Learning Control in Robotics

Trajectory-Based Optimal Control Techniques

© STOCKBYTE, EYEWIRE, DIGITAL VISION & BRAND X PICTURES

Robot Learning

In a not too distant future, robots will be a natural part of daily life in human society, providing assistance in many areas ranging from clinical applications, education and care giving, to normal household environments [1]. It is hard to imagine that all possible tasks can be preprogrammed in such robots. Robots need to be able to learn, either by themselves or with the help of human supervision. Additionally, wear and tear on robots in daily use needs to be automatically compensated for, which requires a form of continuous self-calibration, another form of learning. Finally, robots need to react to stochastic and dynamic environments, i.e., they need to learn how to optimally adapt to uncertainty and unforeseen changes. Robot learning is going to be a key ingredient for the future of autonomous robots.

While robot learning covers a rather large field, from learning to perceive, to plan, to make decisions, etc., we will focus this review on topics of learning control, in particular, as it is concerned with learning control in simulated or actual physical robots. In general, learning control refers to the process of acquiring a control strategy for a particular control system and a particular task by trial and error. Learning control is usually distinguished from adaptive control [2] in that the learning system can have rather general optimization objectives—not just, e.g., minimal tracking error—and is permitted to fail during the process of learning, while adaptive control emphasizes fast convergence without failure. Thus, learning control resembles the way that humans and animals acquire new movement strategies, while adaptive control is a special case of learning control that fulfills stringent performance constraints, e.g., as needed in life-critical systems like airplanes.

Learning control has been an active topic of research for at least three decades. However, given the lack of working robots that actually use learning components, more work needs to be done before robot learning will make it beyond the laboratory environment. This article will survey some ongoing and past activities in robot learning to assess where the field stands and where it is going. We will largely focus on nonwheeled robots and less on topics of state estimation, as typically explored in wheeled robots [3]–[6], and we emphasize learning in continuous state-action spaces rather than discrete state-action spaces [7], [8]. We will illustrate the different topics of robot learning with examples from our own research with anthropomorphic and humanoid robots.

Digital Object Identifier 10.1109/MRA.2010.936957

The Basics of Learning Control

A key question in learning control is what it is that should be learned. To address this issue, it is helpful to begin with one of the most general frameworks of learning control, as originally developed in the middle of the 20th century in the fields of optimization theory, optimal control, and in particular, dynamic programming [9], [10]. Here, the goal of learning control was formalized as the need to acquire a task-dependent control policy π that maps a continuous-valued state vector $\mathbf{x}$

BY STEFAN SCHAAL AND CHRISTOPHER G. ATKESON

1070-9932/10/$26.00©2010 IEEE

of a controlled system and its environment, possibly in a time t dependent way, to a continuous-valued control vector $\mathbf{u}$:

$$\mathbf{u} = \pi(\mathbf{x}, t, \theta). \tag{1}$$

The parameter vector θ contains the problem-specific parameters in the policy π that need to be adjusted by the learning system. The controlled system can generally be expressed as a nonlinear dynamics function

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t, \boldsymbol{\varepsilon}_x) \tag{2}$$

with observation equations

$$\mathbf{y} = \mathbf{h}(\mathbf{x}, \mathbf{u}, t, \boldsymbol{\varepsilon}_y) \tag{3}$$

that describe how the observations $\mathbf{y}$ of the system are derived from the full-state vector $\mathbf{x}$—the terms $\boldsymbol{\varepsilon}_x$ and $\boldsymbol{\varepsilon}_y$ denote noise terms. Thus, learning control means finding a (usually nonlinear) function π that is adequate for a given desired behavior and movement system. A repertoire of motor skills is composed of many such policies that are sequenced and superimposed to achieve complex motor skills.

How the control policy is learned, however, can proceed in many different ways. Assuming that the model equations (2) and (3) are unknown, one classical approach is to learn these models using methods of function approximation and then compute a controller based on the estimated model, which is often discussed as the certainty-equivalence principle in the adaptive control literature [2]. Such techniques are summarized under the name model-based learning, indirect learning, or internal model learning. Alternatively, model-free learning of the policy is possible given an optimization or reward criterion, usually using methods from optimal control or reinforcement learning. Such model-free learning is also known as direct learning, since the policy is learned directly, i.e., without a detour through model identification.

It is useful to distinguish between several general classes of motor tasks that could be the goal of learning. Regulator tasks keep the system at a particular set point of operation—a typical example is balancing a pole on a fingertip or standing upright on two legs. Tracking tasks require the control system to follow a given desired trajectory within the abilities of the control system. Discrete movement tasks, also called one-shot tasks, are defined by achieving a particular goal at which the motor skill terminates. A basketball foul shot or grasping a cup of coffee are representative examples. Periodic movement tasks are typical in the domain of locomotion. At last, complex movement tasks are composed of sequencing and superimposing simpler motor skills, e.g., leading to complex manipulation skills like emptying a dishwasher or assembling a bookshelf.

From the viewpoint of machine learning, robot learning can be classified as supervised learning, reinforcement learning, learning modularizations, or learning feature representations that subserve learning. All learning methods can benefit from giving the learning system prior knowledge about how to accomplish a motor task, and imitation learning or learning from demonstration is a popular approach to introduce this bias.

In summary, the goal of robot learning is to find an appropriate control policy to accomplish a given movement task, assuming that no traditional methods exist to compute the control policy. Approaches to robot learning can be classified and discussed using three dimensions: direct versus indirect control, the learning method used, and the class of tasks in question (Figure 1).

Approaches to Robot Learning

We will use the classification in Figure 1 in the following sections to guide our survey of current and previous work in robot learning. Given space constraints, this survey is not meant to be comprehensive but rather to present illustrative projects in the various areas.

Learning Internal Models for Control

Using learning to acquire internal models for control is useful when the analytical models are too complex to derive, and/or when it can be expected that the models change over time, e.g., due to wear and tear. Various kinds of internal models are used in robotics. The most well known are kinematics and dynamic models. For instance, the direct kinematics of a robot relates joint variables $\mathbf{q}$ to end-effector variables $\mathbf{y}$, i.e., $\mathbf{y} = \mathbf{g}(\mathbf{q})$ [11]. Dynamics models include kinetic terms like forces or torques, as in (2). The previous models are forward models, i.e., they model the causal relationship between input and output variables, and they are proper functions. Often, however, what is needed in control are inverse models, e.g., the inverse kinematics $\mathbf{q} = \mathbf{g}^{-1}(\mathbf{y})$ or the inverse dynamics $\mathbf{u} = \mathbf{f}^{-1}(\mathbf{q}, \dot{\mathbf{q}}, t)$. As discussed in [12], inverse models are often not functions, as the inverse relationships may be a one-to-many map, i.e., just a relation. Such

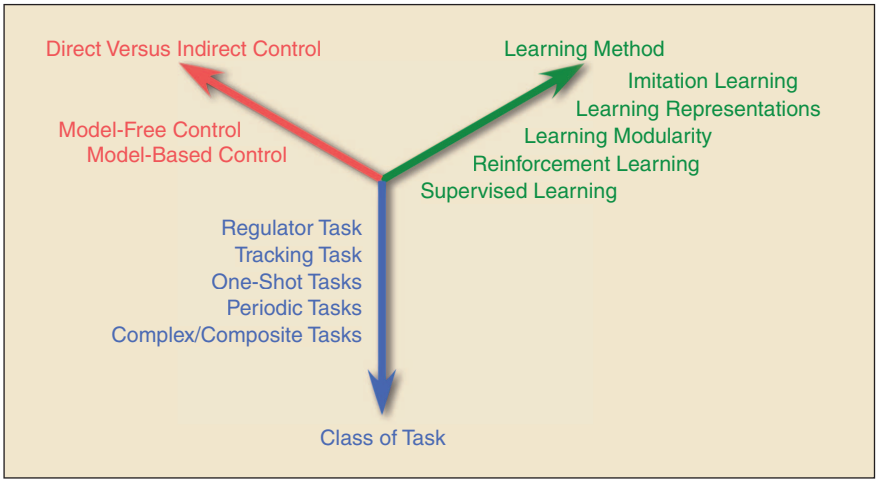

Figure 1. *Classification of robot learning along three dimensions. Topics further out on the arrows can be considered more complex research topics than topics closer to the center.*

cases pose a problem to learning methods and can be addressed with special techniques and representations [13]–[16].

Nonlinear function approximation is needed to learn internal models. It should be noted, as will be explained later, that function approximation is also required for other robot learning problems, e.g., to represent value functions, reward functions, or policies in reinforcement learning—thus, function approximation has a wide applicability in robot learning. While most machine-learning problems in function approximation work by processing a given data set in an offline fashion, robot learning has several features that require specialized algorithms:

- data are available in abundance, typically at a rate from 60 to 1,000 data points per second
- given this continuous stream of data, learning should never stop, but continue forever without degradation over time. For instance, degradation happens in many algorithms if the same data point is given to the learning system repeatedly, e.g., when the robot is standing still
- given the high dimensionality of most interesting robotic systems, the complexity of the function to be learned is often unknown in advance, and the function approximation system needs to be able to add new learning resources as learning proceeds
- learning should happen in real time, be data efficient (squeeze the most information out of each data point), and be computationally efficient (to achieve real-time learning and lookup)
- learning needs to be robust toward shifting input distributions, e.g., as typical when practicing calligraphy on one day and tennis on another day, a topic discussed in the context of catastrophic interference [17]
- learning needs to be able to detect relevant features in the input from ideally hundreds or thousands of input dimensions, and it needs to exclude automatically irrelevant and redundant inputs.

These requirements narrow down the learning algorithms that are applicable to function approximation for robot learning. One approach that has favorable performance is learning with piecewise linear models using nonparametric regression techniques [17]–[22]. Essentially, this technique finds, in the spirit of a first-order Taylor series expansion, the linearization of the function at an input point, and the region (also called a kernel) in which this linearization holds within a certain error bound. Learning this region is the most complex part of these techniques, and the latest developments use Bayesian statistics [23] and dimensionality reduction [22].

A new development, largely due to increasingly faster computing hardware, is the application of Gaussian process regression (GPR) to function approximation in robots [24]–[26]. GPR is a powerful function approximation tool that has gained popularity due to its sound theory, high fitting accuracy, and the relative ease of application with public-domain software libraries. As it requires an iterative optimization that needs to invert a matrix of size $N \times N$, where N is the number of training data points, GPR quickly saturates the computational resources with moderately many data points. Thus, scalability to continual and real-time learning in complex robots will require further research developments; some research along these lines is given in [25] and [27].

Example Application

As mentioned earlier, learning inverse models can be challenging, since the inverse model problem is often a relation and not a function, with a one-to-many mapping. Applying any arbitrary nonlinear function approximation method to the inverse model problem can lead to unpredictably bad performance, as the training data can form nonconvex solution spaces in which averaging is inappropriate [12]. A particularly interesting approach in control involves learning local linearizations of a forward model (which is a proper function) and learning an inverse mapping within the local region of the forward model; see also [15] and [28].

Ting et al. [23] demonstrated such a forward-inverse model learning approach with Bayesian locally weighted regression (BLWR) to learn an inverse kinematics model for a haptic robot arm (Figure 2) for a task-space tracking task. Training data consisted of the arm's joint angles $\mathbf{q}$, joint velocities $\dot{\mathbf{q}}$, end-effector position in Cartesian space $\mathbf{y}$, and end-effector velocities $\dot{\mathbf{y}}$. From this data, a differential forward kinematics model $\dot{\mathbf{y}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}$ was learned, where $\mathbf{J}$ is the Jacobian matrix. The transformation from $\dot{\mathbf{q}}$ to $\dot{\mathbf{y}}$ can be assumed to be locally linear at a particular configuration $\mathbf{q}$ of the robot arm. BLWR is used to learn the forward model in a piecewise linear fashion.

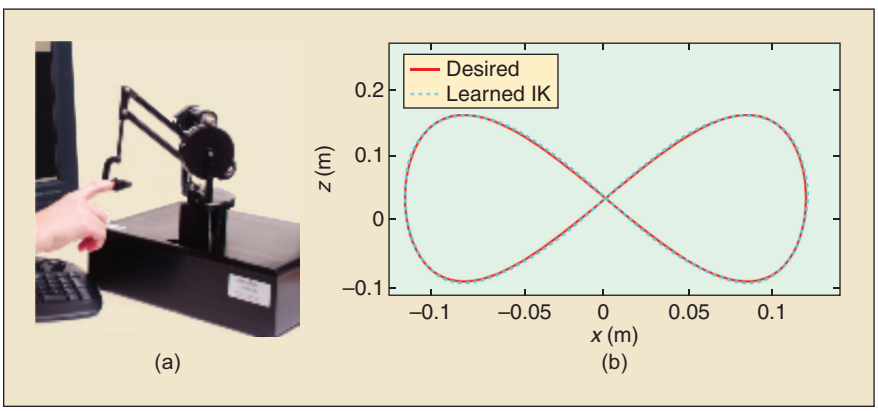

Figure 2. *(a) Phantom robot. (b) Learned-inverse kinematics solution; the difference between the actual and desired trajectory is small.*

The goal of the robot task is to track a desired trajectory $(\mathbf{y}, \dot{\mathbf{y}})$ specified only in terms of x, z Cartesian positions and velocities, i.e., the movement is supposed to be in a vertical plane in front of the robot, but the exact position of the vertical plane is not given. Thus, the task has one degree of redundancy. To learn an inverse kinematics model, the local regions from the piecewise linear forward model can be reused since any local inverse is also locally linear within these regions. Moreover, for locally linear models, all solution spaces for the inverse

model are locally convex, such that an inverse can be learned without problems. The redundancy issue can be solved by applying an additional weight to each data point according to a reward function, resulting in reward-weighted locally weighted regression [15].

Figure 2 shows the performance of the learned inverse model (Learned IK) in a figure-eight tracking task. The learned model as well as the analytical inverse kinematics solution performs with root-mean-squared tracking errors in positions and velocities very close to that of the analytical solution. This performance was acquired from five minutes of real-time training data.

Model-Based Learning

In considering model-based learning, it is useful to start by assuming that the model is perfect. Later, we will address the question of how to design a controller that is robust to flaws in the learned model.

Conventional Dynamic Programming

Designing controllers for linear models is well understood. Work in reinforcement learning has focused using techniques derived from dynamic programming to design controllers for models that are nonlinear. A large part of our own work has emphasized pushing back the curse of dimensionality, as the memory and computational cost of dynamic programming increase exponentially with the dimensionality of the state-action space.

Dynamic programming provides a way to find globally optimal control policies when the model of the control system is known. This section focuses on offline planning of nonlinear control policies for control problems with continuous states and actions, deterministic time invariant discrete time dynamics, $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$, and a time-invariant one-step cost or reward function $L(\mathbf{x},\mathbf{u})$—equivalent formulations exist for continuous time systems [29]–[31]. We are addressing steady-state policies, i.e., policies that are not time variant and have an infinite time horizon. One approach to dynamic programming is to approximate the value function $V(\mathbf{x})$ (the optimal total future cost from each state $V(\mathbf{x}) = \min_{\mathbf{u_k}} \sum_{k=0}^{\infty} L(\mathbf{x_k}, \mathbf{u_k})$) by repeatedly solving the Bellman equation $V(\mathbf{x}) = \min_{\mathbf{u}} \{L(\mathbf{x}, \mathbf{u}) + V(\mathbf{f}(\mathbf{x}, \mathbf{u}))\}$ at sampled states $\mathbf{x}$ until the value function estimates have converged to globally optimal values. Typically, the value function and control law are represented on a regular grid—it should be noted that more efficient adaptive grid methods [32], [33] or function approximation methods [7] also exist. Some type of interpolation is used to approximate these functions within each grid cell. If each dimension of the state and action is represented with a resolution R, and the dimensionality of the state is d_x and that of the action is d_u, the computational cost of the conventional approach is proportional to $R^{d_x} \times R^{d_u}$ and the memory cost is proportional to R^{d_x}. This is known as the curse of dimensionality [9].

We have shown that dynamic programming can be sped up by randomly sampling actions on each sweep rather than exhaustively minimizing the Bellman equation with respect to the action [34]. At each state on each update, the current best action is reevaluated and compared to some number of random actions. Our studies have found that only looking at one random action on each update is most efficient. It is more effective to propagate information about future values by reevaluating the current best action on each update than it is to put a lot of resources into searching for the absolute best action.

With this speedup in action search, currently available cluster computers can easily handle ten-dimensional problems (approximately 10^{10} points can handle grids of size 50^6, 20^8, or 10^{10}, for example). Current supercomputers are created by networking hundreds or thousands of conventional computers. The obvious way to implement dynamic programming on such a cluster is to partition the grid representing the value function and policy across the individual computing nodes, with the borders shared between multiple nodes. When a border cell is updated by its host node, the new value must be communicated to all nodes that have copies of that cell. We have implemented dynamic programming in a cluster of up to 100 nodes, with each node having eight CPU cores and 16 GB of memory. For example, running a cluster of 40 nodes on a six-dimensional problem with 50^6 cells, about 6 GB is used on each node to store its region of the value function and policy.

Decomposing Problems

One way to reduce the curse of dimensionality is to break problems into parts and develop a controller for each part separately. Each subsystem could be ten-dimensional, given the earlier results, and a system that combined two subsystems could be 20 dimensional. For example, we are interested in developing a controller for biped walking [35]. We can approximately model the dynamics of a biped with separate models for sagittal and lateral control. These models are linked by common actions, such as when to put down and lift the feet. Thus, there are two parts of the state vector $\mathbf{x}$: variables that are part of the sagittal state $\mathbf{x}_s$ and variables that are part of the lateral state $\mathbf{x}_l$. There are three parts of the action vector $\mathbf{u}$: variables that are part of the sagittal action $\mathbf{u}_s$, variables that are part of the lateral action $\mathbf{u}_l$, and variables that affect both systems $\mathbf{u}_{sl}$. We can perform dynamic programming on the sagittal system and produce a value function $V_s(\mathbf{x}_s)$ and do the same with the lateral system $V_l(\mathbf{x}_l)$. We can choose an optimal action by minimizing $L((\mathbf{x},\mathbf{u}) + V(\mathbf{f}(\mathbf{x},\mathbf{u}))$ with respect to $\mathbf{u}$, with $V(\mathbf{x})$ approximated by $V_s(\mathbf{x}_s) + V_l(\mathbf{x}_l)$. This approximation ignores the linking of the two systems in the future and can be improved by adding elements to the one-step costs for each subsystem that bias the shared actions to behave as if the other system was present. For example, deviations from the timing usually seen in the complete system can be penalized.

Trajectory Optimization and Trajectory Libraries

Another way to handle complex systems is trajectory optimization. Given a model, a variety of approaches can be used to find a locally optimal sequence of commands for a given initial position and one-step cost [36]–[38]. Interestingly, trajectory optimization is quite popular for generating motion in animation [39]. However, trajectory optimization is not so popular in robotics, because it appears that it does not produce a control law but just a fixed sequence of commands. This is not a correct view.

To generate a control policy, trajectory optimization can be applied to many initial conditions, and the resulting commands can be interpolated as needed. If that is the case, why do we need to deal with dynamic programming and the curse of dimensionality? Dynamic programming is a global optimizer, while trajectory optimization finds local optima. Often, the local optima found are not acceptable. Some way to bias trajectory optimization to produce reasonable trajectories would be useful. Also, if interpolation of the results will be done, it would be useful to produce consistent results so that similar initial conditions lead to similar costs. There may be discontinuities between nearby trajectories that must be handled by interpolation of actions.

One trick to improve trajectories is to use neighboring trajectories to somehow bias or guide the optimization process. A simple way to do this is to use a neighboring trajectory as the initial trajectory in the trajectory-optimization process. Trajectories can be reoptimized using each neighbor in turn as the initial trajectory, and the best result so far can be retained. We have explored building explicit libraries of optimized trajectories to handle large perturbations in bipedal standing balance [40]. One way of using the library is to use the optimized action corresponding to the nearest state in the library at each time step. Another way is to store the derivative of the optimized action with respect to state and use that derivative to modify the suggested action. A third way is to look up states from multiple trajectories and generate a weighted blend of the suggested actions.

The first and second derivatives of a trajectory's cost with respect to state can be used to generate a local Taylor series model of the value function: $V(\mathbf{x}) = V_0 + V_x\mathbf{x} + \mathbf{X}^{\mathbf{T}}V_{xx}\mathbf{X}$. Given a quadratic local model of the value function, it is possible to compute the optimal action and its first derivative, the feedback gains. These observations led to a trajectory optimization method based on second-order gradient descent, differential dynamic programming (DDP) [29]. Although this trajectory optimization method is no longer considered the most efficient way to find an optimal trajectory [sequential quadratic programming (SQP) methods are currently preferred in many fields such as aerospace and animation], the local models of the value function and policy that DDP produces are useful for machine learning. For example, the local model of the policy can be used in a trajectory library to interpolate or extrapolate actions. Discrepancies in adjacent local models of the value function can be used to determine where to allocate additional library resources.

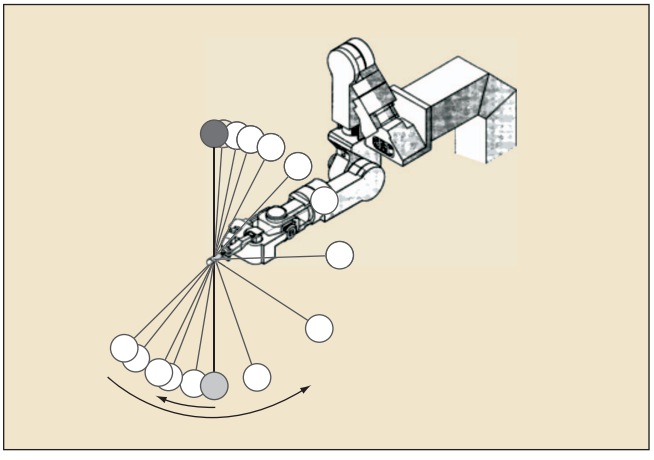

Figure 3. *The robot swinging up an inverted pendulum.*

Robustness

Robustness has not been addressed well in robot learning. Studies often focus on robustness to additive noise. It is much more difficult to design controllers that are robust to the correlated errors caused by parameter error or model structure error. One approach to designing robust controllers is to optimize controller parameters by simulating a controller controlling a noisy robot [41]. It is more useful to optimize controller parameters controlling a set of robots, each with different robot parameters. This allows the effect of correlated controller errors across time to be handled in the optimization.

It is not clear how to perform a similar optimization over a set of models in dynamic programming. Using additive noise and performing stochastic dynamic programming does not capture the effect of correlated errors. One approach is to make the model parameters into model states and perform stochastic dynamic programming on information states that describe distributions of actual states and model parameters. However, this creates a large increase in the number of states, which is not practical for dynamic programming.

Bar-Shalom and Tse showed that DDP can be used to locally optimize controller robustness as well as exploration [42], [43]. This work provides an efficient solution to optimize the typically high-dimensional information state, which includes the means and covariances of the original model states and the means and covariances of the model parameters. Representing the uncertainty using a parametric probability distribution (means and covariances) also reduces the computational cost of propagating uncertainty forward in time. The dynamics of the system are given by an extended Kalman filter. The key observation is that the cost of uncertainty (the state and model parameter covariances) is given by $\mathrm{Trace}(V_{xx}\Sigma)$, the trace of the product of the second derivative of the value function and the covariance matrix of the state. Minimizing the additional cost due to uncertainty makes the controller more robust and guides exploration.

Example Application

We implemented DDP on an actual robot as part of a learning from demonstration experiment (Figure 3). Several robustness issues arose since models are never perfect, especially learned models. 1) We needed initial trajectories that were consistent with the learned models, and sometimes reasonable or feasible trajectories do not exist due to modeling error in the learned model. 2) During optimization, the forward integration of a learned model in time often blows up when the learned model is inaccurate or when the plant is unstable and the current policy fails to stabilize it. 3) The backward integration to produce a value function and a corresponding policy uses derivatives of the learned model, which are often quite inaccurate in the early stages of learning, producing inaccurate value function estimates and ineffective policies. 4) Dynamic planners amplify modeling

error, because they take advantage of any modeling error that reduces cost, and because some planners use derivatives, which can be quite inaccurate. 5) The new knowledge gained in attempting a task may not change the predictions the system makes about the task (falling down might not tell us much about the forces needed in walking). In the task shown in Figure 3, we used a direct reinforcement learning approach that adjusted the task goals in addition to optimal control to overcome modeling errors that the learning system did not handle [44].

We use another form of one-link pendulum swing-up as an example problem to provide the reader with a visualizable example of a value function and policy (Figure 4). In this one-link pendulum swing-up, a motor at the base of the pendulum swings a rigid arm from the downward stable equilibrium to the upright unstable equilibrium and balances the arm there. What makes this challenging is that the one-step cost function penalizes the amount of torque used and the deviation of the current position from the goal. The controller must try to minimize the total cost of the trajectory. The one-step cost function for this example is a weighted sum of the squared position errors ($\tilde{\theta}$: difference between current angle and the goal angle) and the squared torques τ: $L(\mathbf{x}, \mathbf{u}) = 0.1\,\tilde{\theta}^2 T + \tau^2 T$, where 0.1 weights the position error relative to the torque penalty and T is the time step of the simulation (0.01s). Including the time step T in the optimization criterion allows comparison with controllers with different time steps and continuous time controllers. There are no costs associated with the joint velocity. Figure 4 shows the optimal value function and policy. The optimal trajectory is shown as a yellow line in the value function plot and as a black line with a yellow border in the policy plot [Figure 4(b) and (c)]. The value function is cut off above 20 so that we can see the details of the part of the value function that determines the optimal trajectory. The goal is at the state (0,0).

Model-Free Learning

There are several popular methods of approaching model-free robot learning. Value function-based methods are discussed in the context of actor-critic methods, temporal difference (TD) learning, and Q-learning. A novel wave of algorithms avoids value functions and focuses on directly learning the policy, either with gradient methods or probabilistic methods.

Value Function Approaches

Instead of using dynamic programming, the value function $V(\mathbf{x})$ can be estimated with TD learning [7], [45]. Essentially, TD enforces the validity of the Bellman equations for temporally adjacent states, which can be shown to lead to a spatially consistent estimate of the value function for a given policy. To improve the policy, TD needs to be coupled to a simultaneous policy update using actor-critic methods [7].

Alternatively, instead of the value function $V(\mathbf{x})$, the action value function $Q(\mathbf{x},\mathbf{u})$ can be used, which is defined as $Q(\mathbf{x}, \mathbf{u}) = L(\mathbf{x}_0, \mathbf{u}_0) + \min_{\mathbf{u_k}} \sum_{k=1}^{\infty} L(\mathbf{x_k}, \mathbf{u_k})$ [7], [46]. Knowing $Q(\mathbf{x},\mathbf{u})$ for all actions in a state allows choosing the one with the maximal (or minimal for penalty costs) Q-value as the optimal action. Q-learning can be conceived of as TD learning in the joint space of states and actions.

TD and Q-learning work well for discrete state-action spaces but become more problematic in continuous state-action scenarios. In continuous spaces, function approximators need to be used to represent the value function and policy. Achieving reliable estimation of these functions usually requires a large number of samples that densely fill the relevant space for learning, which is hard to accomplish in actual experiments with complex robot systems. There are also no guarantees that, during learning, the robot will not be given unsafe commands. Thus, many practical approaches learn first

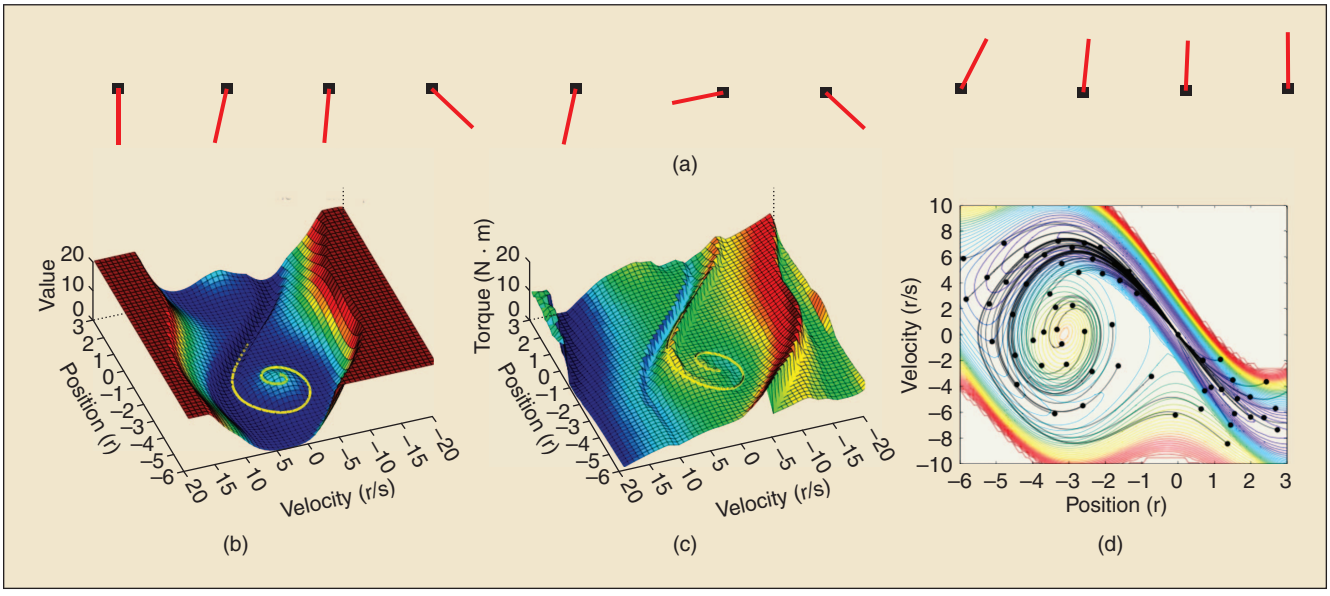

Figure 4. *(a) Configurations from the simulated one link pendulum optimal trajectory every half second and at the end of the trajectory. (b) Value function for one-link example. (c) Policy for one-link example. (d) Trajectory-based approach: random states (dots) and trajectories (black lines) used to plan one-link swing-up, superimposed on a contour map of the value function [33].*

in simulations (which is essentially a model-based approach) until reasonable performance is achieved, before continuing to experiment on an actual robot to adjust the control policy to the true physics of the world [47].

In the end, it is intractable to find a globally optimal control policy in high dimensional robot systems, as global optimality requires exploration of the entire state-action space. Thus, local optimization such as trajectory optimization seems to be more practical, using initialization of the policy from some informed guess, for instance, imitation learning [44], [48]–[51]. Fitted Q-iteration is an example of a model-free learning algorithm that approximates the Q-function only along some sampled trajectories [52], [53]. Recent developments have given up on estimating the value function and rather focus directly on learning the control policy from trajectory rollouts, which is the topic of the following sections.

Policy Gradient Methods

Policy gradient methods usually assume that the cost of motor skill can be written as

$$J(\mathbf{x}_0) = E_\tau\left\{\sum_{k=0}^{N}\gamma^k L(\mathbf{x}_k, \mathbf{u}_k)\right\}, \tag{4}$$

which is the expected sum of discounted rewards ($\gamma \in [0,1]$) over a (potentially infinite) time horizon N. The expectation $E\{\}$ is taken over all trajectories τ that start in state $\mathbf{x}_0$. The goal is to find the motor commands $\mathbf{u}_k$ that optimize this cost function. Most approaches assume that there is a start state $\mathbf{x} = \mathbf{x}_0$ and/or a start state distribution [54]. The control policy is also often compactly parameterized, e.g., by means of a basis function representation $\mathbf{u} = \theta^T\phi(\mathbf{x})$, where θ are the policy parameters [see also (1)], and $\phi(\mathbf{x})$ is a vector of nonlinear basis functions provided by the user. Mainly for the purpose of exploration, the policy can be chosen to be stochastic, e.g., with a normal distribution $\mathbf{u} \sim N(\theta^T\phi(\mathbf{x}), \Sigma)$, although cases exist where only a stochastic policy is optimal [54].

The essence of policy gradient methods is to compute the gradient $\partial J/\partial\theta$ and optimize (4) with gradient-based incremental updates. As discussed in more detail in [55], a variety of algorithms exist to compute the gradient. Finite difference gradients [56] perform a perturbation analysis of the parameter vector θ and estimate the gradient from a first-order numerical Taylor series expansion. The REINFORCE algorithm [57], [58] is a straightforward derivative computation of the logarithm of (4), assuming as the probability of a trajectory $p_\theta(\tau) = p(\mathbf{x}_0)\prod_{k=1}^{N} p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_{k-1})p_\theta(\mathbf{u}_{k-1}|\mathbf{x}_{k-1})$, and emphasizing that the parameters θ only appear in the stochastic policy p_θ such that many terms in the gradient computation drop out. GPOMDP [59] and methods based on the policy gradient theorem [54] are more efficient versions of REINFORCE (for more details, see [55]). Peters and Schaal [60] suggested a second-order gradient method derived from insights of [61] and [62], which is currently among the fastest gradient-learning approaches. Reference [63] emphasized that the choice of injecting noise in the stochastic policy can strongly influence the efficiency of the gradient updates.

Policy gradient methods can scale to high-dimensional state-action spaces, at the cost of finding only locally optimal control policies and have become rather popular in robotics [64]–[66]. One drawback of policy gradients is that they require manual tuning of gradient parameters, which can be tedious. Probabilistic methods, as discussed in the next section, try eliminating gradient computations.

Probabilistic Direct Policy Learning

Transforming reinforcement learning into a probabilistic estimation approach is inspired by the hope of bringing to bear the wealth of statistical learning techniques that were developed over the last 20 years of machine-learning research. An early attempt can be found in [67], where reinforcement learning was formulated as an expectation–maximization (EM) algorithm [68]. The important idea was to treat the reward $L(\mathbf{x},\mathbf{u})$ as a pseudoprobability, i.e., it has to be strictly positive, and the integral over the state-action space of the reward has to result in a finite number. Transforming traditional convex reward functions with the exponential function is often used to achieve this property at the cost that the learning problem gets slightly altered by this change of cost function. Equation (4) can thus be thought of as a likelihood, and the corresponding log likelihood becomes

$$\log J(\mathbf{x}) = \log\int_\tau p_\theta(\tau)R(\tau)d\tau, \text{ where } R(\tau) = \sum_{k=0}^{N}\gamma^k L(\mathbf{x}_k, \mathbf{u}_k). \tag{5}$$

This log likelihood can be optimized with the EM algorithm. In [15], such an approach was used to learn operational space controllers, where the reinforcement learning component enabled a consistent resolution of redundancy. In [69], the previous approach was extended to learning from trajectories—see also contribution by Kober and Peters (pp. 55–62). Extending [70] and [71] added a more thorough treatment of learning in the infinite discounted horizon case, where the algorithm can essentially determine the most suitable temporal window for optimization.

Another way of transforming reinforcement learning into a statistical estimation problem was suggested in [72] and [73]. Here, it was realized that optimization with the stochastic Hamilton-Jacobi-Bellman equations can be transformed into a path-integral estimation problem, which can be derived with the Feynman-Kac theorem [31], [74]. While this formulation is normally based on value functions and requires a model-based approach, Theodorou et al. [31] realized that even model-free methods can be obtained. The resulting reinforcement learning algorithm resembles the one of [69], however, without the requirement that reinforcement is a pseudoprobability. Because of its grounding in first-order principles of optimal control theory, its simplicity, and no open learning parameters except for the exploration noise, this algorithm might be one of the most straightforward methods of trajectory-based reinforcement learning to date. It should also be

mentioned that [75] developed a model-based reinforcement learning framework with a special probabilistic control cost for discrete state-action spaces that, in its limit to continuous state-action spaces, will result in a path-integral formulation.

Example Application

Figure 5 illustrates our application of path-integral reinforcement learning to a robot-learning problem [31]. The robot dog is to jump across a gap. The jump should make as much forward progress as possible, as it is a maneuver in a legged locomotion competition, which scores the speed of the robot. The robot has three degree of freedoms (DoFs) per leg, and thus a total of 12 DoFs. Each DoF was represented as a parameterized movement primitive [76] with 50 basis functions. An initial seed behavior was taught by learning from demonstration, which allowed the robot barely to reach the other side of the gap without falling into the gap—the demonstration was generated from a manual adjustment of knot points in a spline-based trajectory plan for each leg.

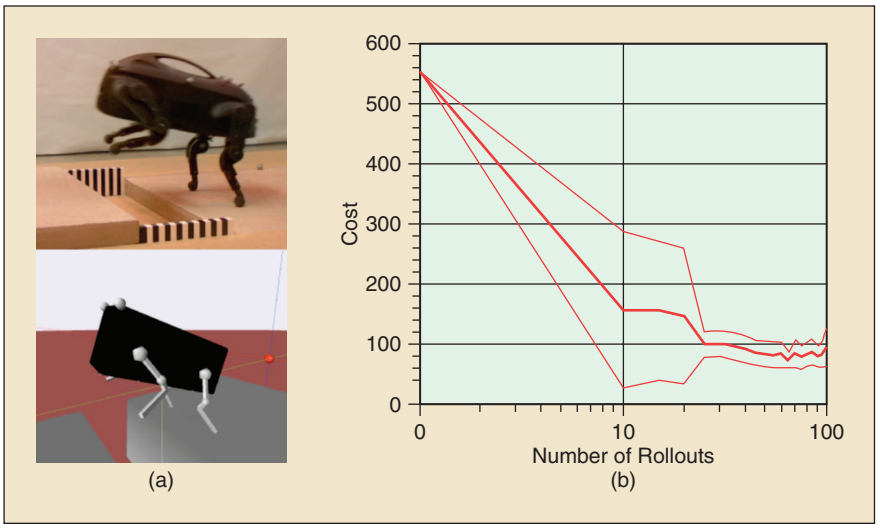

Figure 5. *(a) Actual and simulated robot dog. (b) Learning curve of optimizing the jump behavior with path-integral reinforcement learning.*

Path-integral reinforcement learning primarily used the forward progress as a reward and slightly penalized the squared acceleration of each DoF and the squared norm of the parameter vector, i.e., a typical form of complexity regularization [77]. Learning was performed on a physical simulator of the robot dog, as the real robot dog was not available for this experiment. Figure 5 illustrates that after about 30 trials, the performance of the robot was significantly improved, such that after the jump, almost the entire body was lying on the other side of the gap. It should be noted that applying path-integral reinforcement learning was algorithmically very simple, and manual tuning only focused on generate a good cost function.

Imitation Learning, Policy Parameterizations, and Inverse Reinforcement Learning

While space constraints will not allow us to go into more detail, three interwoven topics in robot learning are worth mentioning.

First, imitation learning has become a popular topic to initialize and speed up robot learning. Reviews on this topic can be found, for instance, in [48], [49], and [78].

Second, determining useful parameterizations for control policies is a topic that is often discussed in conjunction with imitation learning. Many different approaches have been suggested in the literature, for instance, based on splines [79], hidden Markov models [80], nonlinear attractor systems [76], and other methods. Billard et al. [78] provide a survey of this topic.

Finally, designing useful reward functions remains one of the most time-consuming and frustrating topics in robot learning. Thus, extracting the reward function from observed behavior is a topic of great importance for robot learning and imitation learning under the assumption that the observed behavior is optimal under a certain criterion. Inverse reinforcement learning [81], apprenticeship learning [82], and maximum margin planning [83] are some of the prominent examples in the literature.

Conclusions

Recent trends in robot learning are to use trajectory-based optimal control techniques and reinforcement learning to scale complex robotic systems. On the one hand, increased computational power and multiprocessing, and on the other hand, probabilistic reinforcement learning methods and function approximation, have contributed to a steadily increasing interest in robot learning. Imitation learning has helped significantly to start learning with reasonable initial behavior. However, many applications are still restricted to rather low-dimensional domains and toy applications. Future work will have to demonstrate the continual and autonomous learning abilities, which were alluded to in the introduction.

Acknowledgments

This research was supported in part by National Science Foundation grants ECS-0326095, EEC-0540865, and ECCS-0824077, IIS-0535282, CNS-0619937, IIS-0917318, CBET-0922784, EECS-0926052, the DARPA program on Learning Locomotion, the Okawa Foundation, and the ATR Computational Neuroscience Laboratories.

Keywords

Robot learning, learning control, reinforcement learning, optimal control.

References

[1] S. Schaal, "The new robotics—Towards human-centered machines," *HFSP J. Frontiers Interdisciplinary Res. Life Sci.*, vol. 1, no. 2, pp. 115–126, 2007.

[2] K. J. Åström and B. Wittenmark, *Adaptive Control*. Reading, MA: Addison-Wesley, 1989.

[3] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.

[4] M. Buehler, *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, 1st ed. New York: Springer-Verlag, 2009.
[5] M. Buehler, K. Iagnemma, and S. Singh, *The 2005 DARPA Grand Challenge: The Great Robot Race*. New York: Springer-Verlag, 2007.
[6] M. Roy, G. Gordon, and S. Thrun, "Finding approximate POMDP solutions through belief compression," *J. Artif. Intell. Res.*, vol. 23, pp. 1–40, 2005.
[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.
[8] J. Si, *Handbook of Learning and Approximate Dynamic Programming*. Hoboken, NJ: IEEE Press/Wiley-Interscience, 2004.
[9] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.
[10] P. Dyer and S. R. McReynolds, *The Computation and Theory of Optimal Control*. New York: Academic, 1970.
[11] L. Sciavicco and B. Siciliano, *Modelling and Control of Robot Manipulators*. New York: Springer-Verlag, 2000.
[12] I. M. Jordan, D. E. Rumelhart, "Supervised learning with a distal teacher," *Cogn. Sci.*, vol. 16, pp. 307–354, 1992.
[13] A. D'Souza, S. Vijayakumar, and S. Schaal, "Learning inverse kinematics," in *Proc. IEEE Int. Conf. Intelligent Robots and Systems (IROS 2001)*, Maui, HI, Oct. 29–Nov. 3, 2001, pp. 298–301.
[14] D. Bullock, S. Grossberg, and F. H. Guenther, "A self-organizing neural model of motor equivalent reaching and tool use by a multijoint arm," *J. Cogn. Neurosci.*, vol. 5, no. 4, pp. 408–435, 1993.
[15] J. Peters and S. Schaal, "Learning to control in operational space," *Int. J. Robot. Res.*, vol. 27, pp. 197–212, 2008.
[16] Z. Ghahramani and M. I. Jordan, "Supervised learning from incomplete data via an EM approach," in *Advances in Neural Information Processing Systems 6*, J. D. Cowan, G. Tesauro, and J. Alspector, Eds. San Mateo, CA: Morgan Kaufmann, 1994, pp. 120–127.
[17] S. Schaal and C. G. Atkeson, "Constructive incremental learning from only local information," *Neural Comput.*, vol. 10, no. 8, pp. 2047–2084, 1998.
[18] W. S. Cleveland, "Robust locally weighted regression and smoothing scatterplots," *J. Amer. Statist. Assoc.*, vol. 74, pp. 829–836, 1979.
[19] C. G. Atkeson, "Using local models to control movement," in *Advances in Neural Information Processing Systems 1*, D. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 157–183.
[20] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artif. Intell. Rev.*, vol. 11, no. 1–5, pp. 11–73, 1997.
[21] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning for control," *Artif. Intell. Rev.*, vol. 11, no. 1–5, pp. 75–113, 1997.
[22] S. Vijayakumar, A. D'Souza, and S. Schaal, "Incremental online learning in high dimensions," *Neural Comput.*, vol. 17, no. 12, pp. 2602–2634, 2005.
[23] J.-A. Ting, A. D'Souza, S. Vijayakumar, and S. Schaal, "A Bayesian approach to empirical local linearizations for robotics," in *Proc. Int. Conf. Robotics and Automation (ICRA2008)*, Pasadena, CA, May 19–23, 2008, pp. 2860–2865.
[24] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA: MIT Press, 2006.
[25] D. Nguyen-Tuong, M. Seeger, and J. Peters, "Local gaussian process regression for real time online model learning and control," in *Proc. Advances in Neural Information Processing Systems 21 (NIPS 2008)*, D. Schuurmans, J. Benigio, and D. Koller, Eds. Vancouver, BC, Dec. 8–11, 2009, pp. 1193–1200.
[26] M. P. Deisenroth, C. E. Rasmussen, and J. Peters, "Gaussian process dynamic programming," *Neurocomputing*, vol. 72, no. 7–9, pp. 1508–1524, 2009.
[27] L. Csat'o and M. Opper, "Sparse representation for gaussian process models," in *Proc. Advances in Neural Information Processing Systems 13 (NIPS 2000)*, Denver, CO, 2001, pp. 444–450.
[28] D. M. Wolpert and M. Kawato, "Multiple paired forward and inverse models for motor control," *Neural Netw.*, vol. 11, no. 7–8, pp. 1317–1329, 1998.
[29] D. H. Jacobson and D. Q. Mayne, *Differential Dynamic Programming*. New York: American Elsevier, 1970.
[30] K. Doya, "Reinforcement learning in continuous time and space," *Neural Comput.*, vol. 12, no. 1, pp. 219–245, Jan. 2000.
[31] E. Theodorou, J. Buchli, and S. Schaal, "Reinforcement learning in high dimensional state spaces: A path integral approach," submitted for publication.
[32] R. Munos and A. Moore, "Variable resolution discretization in optimal control," *Mach. Learn.*, vol. 49, no. 2/3, p. 33, 2002.
[33] C. G. Atkeson and B. J. Stephens, "Random sampling of states in dynamic programming," *IEEE Trans. Syst., Man, Cybern. B*, vol. 38, no. 4, pp. 924–929, 2008.
[34] C. G. Atkeson, "Randomly sampling actions in dynamic programming," in *Proc. IEEE Int. Symp. Approximate Dynamic Programming and Reinforcement Learning, 2007, ADPRL'07*, pp. 185–192.
[35] E. Whitman and C. G. Atkeson, "Control of a walking biped using a combination of simple policies," in *Proc. IEEE/RAS Int. Conf. Humanoid Robotics*, Paris, France, Dec. 7–10, 2009, pp. 520–527.
[36] Tomlab Optimization Inc. (2010). PROPT—Matlab optimal control software [Online]. Available: http://tomdyn.com/
[37] Technische Universität Darmstadt. (2010). DIRCOL: A direct collocation method for the numerical solution of optimal control problems [Online]. Available: http://www.sim.informatik.tu-darmstadt.de/sw/dircol
[38] Stanford Business Software Corporation. (2010). SNOPT; Software for large-scale nonlinear programming [Online]. Available: http://www.sbsi-sol-optimize.com/asp/sol_product_snopt.htm
[39] A. Safonova, J. K. Hodgins, and N. S. Pollard, "Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces," *ACM Trans. Graph. J. (SIGGRAPH 2004 Proc.)*, vol. 23, no. 3, pp. 514–521, 2004.
[40] L. Chenggang and C. G. Atkeson, "Standing balance control using a trajectory library," presented at the IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS 2009), 2009.
[41] A. Ng, "Pegasus: A policy search method for large MDPs and POMDPs," presented at the Uncertainty in Artificial Intelligence (UAI), 2000.
[42] E. Tse, Y. Bar-Shalom, and L. Meier, III, "Wide-sense adaptive dual control for nonlinear stochastic systems," *IEEE Trans. Automat. Contr.*, vol. 18, no. 2, pp. 98–108, 1973.
[43] Y. Bar-Shalom and E. Tse, "Caution, probing and the value of information in the control of uncertain systems," *Ann. Econ. Social Meas.*, vol. 4, no. 3, pp. 323–338, 1976.
[44] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *Proc. 14th Int. Conf. Machine Learning (ICML'97)*, D. H. Fisher, Jr., Ed. Nashville, TN, July 8–12, 1997, pp. 12–20.
[45] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, no. 1, pp. 9–44, 1988.
[46] C. J. C. H. Watkins, "Learning with delayed rewards," Ph.D. thesis, Cambridge Univ., U.K., 1989.
[47] J. Morimoto and K. Doya, "Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning," *Robot. Auton. Syst.*, vol. 36, no. 1, pp. 37–51, 2001.
[48] S. Schaal, "Is imitation learning the route to humanoid robots?" *Trends Cogn. Sci.*, vol. 3, no. 6, pp. 233–242, 1999.
[49] S. Schaal, A. Ijspeert, and A. Billard, "Computational approaches to motor learning by imitation," *Philos. Trans. R. Soc. London B, Biol. Sci.*, vol. 358, no. 1431, pp. 537–547, 2003.
[50] C. G. Atkeson and S. Schaal, "Learning tasks from a single demonstration," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA'97)*, Albuquerque, NM, Apr. 20–25, 1997, pp. 1706–1712.
[51] S. Schaal, "Learning from demonstration," in *Proc. Advances in Neural Information Processing Systems 9*, M. C. Mozer, M. Jordan, and T. Petsche, Eds. Cambridge, MA, 1997, pp. 1040–1046.
[52] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *J. Mach. Learn. Res.*, vol. 6, pp. 503–556, 2005.
[53] G. Neumann and J. Peters, "Fitted Q-iteration by advantage weighted regression," in *Proc. Advances in Neural Information Processing Systems 21 (NIPS 2008)*, D. Schuurmans, J. Benigio, and D. Koller, Eds. Vancouver, BC, Dec. 8–11, 2009, pp. 1177–1184.

[54] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Advances in Neural Processing Systems 12*, S. A. Solla, T. K. Leen, and K.-R. Müller, Eds. Denver, CO, 2000.

[55] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Netw.*, vol. 21, no. 4, pp. 682–697, May 2008.

[56] P. Sadegh and J. Spall, "Optimal random perturbations for stochastic approximation using a simultaneous perturbation gradient approximation," presented at the *Proc. American Control Conf.*, 1997.

[57] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 3–4, pp. 229–256, 1992.

[58] V. Gullapalli, "A stochastic reinforcement learning algorithm for learning real-valued functions," *Neural Netw.*, vol. 3, no. 6, pp. 671–692, 1990.

[59] D. Aberdeen and J. Baxter, "Scaling internal-state policy-gradient methods for POMDPs," in *Proc. 19th Int. Conf. Machine Learning (ICML-2002)*, Sydney, Australia, 2002, pp. 3–10.

[60] J. Peters and S. Schaal, "Natural actor critic," *Neurocomputing*, vol. 71, no. 7–9, pp. 1180–1190, 2008.

[61] S. Amari, "Natural gradient learning for over- and under-complete bases In ICA," *Neural Comput.*, vol. 11, no. 8, pp. 1875–1883, Nov. 1999.

[62] S. Kakade, "Natural policy gradient," presented at the Advances in Neural Information Processing Systems, Vancouver, CA, 2002.

[63] T. Rückstieß, M. Felder, and J. Schmidhuber, "State-dependent exploration for policy gradient methods," presented at the European Conf. Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2008, Part II, LNAI 5212, 2008.

[64] G. Endo, J. Morimoto, T. Matsubara, J. Nakanish, and G. Cheng, "Learning CPG-based biped locomotion with a policy gradient method: Application to a humanoid robot," *Int. J. Robot. Res.*, vol. 27, no. 2, pp. 213–228, 2008.

[65] R. Tedrake, T. W. Zhang, and S. Seung, "Stochastic policy gradient reinforcement learning on a simple 3D biped," in *Proc. Int. Conf. Intelligent Robots and Systems (IROS 2004)*, Sendai, Japan, Oct. 2004, pp. 2849–2854.

[66] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Proc. IEEE Int. Conf. Intelligent Robotics Systems (IROS 2006)*, Beijing, Oct. 9–15, 2006, pp. 2219–2225.

[67] P. Dayan and G. Hinton, "Using EM for reinforcement learning," *Neural Comput.*, vol. 9, no. 2, pp. 271–278, 1997.

[68] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. R. Statist. Soc. B*, vol. 39, no. 1, pp. 1–38, 1977.

[69] J. Kober and J. Peters, "Learning motor primitives in robotics," in *Proc. Advances in Neural Information Processing Systems 21 (NIPS 2008)*, D. Schuurmans, J. Benigio, and D. Koller, Eds. Vancouver, BC, Dec. 8–11, 2009, pp. 297–304.

[70] M. Toussaint and A. Storkey, "Probabilistic inference for solving discrete and continuous state Markov decision processes," presented at the 23nd Int. Conf. Machine Learning (ICML 2006), 2006.

[71] N. Vlassis, M. Toussaint, G. Kontes, and S. Piperidis, "Learning model-free control by a Monte-Carlo EM algorithm," *Auton. Robots*, vol. 27, no. 2, pp. 123–130, 2009.

[72] H. J. Kappen, "Linear theory for control of nonlinear stochastic systems," *Phys. Rev. Lett.*, vol. 95, no. 20, pp. 200201–200204, Nov. 2005.

[73] H. J. Kappen, "An introduction to stochastic control theory, path integrals and reinforcement learning," in *Cooperative Behavior in Neural Systems*, vol. 887, J. Marro, P. L. Garrido, and J. J. Torres, Eds. 2007, pp. 149–181.

[74] E. Theodorou, J. Buchli, and S. Schaal, "Path integral stochastic optimal control for rigid body dynamics," presented at the IEEE Int. Symp. Approximate Dynamic Programming and Reinforcement Learning (ADPRL2009), Nashville, TN, Mar. 30–Apr. 2, 2009.

[75] E. Todorov, "Efficient computation of optimal actions," *Proc. Nat. Acad. Sci. USA*, vol. 106, no. 28, pp. 11478–11483, July 2009.

[76] A. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *Advances in Neural Information Processing Systems 15*, S. Becker, S. Thrun, and K. Obermayer, Eds. 2003, pp. 1547–1554.

[77] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer-Verlag, 2006.

[78] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Robot programming by demonstration," in *Handbook of Robotics*, vol. 1, B. Siciliano and O. Khatib, Eds. Cambridge, MA: MIT Press, 2008, ch. 59.

[79] Y. Wada and M. Kawato, "Trajectory formation of arm movement by a neural network with forward and inverse dynamics models," *Syst. Comput. Jpn.*, vol. 24, pp. 37–50, 1994.

[80] T. Inamura, I. Toshima, H. Tanie, and Y. Nakamura, "Embodied symbol emergence based on mimesis theory," *Int. J. Robot. Res.*, vol. 23, no. 4–5, p. 363, Apr.-May 2004.

[81] A. Y. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *Proc. 17th Int. Conf. Machine Learning (ICML 2000)*, Stanford, CA, 2000, pp. 663–670.

[82] P. Abbeel and A. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proc. 21st Int. Conf. Machine Learning*, 2004.

[83] N. Ratliff, D. Silver, and J. A. Bagnell, "Learning to search: Functional gradient techniques for imitation learning," *Auton. Robots*, vol. 27, no. 1, pp. 25–53, 2009.

Stefan Schaal is a professor of computer science, neuroscience, and biomedical engineering at the University of Southern California and an invited researcher at the ATR Computational Neuroscience Laboratory in Japan. He has coauthored more than 200 papers in refereed journals and conferences. He is a cofounder of the *IEEE/RAS International Conference and Humanoid Robotics* as well as *Robotics Science and Systems*. He serves on the editorial board of *Neural Networks*, *International Journal of Humanoid Robotics*, and *Frontiers in Neurorobotics*. He is a Member of the German National Academic Foundation (Studienstiftung des Deutschen Volkes), Alexander von Humboldt Foundation, Society for Neuroscience, the Society for Neural Control of Movement, the IEEE, and AAAS. His research interests include topics of statistical and machine learning, neural networks, computational neuroscience, functional brain imaging, nonlinear dynamics, nonlinear control theory, and biomimetic robotics.

Christopher G. Atkeson received his M.S. degree in applied mathematics (computer science) from Harvard University and his Ph.D. degree in brain and cognitive sciences from Massachusetts Institute of Technology (MIT). He is a professor at the Robotics Institute and Human–Computer Interaction Institute, Carnegie Mellon University. He joined the MIT as a faculty in 1986 and moved to the Georgia Institute of Technology College of Computing in 1994. He has received the National Science Foundation Presidential Young Investigator Award, Sloan Research Fellowship, and Teaching Award from the MIT Graduate Student Council. His research focuses on humanoid robotics and robot learning by using challenging dynamic tasks such as juggling. His specific research interests include nonparametric learning, memory-based learning including approaches based on trajectory libraries, reinforcement learning, and other forms of learning based on optimal control, learning from demonstration, and modeling human behavior.

Address for Correspondence: Stefan Schaal, Computer Science, Neuroscience, and Biomedical Engineering, University of Southern California, Los Angeles, CA 90089-2905 USA. E-mail: sschaal@usc.edu.