that all the learned controllers are much faster than LQR to reach the set point. In particular, extra-trees get very close to the set point after only a few control steps, and neural networks take about one second to converge. On the other hand, using neural networks the controllers are much smoother than those achieved by extra-trees.

Figures 3 to 5 show the behavior with ϑ_{ref} varying according to different profiles. It is worth noting that all the controllers are able to approximately follow the given profiles, even if they have been trained only to follow three angular set points: $-0.1, 0, 0.1$. However, as we can see, neural networks are much more accurate (almost overlapping with the reference profile) than extra-trees. Extra-trees perform

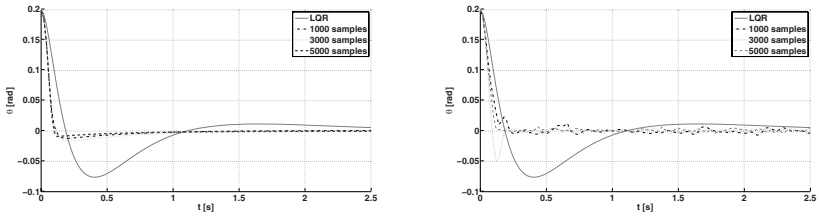

Fig. 2 Performance with $\vartheta_{ref} = 0$ (left:NN, right:Extra-Trees)

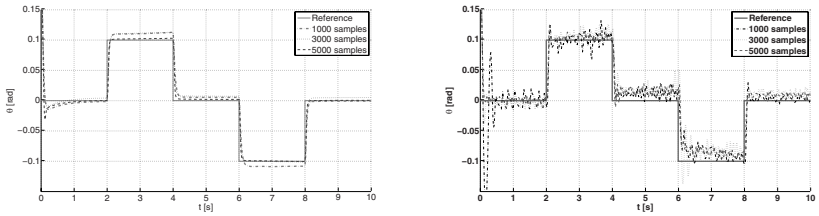

Fig. 3 Performance with ϑ_{ref} piecewise constant (left:NN, right:Extra-Trees)

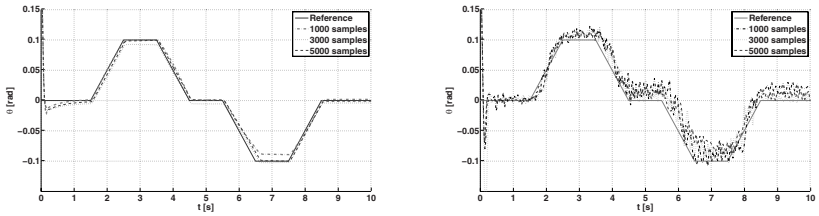

Fig. 4 Performance with ϑ_{ref} piecewise ramp (left:NN, right:Extra-Trees)

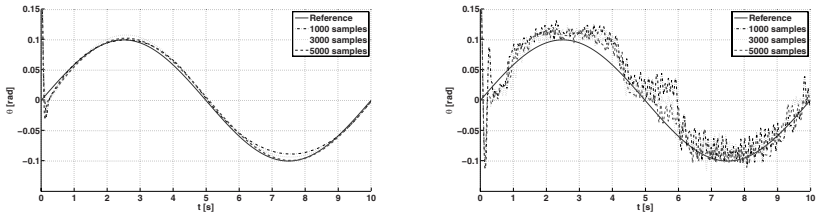

Fig. 5 Performance with ϑ_{ref} sinusoidal (left:NN, right:Extra-Trees)

quite poorly since they produce policies that make the robot reach speeds higher than those experienced using random exploration during the training phase, thus requiring hard extrapolation capabilities. This problem could be overcome by us-

ing the learned controller to collect and add further samples to the training set, and restarting the fitted Q-iteration algorithm.

As expected, controllers trained with larger datasets have better performances, even if it is worth noting that $1,000$ samples (corresponding to $20s$ of real time acquisition) are enough to learn quite good controllers.

6 Conclusions

In this paper, we presented batch RL methods to solve a robot control problem. The system considered here is unstable and non-linear, thus classic controllers require an approximated model. RL methods do not need any model of the robot and overcome problems of parameter identification. RL methods are generally used to solve single-task problems, while controllers generally follow changing reference points. We extended the idea of reference following to RL. The experiments show that a few tens of seconds are enough for batch RL algorithms to learn good controllers (even better than a classic controller like LQR). In particular, we have proposed a novel procedure that allows to learn controllers able to follow a varying reference point. It is interesting to note that the learned controllers effectively generalize to reference point not considered in the training phase. Given these encouraging results, we will experiment the proposed approach on the real robot.

References

1. Baird, L.C.: Residual algorithms: Reinforcement learning with function approximation. In: Proceedings of the 12th Intl. Conf. on Machine Learning, pp. 30–37 (1995)
2. Ernst, D., Geurts, P., Wehenkel, L.: Tree-based batch mode reinforcement learning. Journal of Machine Learning Research **6**, 503–556 (2005)
3. Gordon, G.J.: Approximate solutions to markov decision processes. Ph.D. thesis, Carnegie Mellon University (1999)
4. Landau, L., Lifshitz, E.M.: Mechanics, Course of Theoretical Physics, Volume 1. Pergamon Pres (1976)
5. Ogata, K.: Modern Control Engineering (4th ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA (2001)
6. Ormoneit, D., Sen, S.: Kernel-based reinforcement learning. Machine Learning **49**(2-3), 161–178 (2002)
7. Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T.: Numerical Recipes: The Art of Scientific Computing. Cambridge Univ. Press, New York, 1989. (1989)
8. Reddy, J.: Energy Principles and Variational Methods in Applied Mechanics (2nd ed.). John Wiley and Sons, Hoboken, NJ, USA (2002)
9. Riedmiller, M.: Neural fitted q iteration - first experiences with a data efficient neural reinforcement learning method. In: Proceedings of European Conference on Machine Learning, pp. 317–328 (2005)
10. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)