Reinforcement Learning in Robotics: A Survey

Jens Kober[*][†] J. Andrew Bagnell[‡] Jan Peters[§][¶]

email: jkober@cor-lab.uni-bielefeld.de, dbagnell@ri.cmu.edu, mail@jan-peters.net

Reinforcement learning offers to robotics a framework and set of tools for the design of sophisticated and hard-to-engineer behaviors. Conversely, the challenges of robotic problems provide both inspiration, impact, and validation for developments in reinforcement learning. The relationship between disciplines has sufficient promise to be likened to that between physics and mathematics. In this article, we attempt to strengthen the links between the two research communities by providing a survey of work in reinforcement learning for behavior generation in robots. We highlight both key challenges in robot reinforcement learning as well as notable successes. We discuss how contributions tamed the complexity of the domain and study the role of algorithms, representations, and prior knowledge in achieving these successes. As a result, a particular focus of our paper lies on the choice between model-based and model-free as well as between value function-based and policy search methods. By analyzing a simple problem in some detail we demonstrate how reinforcement learning approaches may be profitably applied, and we note throughout open questions and the tremendous potential for future research.

keywords: reinforcement learning, learning control, robot, survey

1 Introduction

A remarkable variety of problems in robotics may be naturally phrased as ones of *reinforcement learning*. Reinforcement learning (RL) enables a robot to autonomously

---
[*]Bielefeld University, CoR-Lab Research Institute for Cognition and Robotics, Universitätsstr. 25, 33615 Bielefeld, Germany

[†]Honda Research Institute Europe, Carl-Legien-Str. 30, 63073 Offenbach/Main, Germany
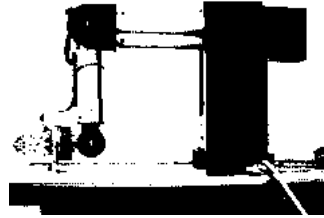
[‡]Carnegie Mellon University, Robotics Institute, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA

[§]Max Planck Institute for Intelligent Systems, Department of Empirical Inference, Spemannstr. 38, 72076 Tübingen, Germany

[¶]Technische Universität Darmstadt, FB Informatik, FG Intelligent Autonomous Systems, Hochschulstr. 10, 64289 Darmstadt, Germany

(a) OBELIX robot

(b) Zebra Zero robot

(c) Autonomous helicopter

(d) Sarcos humanoid DB

Figure 1: This figure illustrates a small sample of robots with behaviors that were reinforcement learned. These cover the whole range of aerial vehicles, robotic arms, autonomous vehicles, and humanoid robots. (a) The OBELIX robot is a wheeled mobile robot that learned to push boxes (Mahadevan and Connell, 1992) with a value function-based approach (Picture reprint with permission of Sridhar Mahadevan). (b) A Zebra Zero robot arm learned a peg-in-hole insertion task (Gullapalli et al., 1994) with a model-free policy gradient approach (Picture reprint with permission of Rod Grupen). (c) Carnegie Mellon's autonomous helicopter leveraged a model-based policy search approach to learn a robust flight controller (Bagnell and Schneider, 2001). (d) The Sarcos humanoid DB learned a pole-balancing task (Schaal, 1996) using forward models (Picture reprint with permission of Stefan Schaal).

discover an optimal behavior through trial-and-error interactions with its environment. Instead of explicitly detailing the solution to a problem, in reinforcement learning the designer of a control task provides feedback in terms of a scalar objective function that measures the one-step performance of the robot. Figure 1 illustrates the diverse set of robots that have learned tasks using reinforcement learning.

Consider, for example, attempting to train a robot to return a table tennis ball over the net (Muelling et al., 2012). In this case, the robot might make an observations of dynamic variables specifying ball position and velocity and the internal dynamics of the joint position and velocity. This might in fact capture well the *state s* of the system – providing a complete statistic for predicting future observations. The *actions a* available to the robot might be the torque sent to motors or the desired accelerations sent to an inverse dynamics control system. A function π that generates the motor commands (i.e., the actions) based on the incoming ball and current internal arm observations (i.e.,
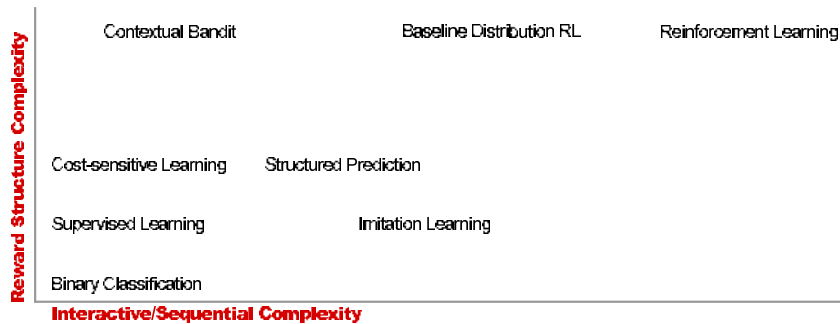
Figure 2: A illustration of the inter-relations between well-studied learning problems in the literature along axes that attempt to capture both the information and complexity available in reward signals and the complexity of sequential interaction between learner and environment. Each problem subsumes those to the left and below; reduction techniques provide methods whereby harder problems (above and right) may be addressed using repeated application of algorithms built for simpler problems. (Langford and Zadrozny, 2005)

the state) would be called the *policy*. A reinforcement learning problem is to find a policy that optimizes the long term sum of *rewards* $R(s, a)$; a reinforcement learning algorithm is one designed to find such a (near)-optimal policy. The reward function in this example could be based on the success of the hits as well as secondary criteria like energy consumption.

1.1 Reinforcement Learning in the Context of Machine Learning

In the problem of reinforcement learning, an agent explores the space of possible strategies and receives feedback on the outcome of the choices made. From this information, a "good" – or ideally optimal – policy (i.e., strategy or controller) must be deduced.

Reinforcement learning may be understood by contrasting the problem with other areas of study in machine learning. In supervised learning (Langford and Zadrozny, 2005), an agent is directly presented a sequence of independent examples of correct predictions to make in different circumstances. In imitation learning, an agent is provided demonstrations of actions of a good strategy to follow in given situations (Argall et al., 2009; Schaal, 1999).

To aid in understanding the RL problem and its relation with techniques widely used within robotics, Figure 2 provides a schematic illustration of two axes of problem variability: the complexity of sequential interaction and the complexity of reward structure. This hierarchy of problems, and the relations between them, is a complex one, varying in manifold attributes and difficult to condense to something like a simple linear ordering on problems. Much recent work in the machine learning community has focused on understanding the diversity and the inter-relations between problem classes. The figure should be understood in this light as providing a crude picture of the relationship between areas of machine learning research important for robotics.

Each problem subsumes those that are both below and to the left in the sense that one may always frame the simpler problem in terms of the more complex one; note that some problems are not linearly ordered. In this sense, reinforcement learning subsumes much of the scope of classical machine learning as well as contextual bandit and imitation learning problems. Reduction algorithms (Langford and Zadrozny, 2005) are used to convert effective solutions for one class of problems into effective solutions for others, and have proven to be a key technique in machine learning.

At lower left, we find the paradigmatic problem of supervised learning, which plays a crucial role in applications as diverse as face detection and spam filtering. In these problems (including binary classification and regression), a learner's goal is to map observations (typically known as features or covariates) to actions which are usually a discrete set of classes or a real value. These problems possess no interactive component: the design and analysis of algorithms to address these problems rely on training and testing instances as independent and identical distributed random variables. This rules out any notion that a decision made by the learner will impact future observations: supervised learning algorithms are built to operate in a world in which every decision has no effect on the future examples considered. Further, within supervised learning scenarios, during a training phase the "correct" or preferred answer is provided to the learner, so there is no ambiguity about action choices.

More complex reward structures are also often studied: one such is known as cost-sensitive learning, where each training example and each action or prediction is annotated with a cost for making such a prediction. Learning techniques exist that reduce such problems to the simpler classification problem, and active research directly addresses such problems as they are crucial in practical learning applications.

Contextual bandit or associative reinforcement learning problems begin to address the fundamental problem of exploration-vs-exploitation, as information is provided only about a chosen action and not what-might-have-been. These find wide-spread application in problems as diverse as pharmaceutical drug discovery to ad placement on the web, and are one of the most active research areas in the field.

Problems of imitation learning and structured prediction may be seen to vary from supervised learning on the alternate dimension of sequential interaction. Structured prediction, a key technique used within computer vision and robotics, where many predictions are made in concert by leveraging inter-relations between them, may be seen as a simplified variant of imitation learning (Daumé III et al., 2009; Ross et al., 2011a). In imitation learning, we assume that an expert (for example, a human pilot) that we wish to mimic provides demonstrations of a task. While "correct answers" are provided to the learner, complexity arises because any mistake by the learner modifies the future observations from what would have been seen had the expert chosen the controls. Such problems provably lead to compounding errors and violate the basic assumption of independent examples required for successful supervised learning. In fact, in sharp contrast with supervised learning problems where only a single data-set needs to be collected, repeated interaction between learner and teacher appears to both necessary and sufficient (Ross et al., 2011b) to provide performance guarantees in both theory and practice in imitation learning problems.

Reinforcement learning embraces the full complexity of these problems by requiring both interactive, sequential prediction as in imitation learning as well as complex reward structures with only "bandit" style feedback on the actions actually chosen. It is this combination that enables so many problems of relevance to robotics to be framed in these terms; it is this same combination that makes the problem both information-theoretically and computationally hard.

We note here briefly the problem termed "Baseline Distribution RL": this is the standard RL problem with the additional benefit for the learner that it may draw initial states from a distribution provided by an expert instead of simply an initial state chosen by the problem. As we describe further in Section 5.1, this additional information of which states matter dramatically affects the complexity of learning.

1.2 Reinforcement Learning in the Context of Optimal Control

Reinforcement Learning (RL) is very closely related to the theory of classical optimal control, as well as dynamic programming, stochastic programming, simulation-optimization, stochastic search, and optimal stopping (Powell, 2012). Both RL and optimal control address the problem of finding an optimal policy (often also called the controller or control policy) that optimizes an objective function (i.e., the accumulated cost or reward), and both rely on the notion of a system being described by an underlying set of states, controls and a plant or model that describes transitions between states. However, optimal control assumes perfect knowledge of the system's description in the form of a model (i.e., a function T that describes what the next state of the robot will be given the current state and action). For such models, optimal control ensures strong guarantees which, nevertheless, often break down due to model and computational approximations. In contrast, reinforcement learning operates directly on measured data and rewards from interaction with the environment. Reinforcement learning research has placed great focus on addressing cases which are analytically intractable using approximations and data-driven techniques. One of the most important approaches to reinforcement learning within robotics centers on the use of classical optimal control techniques (e.g. Linear-Quadratic Regulation and Differential Dynamic Programming) to system models learned via repeated interaction with the environment (Atkeson, 1998; Bagnell and Schneider, 2001; Coates et al., 2009). A concise discussion of viewing reinforcement learning as "adaptive optimal control" is presented in (Sutton et al., 1991).

1.3 Reinforcement Learning in the Context of Robotics

Robotics as a reinforcement learning domain differs considerably from most well-studied reinforcement learning benchmark problems. In this article, we highlight the challenges faced in tackling these problems. Problems in robotics are often best represented with high-dimensional, continuous states and actions (note that the 10-30 dimensional continuous actions common in robot reinforcement learning are considered large (Powell, 2012)). In robotics, it is often unrealistic to assume that the true state is completely observable and noise-free. The learning system will not be able to know precisely in

which state it is and even vastly different states might look very similar. Thus, robotics reinforcement learning are often modeled as partially observed, a point we take up in detail in our formal model description below. The learning system must hence use filters to estimate the true state. It is often essential to maintain the information state of the environment that not only contains the raw observations but also a notion of uncertainty on its estimates (e.g., both the mean and the variance of a Kalman filter tracking the ball in the robot table tennis example).

Experience on a real physical system is tedious to obtain, expensive and often hard to reproduce. Even getting to the same initial state is impossible for the robot table tennis system. Every single trial run, also called a roll-out, is costly and, as a result, such applications force us to focus on difficulties that do not arise as frequently in classical reinforcement learning benchmark examples. In order to learn within a reasonable time frame, suitable approximations of state, policy, value function, and/or system dynamics need to be introduced. However, while real-world experience is costly, it usually cannot be replaced by learning in simulations alone. In analytical or learned models of the system even small modeling errors can accumulate to a substantially different behavior, at least for highly dynamic tasks. Hence, algorithms need to be robust with respect to models that do not capture all the details of the real system, also referred to as *under-modeling*, and to model uncertainty. Another challenge commonly faced in robot reinforcement learning is the generation of appropriate reward functions. Rewards that guide the learning system quickly to success are needed to cope with the cost of real-world experience. This problem is called *reward shaping* (Laud, 2004) and represents a substantial manual contribution. Specifying good reward functions in robotics requires a fair amount of domain knowledge and may often be hard in practice.

Not every reinforcement learning method is equally suitable for the robotics domain. In fact, many of the methods thus far demonstrated on difficult problems have been model-based (Atkeson et al., 1997; Abbeel et al., 2007; Deisenroth and Rasmussen, 2011) and robot learning systems often employ policy search methods rather than value function-based approaches (Gullapalli et al., 1994; Miyamoto et al., 1996; Bagnell and Schneider, 2001; Kohl and Stone, 2004; Tedrake et al., 2005; Peters and Schaal, 2008a,b; Kober and Peters, 2009; Deisenroth et al., 2011). Such design choices stand in contrast to possibly the bulk of the early research in the machine learning community (Kaelbling et al., 1996; Sutton and Barto, 1998). We attempt to give a fairly complete overview on real robot reinforcement learning citing most original papers while grouping them based on the key insights employed to make the Robot Reinforcement Learning problem tractable. We isolate key insights such as choosing an appropriate representation for a value function or policy, incorporating prior knowledge, and transfer knowledge from simulations.

This paper surveys a wide variety of tasks where reinforcement learning has been successfully applied to robotics. If a task can be phrased as an optimization problem and exhibits temporal structure, reinforcement learning can often be profitably applied to both phrase and solve that problem. The goal of this paper is twofold. On the one hand, we hope that this paper can provide indications for the robotics community which type of problems can be tackled by reinforcement learning and provide pointers to approaches that are promising. On the other hand, for the reinforcement learning community, this

6

paper can point out novel real-world test beds and remarkable opportunities for research on open questions. We focus mainly on results that were obtained on physical robots with tasks going beyond typical reinforcement learning benchmarks.

We concisely present reinforcement learning techniques in the context of robotics in Section 2. The challenges in applying reinforcement learning in robotics are discussed in Section 3. Different approaches to making reinforcement learning tractable are treated in Sections 4 to 6. In Section 7, the example of ball-in-a-cup is employed to highlight which of the various approaches discussed in the paper have been particularly helpful to make such a complex task tractable. Finally, in Section 8, we summarize the specific problems and benefits of reinforcement learning in robotics and provide concluding thoughts on the problems and promise of reinforcement learning in robotics.

2 A Concise Introduction to Reinforcement Learning

In reinforcement learning, an agent tries to maximize the accumulated reward over its life-time. In an *episodic* setting, where the task is restarted after each end of an episode, the objective is to maximize the total reward per episode. If the task is on-going without a clear beginning and end, either the average reward over the whole life-time or a discounted return (i.e., a weighted average where distant rewards have less influence) can be optimized. In such reinforcement learning problems, the agent and its environment may be modeled being in a state $s \in S$ and can perform actions $a \in A$, each of which may be members of either discrete or continuous sets and can be multi-dimensional. A state s contains all relevant information about the current situation to predict future states (or observables); an example would be the current position of a robot in a navigation task[1]. An action a is used to control (or change) the state of the system. For example, in the navigation task we could have the actions corresponding to torques applied to the wheels. For every step, the agent also gets a reward R, which is a scalar value and assumed to be a function of the state and observation. (It may equally be modeled as a random variable that depends on only these variables.) In the navigation task, a possible reward could be designed based on the energy costs for taken actions and rewards for reaching targets. The goal of reinforcement learning is to find a mapping from states to actions, called policy π, that picks actions a in given states s maximizing the cumulative expected reward. The policy π is either deterministic or probabilistic. The former always uses the exact same action for a given state in the form $a = \pi(s)$, the later draws a sample from a distribution over actions when it encounters a state, i.e., $a \sim \pi(s, a) = P(a|s)$. The reinforcement learning agent needs to discover the relations between states, actions, and rewards. Hence exploration is required which can either be directly embedded in the policy or performed separately and only as part of the learning process.

Classical reinforcement learning approaches are based on the assumption that we have a *Markov Decision Process* (MDP) consisting of the set of states S, set of actions A, the rewards R and transition probabilities T that capture the dynamics of a system.

---

[1]When only observations but not the complete state is available, the sufficient statistics of the filter can alternatively serve as state s. Such a state is often called information or belief state.

Transition probabilities (or densities in the continuous state case) $T(s', a, s) = P(s'|s, a)$ describe the effects of the actions on the state. Transition probabilities generalize the notion of deterministic dynamics to allow for modeling outcomes are uncertain even given full state. The Markov property requires that the next state s' and the reward only depend on the previous state s and action a (Sutton and Barto, 1998), and not on additional information about the past states or actions. In a sense, the Markov property recapitulates the idea of state – a state if a sufficient statistic for predicting the future, rendering previous observations irrelevant. In general in robotics, we may only be able to find some approximate notion of state.

Different types of reward functions are commonly used, including rewards depending only on the current state $R = R(s)$, rewards depending on the current state and action $R = R(s, a)$, and rewards including the transitions $R = R(s', a, s)$. Most of the theoretical guarantees only hold if the problem adheres to a Markov structure, however in practice, many approaches work very well for many problems that do not fulfill this requirement.

2.1 Goals of Reinforcement Learning

The goal of reinforcement learning is to discover an optimal policy π^* that maps states (or observations) to actions so as to maximize the expected return J, which corresponds to the cumulative expected reward. There are different models of optimal behavior (Kaelbling et al., 1996) which result in different definitions of the expected return. A finite-horizon model only attempts to maximize the expected reward for the horizon H, i.e., the next H (time-)steps h

$$J = E \left\{ \sum_{h=0}^{H} R_h \right\}.$$

This setting can also be applied to model problems where it is known how many steps are remaining.

Alternatively, future rewards can be discounted by a discount factor γ (with $0 \leq \gamma < 1$)

$$J = E \left\{ \sum_{h=0}^{\infty} \gamma^h R_h \right\}.$$

This is the setting most frequently discussed in classical reinforcement learning texts. The parameter γ affects how much the future is taken into account and needs to be tuned manually. As illustrated in (Kaelbling et al., 1996), this parameter often qualitatively changes the form of the optimal solution. Policies designed by optimizing with small γ are myopic and greedy, and may lead to poor performance if we actually care about longer term rewards. It is straightforward to show that the optimal control law can be unstable if the discount factor is too low (e.g., it is not difficult to show this destabilization even for discounted linear quadratic regulation problems). Hence, discounted formulations are frequently inadmissible in robot control.

In the limit when γ approaches 1, the metric approaches what is known as the average-reward criterion (Bertsekas, 1995),

$$J = \lim_{H \to \infty} E \left\{ \frac{1}{H} \sum_{h=0}^{H} R_h \right\}.$$

This setting has the problem that it cannot distinguish between policies that initially gain a transient of large rewards and those that do not. This transient phase, also called prefix, is dominated by the rewards obtained in the long run. If a policy accomplishes both an optimal prefix as well as an optimal long-term behavior, it is called bias optimal Lewis and Puterman (2001). An example in robotics would be the transient phase during the start of a rhythmic movement, where many policies will accomplish the same long-term reward but differ substantially in the transient (e.g., there are many ways of starting the same gait in dynamic legged locomotion) allowing for room for improvement in practical application.

In real-world domains, the shortcomings of the discounted formulation are often more critical than those of the average reward setting as stable behavior is often more important than a good transient (Peters et al., 2004). We also often encounter an episodic control task, where the task runs only for H time-steps and then reset (potentially by human intervention) and started over. This horizon, H, may be arbitrarily large, as long as the expected reward over the episode can be guaranteed to converge. As such episodic tasks are probably the most frequent ones, finite-horizon models are often the most relevant.

Two natural goals arise for the learner. In the first, we attempt to find an optimal strategy at the end of a phase of training or interaction. In the second, the goal is to maximize the reward over the whole time the robot is interacting with the world.

In contrast to supervised learning, the learner must first discover its environment and is not told the optimal action it needs to take. To gain information about the rewards and the behavior of the system, the agent needs to explore by considering previously unused actions or actions it is uncertain about. It needs to decide whether to play it safe and stick to well known actions with (moderately) high rewards or to dare trying new things in order to discover new strategies with an even higher reward. This problem is commonly known as the *exploration-exploitation trade-off*.

In principle, reinforcement learning algorithms for Markov Decision Processes with performance guarantees are known (Kakade, 2003; Kearns and Singh, 2002; Brafman and Tennenholtz, 2002) with polynomial scaling in the size of the state and action spaces, an additive error term, as well as in the horizon length (or a suitable substitute including the discount factor or "mixing time" (Kearns and Singh, 2002)). However, state spaces in robotics problems are often tremendously large as they scale exponentially in the number of state variables and often are continuous. This challenge of exponential growth is often referred to as the *curse of dimensionality* (Bellman, 1957) (also discussed in Section 3.1).

Off-policy methods learn independent of the employed policy, i.e., an explorative strategy that is different from the desired final policy can be employed during the learning process. *On-policy* methods collect sample information about the environment using the current policy. As a result, exploration must be built into the policy and determine

the speed of the policy improvements. Such exploration and the performance of the policy can result in an exploration-exploitation trade-off between long- and short-term improvement of the policy. Modeling exploration models with probability distributions has surprising implications, e.g., stochastic policies have been shown to be the optimal stationary policies for selected problems (Sutton et al., 1999; Jaakkola et al., 1993) and can even break the curse of dimensionality (Rust, 1997). Furthermore, stochastic policies often allow the derivation of new policy update steps with surprising ease.

The agent needs to determine a correlation between actions and reward signals. An action taken does not have to have an immediate effect on the reward but can also influence a reward in the distant future. The difficulty in assigning credit for rewards is directly related to the horizon or mixing time of the problem. It also increases with the dimensionality of the actions as not all parts of the action may contribute equally.

The classical reinforcement learning setup is a MDP where additionally to the states S, actions A, and rewards R we also have transition probabilities $T(s', a, s)$. Here, the reward is modeled as a reward function $R(s, a)$. If both the transition probabilities and reward function are known, this can be seen as an optimal control problem (Powell, 2012).

2.2 Reinforcement Learning in the Average Reward Setting

We focus on the average-reward model in this section. Similar derivations exist for the finite horizon and discounted reward cases. In many instances, the average-reward case is often more suitable in a robotic setting as we do not have to choose a discount factor and we do not have to explicitly consider time in the derivation.

To make a policy able to be optimized by continuous optimization techniques, we write a policy as a conditional probability distribution $\pi(s, a) = P(a|s)$. Below, we consider restricted policies that are parametrized by a vector θ. In reinforcement learning, the policy is usually considered to be stationary and memory-less. Reinforcement learning and optimal control aim at finding the optimal policy π^* or equivalent policy parameters θ^* which maximize the average return $J(\pi) = \sum_{s,a} \mu^\pi(s) \pi(s, a) R(s, a)$ where μ^π is the stationary state distribution generated by policy π acting in the environment, i.e., the MDP. It can be shown (Puterman, 1994) that such policies that map states (even deterministically) to actions are sufficient to ensure optimality in this setting – a policy needs neither to remember previous states visited, actions taken, or the particular time step. For simplicity and to ease exposition, we assume that this distribution is unique. Markov Decision Processes where this fails (i.e., non-ergodic processes) require more care in analysis, but similar results exist (Puterman, 1994). The transitions between states s caused by actions a are modeled as $T(s, a, s') = P(s'|s, a)$. We can then frame the control problem as an optimization of

$$\max_\pi J(\pi) = \sum_{s,a} \mu^\pi(s) \pi(s, a) R(s, a), \tag{1}$$

$$\text{s.t. } \mu^\pi(s') = \sum_{s,a} \mu^\pi(s) \pi(s, a) T(s, a, s'), \ \forall s' \in S, \tag{2}$$

$$1 = \sum_{s,a} \mu^\pi(s) \pi(s, a) \tag{3}$$

$$\pi(s, a) \geq 0, \ \forall s \in S, a \in A.$$

10

Here, Equation (2) defines stationarity of the state distributions μ^π (i.e., it ensures that it is well defined) and Equation (3) ensures a proper state-action probability distribution. This optimization problem can be tackled in two substantially different ways (Bellman, 1967, 1971). We can search the optimal solution directly in this original, primal problem or we can optimize in the Lagrange dual formulation. Optimizing in the primal formulation is known as *policy search* in reinforcement learning while searching in the dual formulation is known as a *value function-based approach*.

2.2.1 Value Function Approaches

Much of the reinforcement learning literature has focused on solving the optimization problem in Equations (1-3) in its dual form (Gordon, 1999; Puterman, 1994)[2]. Using Lagrange multipliers $V^\pi(s')$ and $\bar{R}$, we can express the Lagrangian of the problem by

$$
\begin{aligned}
L &= \sum_{s,a} \mu^\pi(s)\,\pi(s,a)\,R(s,a) \\
&\quad + \sum_{s'} V^\pi(s') \left[\sum_{s,a} \mu^\pi(s)\,\pi(s,a)\,T(s,a,s') - \mu^\pi(s') \right] \\
&\quad + \bar{R}\left[1 - \sum_{s,a} \mu^\pi(s)\,\pi(s,a) \right] \\
&= \sum_{s,a} \mu^\pi(s)\,\pi(s,a) \left[R(s,a) + \sum_{s'} V^\pi(s')\,T(s,a,s') - \bar{R} \right] \\
&\quad - \sum_{s'} V^\pi(s')\,\mu^\pi(s') \underbrace{\sum_{a'} \pi(s',a')}_{=1} + \bar{R}.
\end{aligned}
$$

Using the property $\sum_{s',a'} V(s')\,\mu^\pi(s')\,\pi(s',a') = \sum_{s,a} V(s)\,\mu^\pi(s)\,\pi(s,a)$, we can obtain the Karush-Kuhn-Tucker conditions (Kuhn and Tucker, 1950) by differentiating with respect to $\mu^\pi(s)\,\pi(s,a)$ which yields extrema at

$$
\partial_{\mu^\pi \pi} L = R(s,a) + \sum_{s'} V^\pi(s')\,T(s,a,s') - \bar{R} - V^\pi(s) = 0.
$$

This statement implies that there are as many equations as the number of states multiplied by the number of actions. For each state there can be one or several optimal actions a^* that result in the *same* maximal value, and, hence, can be written in terms of the optimal action a^* as $V^{\pi^*}(s) = R(s,a^*) - \bar{R} + \sum_{s'} V^{\pi^*}(s')\,T(s,a^*,s')$. As a^* is generated by the same optimal policy π^*, we know the condition for the multipliers at

---

[2]For historical reasons, what we call the dual is often referred to in the literature as the primal. We argue that problem of optimizing expected reward is the fundamental problem, and values are an auxiliary concept.

optimality is

$$V^*(s) = \max_{a^*} \left[R(s, a^*) - \bar{R} + \sum_{s'} V^*(s') T(s, a^*, s') \right],\qquad(4)$$

where $V^*(s)$ is a shorthand notation for $V^{\pi^*}(s)$. This statement is equivalent to the *Bellman Principle of Optimality* (Bellman, 1957)[3] that states "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." Thus, we have to perform an optimal action a^*, and, subsequently, follow the optimal policy π^* in order to achieve a global optimum. When evaluating Equation (4), we realize that optimal value function $V^*(s)$ corresponds to the long term additional reward, beyond the average reward $\bar{R}$, gained by starting in state s while taking optimal actions a^* (according to the optimal policy π^*). This principle of optimality has also been crucial in enabling the field of optimal control (Kirk, 1970).

Hence, we have a dual formulation of the original problem that serves as condition for optimality. Many traditional reinforcement learning approaches are based on identifying (possibly approximate) solutions to this equation, and are known as *value function methods*. Instead of directly learning a policy, they first approximate the Lagrangian multipliers $V^*(s)$, also called the value function, and use it to reconstruct the optimal policy. The value function $V^\pi(s)$ is defined equivalently, however instead of always taking the optimal action a^*, the action a is picked according to a policy π

$$V^\pi(s) = \sum_a \pi(s, a) \left(R(s, a) - \bar{R} + \sum_{s'} V^\pi(s') T(s, a, s') \right).$$

Instead of the value function $V^\pi(s)$ many algorithms rely on the state-action value function $Q^\pi(s, a)$ instead, which has advantages for determining the optimal policy as shown below. This function is defined as

$$Q^\pi(s, a) = R(s, a) - \bar{R} + \sum_{s'} V^\pi(s') T(s, a, s').$$

In contrast to the value function $V^\pi(s)$, the state-action value function $Q^\pi(s, a)$ explicitly contains the information about the effects of a particular action. The optimal state-action value function is

$$
\begin{aligned}
Q^*(s, a) &= R(s, a) - \bar{R} + \sum_{s'} V^*(s') T(s, a, s'). \\
&= R(s, a) - \bar{R} + \sum_{s'} \left(\max_{a'} Q^*(s', a') \right) T(s, a, s').
\end{aligned}
$$

---

[3]This optimality principle was originally formulated for a setting with discrete time steps and continuous states and actions but is also applicable for discrete states and actions.

It can be shown that an optimal, deterministic policy $\pi^*(s)$ can be reconstructed by always picking the action a^* in the current state that leads to the state s with the highest value $V^*(s)$

$$\pi^*(s) = \arg\max_a \left(R(s,a) - \bar{R} + \sum_{s'} V^*(s') T(s,a,s') \right)$$

If the optimal value function $V^*(s')$ and the transition probabilities $T(s,a,s')$ for the following states are known, determining the optimal policy is straightforward in a setting with discrete actions as an exhaustive search is possible. For continuous spaces, determining the optimal action a^* is an optimization problem in itself. If both states and actions are discrete, the value function and the policy may, in principle, be represented by tables and picking the appropriate action is reduced to a look-up. For large or continuous spaces representing the value function as a table becomes intractable. Function approximation is employed to find a lower dimensional representation that matches the real value function as closely as possible, as discussed in Section 2.4. Using the state-action value function $Q^*(s,a)$ instead of the value function $V^*(s)$

$$\pi^*(s) = \arg\max_a (Q^*(s,a)),$$

avoids having to calculate the weighted sum over the successor states, and hence no knowledge of the transition function is required.

A wide variety of methods of value function based reinforcement learning algorithms that attempt to estimate $V^*(s)$ or $Q^*(s,a)$ have been developed and can be split mainly into three classes: (i) dynamic programming-based optimal control approaches such as policy iteration or value iteration, (ii) rollout-based Monte Carlo methods and (iii) temporal difference methods such as TD(λ), Q-learning, and SARSA.

Dynamic Programming-Based Methods require a model of the transition probabilities $T(s',a,s)$ and the reward function $R(s,a)$ to calculate the value function. The model does not necessarily need to be predetermined but can also be learned from data, potentially incrementally. Such methods are called *model-based*. Typical methods include policy iteration and value iteration.

Policy iteration alternates between the two phases of policy evaluation and policy improvement. The approach is initialized with an arbitrary policy. Policy evaluation determines the value function for the current policy. Each state is visited and its value is updated based on the current value estimates of its successor states, the associated transition probabilities, as well as the policy. This procedure is repeated until the value function converges to a fixed point, which corresponds to the true value function. Policy improvement greedily selects the best action in every state according to the value function as shown above. The two steps of policy evaluation and policy improvement are iterated until the policy does not change any longer.

Policy iteration only updates the policy once the policy evaluation step has converged. In contrast, value iteration combines the steps of policy evaluation and policy improve-

ment by directly updating the value function based on Eq. (4) every time a state is updated.

Monte Carlo Methods use sampling in order to estimate the value function. This procedure can be used to replace the policy evaluation step of the dynamic programming-based methods above. Monte Carlo methods are *model-free*, i.e., they do not need an explicit transition function. They perform roll-outs by executing the current policy on the system, hence operating on-policy. The frequencies of transitions and rewards are kept track of and are used to form estimates of the value function. For example, in an episodic setting the state-action value of a given state action pair can be estimated by averaging all the returns that were received when starting from them.

Temporal Difference Methods, unlike Monte Carlo methods, do not have to wait until an estimate of the return is available (i.e., at the end of an episode) to update the value function. Rather, they use temporal errors and only have to wait until the next time step. The temporal error is the difference between the old estimate and a new estimate of the value function, taking into account the reward received in the current sample. These updates are done iterativley and, in contrast to dynamic programming methods, only take into account the sampled successor states rather than the complete distributions over successor states. Like the Monte Carlo methods, these methods are model-free, as they do not use a model of the transition function to determine the value function. In this setting, the value function cannot be calculated analytically but has to be estimated from sampled transitions in the MDP. For example, the value function could be updated iteratively by

$$V'(s) = V(s) + \alpha \left(R(s, a) - \bar{R} + V(s') - V(s) \right),$$

where $V(s)$ is the old estimate of the value function, $V'(s)$ the updated one, and α is a learning rate. This update step is called the TD(0)-algorithm in the discounted reward case. In order to perform action selection a model of the transition function is still required.

The equivalent temporal difference learning algorithm for state-action value functions is the average reward case version of SARSA with

$$Q'(s, a) = Q(s, a) + \alpha \left(R(s, a) - \bar{R} + Q(s', a') - Q(s, a) \right),$$

where $Q(s, a)$ is the old estimate of the state-action value function and $Q'(s, a)$ the updated one. This algorithm is on-policy as both the current action a as well as the subsequent action a' are chosen according to the current policy π. The off-policy variant is called R-learning (Schwartz, 1993), which is closely related to Q-learning, with the updates

$$Q'(s, a) = Q(s, a) + \alpha \left(R(s, a) - \bar{R} + \max_{a'} Q(s', a') - Q(s, a) \right).$$

VALUE FUNCTION APPROACHES

Approach	Employed by...
Model-Based	Bakker et al. (2006); Hester et al. (2010, 2012); Kalmár et al. (1998); Martínez-Marín and Duckett (2005); Schaal (1996); Touzet (1997)
Model-Free	Asada et al. (1996); Bakker et al. (2003); Benbrahim et al. (1992); Benbrahim and Franklin (1997); Birdwell and Livingston (2007); Bitzer et al. (2010); Conn and Peters II (2007); Duan et al. (2007, 2008); Fagg et al. (1998); Gaskett et al. (2000); Gräve et al. (2010); Hafner and Riedmiller (2007); Huang and Weng (2002); Huber and Grupen (1997); Ilg et al. (1999); Katz et al. (2008); Kimura et al. (2001); Kirchner (1997); Konidaris et al. (2011a, 2012); Kroemer et al. (2009, 2010); Kwok and Fox (2004); Latzke et al. (2007); Mahadevan and Connell (1992); Matarić (1997); Morimoto and Doya (2001); Nemec et al. (2009, 2010); Oßwald et al. (2010); Paletta et al. (2007); Pendrith (1999); Platt et al. (2006); Riedmiller et al. (2009); Rottmann et al. (2007); Smart and Kaelbling (1998, 2002); Soni and Singh (2006); Tamošiūnaitė et al. (2011); Thrun (1995); Tokic et al. (2009); Touzet (1997); Uchibe et al. (1998); Wang et al. (2006); Willgoss and Iqbal (1999)

Table 1: This table illustrates different value function based reinforcement learning methods employed for robotic tasks (both average and discounted reward cases) and associated publications.

These methods do not require a model of the transition function for determining the deterministic optimal policy $\pi^*(s)$. H-learning (Tadepalli and Ok, 1994) is a related method that estimates a model of the transition probabilities and the reward function in order to perform updates that are reminiscent of value iteration.

An overview of publications using value function based methods is presented in Table 1. Here, model-based methods refers to all methods that employ a predetermined or a learned model of system dynamics.

2.2.2 Policy Search

The primal formulation of the problem in terms of policy rather then value offers many features relevant to robotics. It allows for a natural integration of expert knowledge, e.g., through both structure and initializations of the policy. It allows domain-appropriate pre-structuring of the policy in an approximate form without changing the original problem. Optimal policies often have many fewer parameters than optimal value functions. For example, in linear quadratic control, the value function has quadratically many parameters in the dimensionality of the state-variables while the policy requires only linearly many parameters. Local search in policy space can directly lead to good results as exhibited by early hill-climbing approaches (Kirk, 1970), as well as more recent successes

(see Table 2). Additional constraints can be incorporated naturally, e.g., regularizing the change in the path distribution. As a result, policy search often appears more natural to robotics.

Nevertheless, policy search has been considered the harder problem for a long time as the optimal solution cannot directly be determined from Equations (1-3) while the solution of the dual problem leveraging *Bellman Principle of Optimality* (Bellman, 1957) enables dynamic programming based solutions.

Notwithstanding this, in robotics, policy search has recently become an important alternative to value function based methods due to better scalability as well as the convergence problems of approximate value function methods (see Sections 2.3 and 4.2). Most policy search methods optimize locally around existing policies π, parametrized by a set of policy parameters θ_i, by computing changes in the policy parameters $\Delta\theta_i$ that will increase the expected return and results in iterative updates of the form

$$\theta_{i+1} = \theta_i + \Delta\theta_i.$$

The computation of the policy update is the key step here and a variety of updates have been proposed ranging from pairwise comparisons (Strens and Moore, 2001; Ng et al., 2004a) over gradient estimation using finite policy differences (Geng et al., 2006; Kohl and Stone, 2004; Mitsunaga et al., 2005; Roberts et al., 2010; Sato et al., 2002; Tedrake et al., 2005), and general stochastic optimization methods (such as Nelder-Mead (Bagnell and Schneider, 2001), cross entropy (Rubinstein and Kroese, 2004) and population-based methods (Goldberg, 1989)) to approaches coming from optimal control such as differential dynamic programming (DDP) (Atkeson, 1998) and multiple shooting approaches (Betts, 2001). We may broadly break down policy-search methods into "black box" and "white box" methods. Black box methods are general stochastic optimization algorithms (Spall, 2003) using only the expected return of policies, estimated by sampling, and do not leverage any of the internal structure of the RL problem. These may be very sophisticated techniques (Tesch et al., 2011) that use response surface estimates and bandit-like strategies to achieve good performance. White box methods take advantage of some of additional structure within the reinforcement learning domain, including, for instance, the (approximate) Markov structure of problems, developing approximate models, value-function estimates when available (Peters and Schaal, 2008c), or even simply the causal ordering of actions and rewards. A major open issue within the field is the relative merits of the these two approaches: in principle, white box methods leverage more information, but with the exception of models (which have been demonstrated repeatedly to often make tremendous performance improvements, see Section 6), the performance gains are traded-off with additional assumptions that may be violated and less mature optimization algorithms. Some recent work including (Stulp and Sigaud, 2012; Tesch et al., 2011) suggest that much of the benefit of policy search is achieved by black-box methods.

Some of the most popular white-box general reinforcement learning techniques that have translated particularly well into the domain of robotics include: (i) policy gradient approaches based on likelihood-ratio estimation (Sutton et al., 1999), (ii) policy updates

16

inspired by expectation-maximization (Toussaint et al., 2010), and (iii) the path integral methods (Kappen, 2005).

Let us briefly take a closer look at gradient-based approaches first. The updates of the policy parameters are based on a hill-climbing approach, that is following the gradient of the expected return J for a defined step-size α

$$\theta_{i+1} = \theta_i + \alpha \nabla_\theta J.$$

Different methods exist for estimating the gradient $\nabla_\theta J$ and many algorithms require tuning of the step-size α.

In *finite difference gradients* P perturbed policy parameters are evaluated to obtain an estimate of the gradient. Here we have $\Delta \hat{J}_p \approx J(\theta_i + \Delta \theta_p) - J_{\text{ref}}$, where $p = [1..P]$ are the individual perturbations, $\Delta \hat{J}_p$ the estimate of their influence on the return, and J_{ref} is a reference return, e.g., the return of the unperturbed parameters. The gradient can now be estimated by linear regression

$$\nabla_\theta J \approx \left(\Delta \mathbf{\Theta}^\mathrm{T} \Delta \mathbf{\Theta} \right)^{-1} \Delta \mathbf{\Theta}^\mathrm{T} \Delta \hat{\boldsymbol{J}},$$

where the matrix $\Delta \mathbf{\Theta}$ contains all the stacked samples of the perturbations $\Delta \theta_p$ and $\Delta \hat{\boldsymbol{J}}$ contains the corresponding $\Delta \hat{J}_p$. In order to estimate the gradient the number of perturbations needs to be at least as large as the number of parameters. The approach is very straightforward and even applicable to policies that are not differentiable. However, it is usually considered to be very noisy and inefficient. For the finite difference approach tuning the step-size α for the update, the number of perturbations P, and the type and magnitude of perturbations are all critical tuning factors.

Likelihood ratio methods rely on the insight that in an episodic setting where the episodes τ are generated according to the distribution $P^\theta(\tau) = P(\tau|\theta)$ with the return of an episode $J^\tau = \sum_{h=1}^{H} R_h$ and number of steps H the expected return for a set of policy parameter θ can be expressed as

$$J^\theta = \sum_\tau P^\theta(\tau) J^\tau. \tag{5}$$

The gradient of the episode distribution can be written as[4]

$$\nabla_\theta P^\theta(\tau) = P^\theta(\tau) \nabla_\theta \log P^\theta(\tau), \tag{6}$$

which is commonly known as the the likelihood ratio or *REINFORCE* (Williams, 1992) trick. Combining Equations (5) and (6) we get the gradient of the expected return in the form

$$\nabla_\theta J^\theta = \sum_\tau \nabla_\theta P^\theta(\tau) J^\tau = \sum_\tau P^\theta(\tau) \nabla_\theta \log P^\theta(\tau) J^\tau = E\left\{ \nabla_\theta \log P^\theta(\tau) J^\tau \right\}.$$

If we have a stochastic policy $\pi^\theta(s, a)$ that generates the episodes τ, we do not need to keep track of the probabilities of the episodes but can directly express the gradient in

---

[4]From multi-variate calculus we have $\nabla_\theta \log P^\theta(\tau) = \nabla_\theta P^\theta(\tau) / P^\theta(\tau)$.

terms of the policy as $\nabla_\theta \log P^\theta(\tau) = \sum_{h=1}^{H} \nabla_\theta \log \pi^\theta(s, a)$. Finally the gradient of the expected return with respect to the policy parameters can be estimated as

$$\nabla_\theta J^\theta = E\left\{\left(\sum_{h=1}^{H} \nabla_\theta \log \pi^\theta(s_h, a_h)\right) J^\tau\right\}.$$

If we now take into account that rewards at the beginning of an episode cannot be caused by actions taken at the end of an episode, we can replace the return of the episode J^τ by the state-action value function $Q^\pi(s, a)$ and get (Peters and Schaal, 2008c)

$$\nabla_\theta J^\theta = E\left\{\sum_{h=1}^{H} \nabla_\theta \log \pi^\theta(s_h, a_h) Q^\pi(s_h, a_h)\right\},$$

which is equivalent to the *policy gradient theorem* (Sutton et al., 1999). In practice, it is often advisable to subtract a reference J_{ref}, also called baseline, from the return of the episode J^τ or the state-action value function $Q^\pi(s, a)$ respectively to get better estimates, similar to the finite difference approach. In these settings, the exploration is automatically taken care of by the stochastic policy.

Initial gradient-based approaches such as finite differences gradients or REINFORCE (Williams, 1992) have been rather slow. The weight perturbation algorithm is related to REINFORCE but can deal with non-Gaussian distributions which significantly improves the signal to noise ratio of the gradient (Roberts et al., 2010). Recent natural policy gradient approaches (Peters and Schaal, 2008c,b) have allowed for faster convergence which may be advantageous for robotics as it reduces the learning time and required real-world interactions.

A different class of safe and fast policy search methods, that are inspired by expectation-maximization, can be derived when the reward is treated as an improper probability distribution (Dayan and Hinton, 1997). Some of these approaches have proven successful in robotics, e.g., reward-weighted regression (Peters and Schaal, 2008a), Policy Learning by Weighting Exploration with the Returns (Kober and Peters, 2009), Monte Carlo Expectation-Maximization(Vlassis et al., 2009), and Cost-regularized Kernel Regression (Kober et al., 2010). Algorithms with closely related update rules can also be derived from different perspectives including Policy Improvements with Path Integrals (Theodorou et al., 2010) and Relative Entropy Policy Search (Peters et al., 2010a).

Finally, the Policy Search by Dynamic Programming (Bagnell et al., 2003) method is a general strategy that combines policy search with the principle of optimality. The approach learns a non-stationary policy backward in time like dynamic programming methods, but does not attempt to enforce the Bellman equation and the resulting approximation instabilities (See Section 2.4). The resulting approach provides some of the strongest guarantees that are currently known under function approximation and limited observability It has been demonstrated in learning walking controllers and in finding near-optimal trajectories for map exploration (Kollar and Roy, 2008). The resulting method is more expensive than the value function methods because it scales quadratically in the effective time horizon of the problem. Like DDP methods (Atkeson, 1998), it is tied to a non-stationary (time-varying) policy.

Approach	Employed by...
Gradient	Deisenroth and Rasmussen (2011); Deisenroth et al. (2011); Endo et al. (2008); Fidelman and Stone (2004); Geng et al. (2006); Guenter et al. (2007); Gullapalli et al. (1994); Hailu and Sommer (1998); Ko et al. (2007); Kohl and Stone (2004); Kolter and Ng (2009a); Michels et al. (2005); Mitsunaga et al. (2005); Miyamoto et al. (1996); Ng et al. (2004b,a); Peters and Schaal (2008c,b); Roberts et al. (2010); Rosenstein and Barto (2004); Tamei and Shibata (2009); Tedrake (2004); Tedrake et al. (2005)
Other	Abbeel et al. (2006, 2007); Atkeson and Schaal (1997); Atkeson (1998); Bagnell and Schneider (2001); Bagnell (2004); Buchli et al. (2011); Coates et al. (2009); Daniel et al. (2012); Donnart and Meyer (1996); Dorigo and Colombetti (1993); Erden and Leblebicioğlu (2008); Kalakrishnan et al. (2011); Kober and Peters (2009); Kober et al. (2010); Kolter et al. (2008); Kuindersma et al. (2011); Lizotte et al. (2007); Matarić (1994); Pastor et al. (2011); Peters and Schaal (2008a); Peters et al. (2010a); Schaal and Atkeson (1994); Stulp et al. (2011); Svinin et al. (2001); Tamošiūnaitė et al. (2011); Yasuda and Ohkura (2008); Youssef (2005)

Table 2: This table illustrates different policy search reinforcement learning methods employed for robotic tasks and associated publications.

An overview of publications using policy search methods is presented in Table 2.

One of the key open issues in the field is determining when it is appropriate to use each of these methods. Some approaches leverage significant structure specific to the RL problem (e.g. (Theodorou et al., 2010)), including reward structure, Markovanity, causality of reward signals (Williams, 1992), and value-function estimates when available (Peters and Schaal, 2008c). Others embed policy search as a generic, black-box, problem of stochastic optimization (Bagnell and Schneider, 2001; Lizotte et al., 2007; Kuindersma et al., 2011; Tesch et al., 2011). Significant open questions remain regarding which methods are best in which circumstances and further, at an even more basic level, how effective leveraging the kinds of problem structures mentioned above are in practice.

2.3 Value Function Approaches versus Policy Search

Some methods attempt to find a value function or policy which eventually can be employed without significant further computation, whereas others (e.g., the roll-out methods) perform the same amount of computation each time.

If a complete optimal value function is known, a globally optimal solution follows simply by greedily choosing actions to optimize it. However, value-function based approaches have thus far been difficult to translate into high dimensional robotics as they require function approximation for the value function. Most theoretical guarantees no

longer hold for this approximation and even finding the optimal action can be a hard problem due to the brittleness of the approximation and the cost of optimization. For high dimensional actions, it can be as hard computing an improved policy for all states in policy search as finding a single optimal action on-policy for one state by searching the state-action value function.

In principle, a value function requires total coverage of the state space and the *largest local error* determines the quality of the resulting policy. A particularly significant problem is the error propagation in value functions. A small change in the policy may cause a large change in the value function, which again causes a large change in the policy. While this may lead more quickly to good, possibly globally optimal solutions, such learning processes often prove unstable under function approximation (Boyan and Moore, 1995; Kakade and Langford, 2002; Bagnell et al., 2003) and are considerably more dangerous when applied to real systems where overly large policy deviations may lead to dangerous decisions.

In contrast, policy search methods usually only consider the current policy and its neighborhood in order to gradually improve performance. The result is that usually only local optima, and not the global one, can be found. However, these methods work well in conjunction with continuous features. Local coverage and local errors results into improved scaleability in robotics.

Policy search methods are sometimes called *actor*-only methods; value function methods are sometimes called *critic*-only methods. The idea of a critic is to first observe and estimate the performance of choosing controls on the system (i.e., the value function), then derive a policy based on the gained knowledge. In contrast, the actor directly tries to deduce the optimal policy. A set of algorithms called *actor-critic* methods attempt to incorporate the advantages of each: a policy is explicitly maintained, as is a value-function for the current policy. The value function (i.e., the critic) is not employed for action selection. Instead, it observes the performance of the actor and decides when the policy needs to be updated and which action should be preferred. The resulting update step features the local convergence properties of policy gradient algorithms while reducing update variance (Greensmith et al., 2004). There is a trade-off between the benefit of reducing the variance of the updates and having to learn a value function as the samples required to estimate the value function could also be employed to obtain better gradient estimates for the update step. Rosenstein and Barto (2004) propose an actor-critic method that additionally features a supervisor in the form of a stable policy.

2.4 Function Approximation

Function approximation (Rivlin, 1969) is a family of mathematical and statistical techniques used to represent a function of interest when it is computationally or information-theoretically intractable to represent the function exactly or explicitly (e.g. in tabular form). Typically, in reinforcement learning te function approximation is based on sample data collected during interaction with the environment. Function approximation is critical in nearly every RL problem, and becomes inevitable in continuous state ones. In large discrete spaces it is also often impractical to visit or even represent all states and

actions, and function approximation in this setting can be used as a means to generalize to neighboring states and actions.

Function approximation can be employed to represent policies, value functions, and forward models. Broadly speaking, there are two kinds of function approximation methods: *parametric* and *non-parametric*. A parametric function approximator uses a finite set of parameters or arguments with the goal is to find parameters that make this approximation fit the observed data as closely as possible. Examples include linear basis functions and neural networks. In contrast, non-parametric methods expand representational power in relation to collected data and hence are not limited by the representation power of a chosen parametrization (Bishop, 2006). A prominent example that has found much use within reinforcement learning is Gaussian process regression (Rasmussen and Williams, 2006). A fundamental problem with using supervised learning methods developed in the literature for function approximation is that most such methods are designed for independently and identically distributed sample data. However, the data generated by the reinforcement learning process is usually neither independent nor identically distributed. Usually, the function approximator itself plays some role in the data collection process (for instance, by serving to define a policy that we execute on a robot.)

Linear basis function approximators form one of the most widely used approximate value function techniques in continuous (and discrete) state spaces. This is largely due to the simplicity of their representation as well as a convergence theory, albeit limited, for the approximation of value functions based on samples (Tsitsiklis and Van Roy, 1997). Let us briefly take a closer look at a radial basis function network to illustrate this approach. The value function maps states to a scalar value. The state space can be covered by a grid of points, each of which correspond to the center of a Gaussian-shaped basis function. The value of the approximated function is the weighted sum of the values of all basis functions at the query point. As the influence of the Gaussian basis functions drops rapidly, the value of the query points will be predominantly influenced by the neighboring basis functions. The weights are set in a way to minimize the error between the observed samples and the reconstruction. For the mean squared error, these weights can be determined by linear regression. Kolter and Ng (2009b) discuss the benefits of regularization of such linear function approximators to avoid over-fitting.

Other possible function approximators for value functions include wire fitting, which Baird and Klopf (1993) suggested as an approach that makes continuous action selection feasible. The Fourier basis had been suggested by Konidaris et al. (2011b). Even discretizing the state-space can be seen as a form of function approximation where coarse values serve as estimates for a smooth continuous function. One example is tile coding (Sutton and Barto, 1998), where the space is subdivided into (potentially irregularly shaped) regions, called tiling. The number of different tilings determines the resolution of the final approximation. For more examples, please refer to Sections 4.1 and 4.2.

Policy search also benefits from a compact representation of the policy as discussed in Section 4.3.

Models of the system dynamics can be represented using a wide variety of techniques. In this case, it is often important to model the uncertainty in the model (e.g., by a stochastic model or Bayesian estimates of model parameters) to ensure that the learn-

ing algorithm does not exploit model inaccuracies. See Section 6 for a more detailed discussion.

3 Challenges in Robot Reinforcement Learning

Reinforcement learning is generally a hard problem and many of its challenges are particularly apparent in the robotics setting. As the states and actions of most robots are inherently continuous, we are forced to consider the resolution at which they are represented. We must decide how fine grained the control is that we require over the robot, whether we employ discretization or function approximation, and what time step we establish. Additionally, as the dimensionality of both states and actions can be high, we face the "Curse of Dimensionality" (Bellman, 1957) as discussed in Section 3.1. As robotics deals with complex physical systems, samples can be expensive due to the long execution time of complete tasks, required manual interventions, and the need maintenance and repair. In these real-world measurements, we must cope with the uncertainty inherent in complex physical systems. A robot requires that the algorithm runs in real-time. The algorithm must be capable of dealing with delays in sensing and execution that are inherent in physical systems (see Section 3.2). A simulation might alleviate many problems but these approaches need to be robust with respect to model errors as discussed in Section 3.3. An often underestimated problem is the goal specification, which is achieved by designing a good reward function. As noted in Section 3.4, this choice can make the difference between feasibility and an unreasonable amount of exploration.

3.1 Curse of Dimensionality

When Bellman (1957) explored optimal control in discrete high-dimensional spaces, he faced an exponential explosion of states and actions for which he coined the term "Curse of Dimensionality". As the number of dimensions grows, exponentially more data and computation are needed to cover the complete state-action space. . For example, if we assume that each dimension of a state-space is discretized into ten levels, we have 10 states for a one-dimensional state-space, $10^3 = 1000$ unique states for a three-dimensional state-space, and 10^n possible states for a n-dimensional state space. Evaluating every state quickly becomes infeasible with growing dimensionality, even for discrete states. Bellman originally coined the term in the context of optimization, but it also applies to function approximation and numerical integration (Donoho, 2000). While supervised learning methods have tamed this exponential growth by considering only competitive optimality with respect to a limited class of function approximators, such results are much more difficult in reinforcement learning where data must collected throughout state-space to ensure global optimality.

Robotic systems often have to deal with these high dimensional states and actions due to the many degrees of freedom of modern anthropomorphic robots. For example, in the ball-paddling task shown in Figure 3, a proper representation of a robot's state would consist of its joint angles and velocities for each of its seven degrees of freedom as well as the Cartesian position and velocity of the ball. The robot's actions would be the
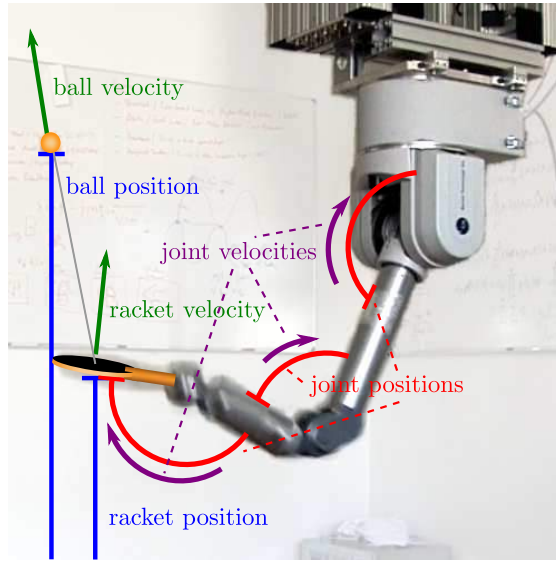
Figure 3: This Figure illustrates the state space used in the modeling of a robot reinforcement learning task of paddling a ball.

generated motor commands, which often are torques or accelerations. In this example, we have $2 \times (7+3) = 20$ state dimensions and 7-dimensional continuous actions. Obviously, other tasks may require even more dimensions. For example, human-like actuation often follows the antagonistic principle (Yamaguchi and Takanishi, 1997) which additionally enables control of stiffness. Such dimensionality is a major challenge for both the robotics and the reinforcement learning communities.

In robotics, such tasks are often rendered tractable to the robot engineer by a hierarchical task decomposition that shifts some complexity to a lower layer of functionality. Classical reinforcement learning approaches often consider a grid-based representation with discrete states and actions, often referred to as a *grid-world.* A navigational task for mobile robots could be projected into this representation by employing a number of actions like "move to the cell to the left" that use a lower level controller that takes care of accelerating, moving, and stopping while ensuring precision. In the ball-paddling example, we may simplify by controlling the robot in racket space (which is lower-dimensional as the racket is orientation-invariant around the string's mounting point) with an operational space control law (Nakanishi et al., 2008). Many commercial robot systems also encapsulate some of the state and action components in an embedded control system (e.g., trajectory fragments are frequently used as actions for industrial robots). However, this form of a state dimensionality reduction severely limits the dynamic capabilities of the robot according to our experience (Schaal et al., 2002; Peters et al., 2010b).

The reinforcement learning community has a long history of dealing with dimensionality using computational abstractions. It offers a larger set of applicable tools ranging from adaptive discretizations (Buşoniu et al., 2010) and function approximation approaches (Sutton and Barto, 1998) to macro-actions or options (Barto and Mahadevan, 2003; Hart

and Grupen, 2011). Options allow a task to be decomposed into elementary components and quite naturally translate to robotics. Such options can autonomously achieve a sub-task, such as opening a door, which reduces the planning horizon (Barto and Mahadevan, 2003). The automatic generation of such sets of options is a key issue in order to enable such approaches. We will discuss approaches that have been successful in robot reinforcement learning in Section 4.

3.2 Curse of Real-World Samples

Robots inherently interact with the physical world. Hence, robot reinforcement learning suffers from most of the resulting real-world problems. For example, robot hardware is usually expensive, suffers from wear and tear, and requires careful maintenance. Repairing a robot system is a non-negligible effort associated with cost, physical labor and long waiting periods. To apply reinforcement learning in robotics, safe exploration becomes a key issue of the learning process (Schneider, 1996; Bagnell, 2004; Deisenroth and Rasmussen, 2011; Moldovan and Abbeel, 2012), a problem often neglected in the general reinforcement learning community. Perkins and Barto (2002) have come up with a method for constructing reinforcement learning agents based on Lyapunov functions. Switching between the underlying controllers is always safe and offers basic performance guarantees.

However, several more aspects of the real-world make robotics a challenging domain. As the dynamics of a robot can change due to many external factors ranging from temperature to wear, the learning process may never fully converge, i.e., it needs a "tracking solution" (Sutton et al., 2007). Frequently, the environment settings during an earlier learning period cannot be reproduced. External factors are not always clear – for example, how light conditions affect the performance of the vision system and, as a result, the task's performance. This problem makes comparing algorithms particularly hard. Furthermore, the approaches often have to deal with uncertainty due to inherent measurement noise and the inability to observe all states directly with sensors.

Most real robot learning tasks require some form of human supervision, e.g., putting the pole back on the robot's end-effector during pole balancing (see Figure 1d) after a failure. Even when an automatic reset exists (e.g., by having a smart mechanism that resets the pole), learning speed becomes essential as a task on a real robot cannot be sped up. In some tasks like a slowly rolling robot, the dynamics can be ignored; in others like a flying robot, they cannot. Especially in the latter case, often the whole episode needs to be completed as it is not possible to start from arbitrary states.

For such reasons, real-world samples are expensive in terms of time, labor and, potentially, finances. In robotic reinforcement learning, it is often considered to be more important to limit the real-world interaction time instead of limiting memory consumption or computational complexity. Thus, sample efficient algorithms that are able to learn from a small number of trials are essential. In Section 6 we will point out several approaches that allow the amount of required real-world interactions to be reduced.

Since the robot is a physical system, there are strict constraints on the interaction between the learning algorithm and the robot setup. For dynamic tasks, the movement

cannot be paused and actions must be selected within a time-budget without the opportunity to pause to think, learn or plan between actions. These constraints are less severe in an episodic setting where the time intensive part of the learning can be postponed to the period between episodes. Hester et al. (2012) has proposed a real-time architecture for model-based value function reinforcement learning methods taking into account these challenges.

As reinforcement learning algorithms are inherently implemented on a digital computer, the discretization of time is unavoidable despite that physical systems are inherently continuous time systems. Time-discretization of the actuation can generate undesirable artifacts (e.g., the distortion of distance between states) even for idealized physical systems, which cannot be avoided. As most robots are controlled at fixed sampling frequencies (in the range between 500Hz and 3kHz) determined by the manufacturer of the robot, the upper bound on the rate of temporal discretization is usually pre-determined. The lower bound depends on the horizon of the problem, the achievable speed of changes in the state, as well as delays in sensing and actuation.

All physical systems exhibit such delays in sensing and actuation. The state of the setup (represented by the filtered sensor signals) may frequently lag behind the real state due to processing and communication delays. More critically, there are also communication delays in actuation as well as delays due to the fact that neither motors, gear boxes nor the body's movement can change instantly. Due to these delays, actions may not have instantaneous effects but are observable only several time steps later. In contrast, in most general reinforcement learning algorithms, the actions are assumed to take effect instantaneously as such delays would violate the usual Markov assumption. This effect can be addressed by putting some number of recent actions into the state. However, this significantly increases the dimensionality of the problem.

The problems related to time-budgets and delays can also be avoided by increasing the duration of the time steps. One downside of this approach is that the robot cannot be controlled as precisely; another is that it may complicate a description of system dynamics.

3.3 Curse of Under-Modeling and Model Uncertainty

One way to offset the cost of real-world interaction is to use accurate models as simulators. In an ideal setting, this approach would render it possible to learn the behavior in simulation and subsequently transfer it to the real robot. Unfortunately, creating a sufficiently accurate model of the robot and its environment is challenging and often requires very many data samples. As small model errors due to this under-modeling accumulate, the simulated robot can quickly diverge from the real-world system. When a policy is trained using an imprecise forward model as simulator, the behavior will not transfer without significant modifications as experienced by Atkeson (1994) when learning the underactuated pendulum swing-up. The authors have achieved a direct transfer in only a limited number of experiments; see Section 6.1 for examples.

For tasks where the system is self-stabilizing (that is, where the robot does not require active control to remain in a safe state or return to it), transferring policies often works

well. Such tasks often feature some type of dampening that absorbs the energy introduced by perturbations or control inaccuracies. If the task is inherently stable, it is safer to assume that approaches that were applied in simulation work similarly in the real world (Kober and Peters, 2010). Nevertheless, tasks can often be learned better in the real world than in simulation due to complex mechanical interactions (including contacts and friction) that have proven difficult to model accurately. For example, in the ball-paddling task (Figure 3) the elastic string that attaches the ball to the racket always pulls back the ball towards the racket even when hit very hard. Initial simulations (including friction models, restitution models, dampening models, models for the elastic string, and air drag) of the ball-racket contacts indicated that these factors would be very hard to control. In a real experiment, however, the reflections of the ball on the racket proved to be less critical than in simulation and the stabilizing forces due to the elastic string were sufficient to render the whole system self-stabilizing.

In contrast, in unstable tasks small variations have drastic consequences. For example, in a pole balancing task, the equilibrium of the upright pole is very brittle and constant control is required to stabilize the system. Transferred policies often perform poorly in this setting. Nevertheless, approximate models serve a number of key roles which we discuss in Section 6, including verifying and testing the algorithms in simulation, establishing proximity to theoretically optimal solutions, calculating approximate gradients for local policy improvement, identifing strategies for collecting more data, and performing "mental rehearsal".

3.4 Curse of Goal Specification

In reinforcement learning, the desired behavior is implicitly specified by the reward function. The goal of reinforcement learning algorithms then is to maximize the accumulated long-term reward. While often dramatically simpler than specifying the behavior itself, in practice, it can be surprisingly difficult to define a good reward function in robot reinforcement learning. The learner must observe variance in the reward signal in order to be able to improve a policy: if the same return is always received, there is no way to determine which policy is better or closer to the optimum.

In many domains, it seems natural to provide rewards only upon task achievement – for example, when a table tennis robot wins a match. This view results in an apparently simple, binary reward specification. However, a robot may receive such a reward so rarely that it is unlikely to ever succeed in the lifetime of a real-world system. Instead of relying on simpler binary rewards, we frequently need to include intermediate rewards in the scalar reward function to guide the learning process to a reasonable solution, a process known as *reward shaping* (Laud, 2004).

Beyond the need to shorten the effective problem horizon by providing intermediate rewards, the trade-off between different factors may be essential. For instance, hitting a table tennis ball very hard may result in a high score but is likely to damage a robot or shorten its life span. Similarly, changes in actions may be penalized to avoid high frequency controls that are likely to be very poorly captured with tractable low dimensional state-space or rigid-body models. Reinforcement learning algorithms are also notorious

for exploiting the reward function in ways that are not anticipated by the designer. For example, if the distance between the ball and the desired highest point is part of the reward in ball paddling (see Figure 3), many locally optimal solutions would attempt to simply move the racket upwards and keep the ball on it. Reward shaping gives the system a notion of closeness to the desired behavior instead of relying on a reward that only encodes success or failure (Ng et al., 1999).

Often the desired behavior can be most naturally represented with a reward function in a particular state and action space. However, this representation does not necessarily correspond to the space where the actual learning needs to be performed due to both computational and statistical limitations. Employing methods to render the learning problem tractable often result in different, more abstract state and action spaces which might not allow accurate representation of the original reward function. In such cases, a rewardartfully specifiedin terms of the features of the space in which the learning algorithm operates can prove remarkably effective. There is also a trade-off between the complexity of the reward function and the complexity of the learning problem. For example, in the ball-in-a-cup task (Section 7) the most natural reward would be a binary value depending on whether the ball is in the cup or not. To render the learning problem tractable, a less intuitive reward needed to be devised in terms of a Cartesian distance with additional directional information (see Section 7.1 for details). Another example is Crusher (Ratliff et al., 2006a), an outdoor robot, where the human designer was interested in a combination of minimizing time and risk to the robot. However, the robot reasons about the world on the long time horizon scale as if it was a very simple, deterministic, holonomic robot operating on a fine grid of continuous costs. Hence, the desired behavior cannot be represented straightforwardly in this state-space. Nevertheless, a remarkably human-like behavior that seems to respect time and risk priorities can be achieved by carefully mapping features describing each state (discrete grid location with features computed by an on-board perception system) to cost.

Inverse optimal control, also known as inverse reinforcement learning (Russell, 1998), is a promising alternative to specifying the reward function manually. It assumes that a reward function can be reconstructed from a set of expert demonstrations. This reward function does not necessarily correspond to the true reward function, but provides guarantees on the resulting performance of learned behaviors (Abbeel and Ng, 2004; Ratliff et al., 2006b). Inverse optimal control was initially studied in the control community (Kalman, 1964) and in the field of economics (Keeney and Raiffa, 1976). The initial results were only applicable to limited domains (linear quadratic regulator problems) and required closed form access to plant and controller, hence samples from human demonstrations could not be used. Russell (1998) brought the field to the attention of the machine learning community. Abbeel and Ng (2004) defined an important constraint on the solution to the inverse RL problem when reward functions are linear in a set of features: a policy that is extracted by observing demonstrations has to earn the same reward as the policy that is being demonstrated. Ratliff et al. (2006b) demonstrated that inverse optimal control can be understood as a generalization of ideas in machine learning of *structured prediction* and introduced efficient sub-gradient based algorithms with regret bounds that enabled large scale application of the technique within robotics. Ziebart

et al. (2008) extended the technique developed by Abbeel and Ng (2004) by rendering the idea robust and probabilistic, enabling its effective use for both learning policies and predicting the behavior of sub-optimal agents. These techniques, and many variants, have been recently successfully applied to outdoor robot navigation (Ratliff et al., 2006a; Silver et al., 2008, 2010), manipulation (Ratliff et al., 2007), and quadruped locomotion (Ratliff et al., 2006a, 2007; Kolter et al., 2007).

More recently, the notion that complex policies can be built on top of simple, easily solved optimal control problems by exploiting rich, parametrized reward functions has been exploited within reinforcement learning more directly. In (Sorg et al., 2010; Zucker and Bagnell, 2012), complex policies are derived by adapting a reward function for simple optimal control problems using policy search techniques. Zucker and Bagnell (2012) demonstrate that this technique can enable efficient solutions to robotic marble-maze problems that effectively transfer between mazes of varying design and complexity. These works highlight the natural trade-off between the complexity of the reward function and the complexity of the underlying reinforcement learning problem for achieving a desired behavior.

4 Tractability Through Representation

As discussed above, reinforcement learning provides a framework for a remarkable variety of problems of significance to both robotics and machine learning. However, the computational and information-theoretic consequences that we outlined above accompany this power and generality. As a result, naive application of reinforcement learning techniques in robotics is likely to be doomed to failure. The remarkable successes that we reference in this article have been achieved by leveraging a few key principles – effective representations, approximate models, and prior knowledge or information. In the following three sections, we review these principles and summarize how each has been made effective in practice. We hope that understanding these broad approaches will lead to new successes in robotic reinforcement learning by combining successful methods and encourage research on novel techniques that embody each of these principles.

Much of the success of reinforcement learning methods has been due to the clever use of approximate representations. The need of such approximations is particularly pronounced in robotics, where table-based representations (as discussed in Section 2.2.1) are rarely scalable. The different ways of making reinforcement learning methods tractable in robotics are tightly coupled to the underlying optimization framework. Reducing the dimensionality of states or actions by smart state-action discretization is a representational simplification that may enhance both policy search and value function-based methods (see 4.1). A value function-based approach requires an accurate and robust but general function approximator that can capture the value function with sufficient precision (see Section 4.2) while maintaining stability during learning. Policy search methods require a choice of policy representation that controls the complexity of representable policies to enhance learning speed (see Section 4.3). An overview of publications that make particular use of efficient representations to render the learning problem tractable is presented

in Table 3.

4.1 Smart State-Action Discretization

Decreasing the dimensionality of state or action spaces eases most reinforcement learning problems significantly, particularly in the context of robotics. Here, we give a short overview of different attempts to achieve this goal with smart discretization.

Hand Crafted Discretization. A variety of authors have manually developed discretizations so that basic tasks can be learned on real robots. For low-dimensional tasks, we can generate discretizations straightforwardly by splitting each dimension into a number of regions. The main challenge is to find the right number of regions for each dimension that allows the system to achieve a good final performance while still learning quickly. Example applications include balancing a ball on a beam (Benbrahim et al., 1992), one degree of freedom ball-in-a-cup (Nemec et al., 2010), two degree of freedom crawling motions (Tokic et al., 2009), and gait patterns for four legged walking (Kimura et al., 2001). Much more human experience is needed for more complex tasks. For example, in a basic navigation task with noisy sensors (Willgoss and Iqbal, 1999), only some combinations of binary state or action indicators are useful (e.g., you can drive left and forward at the same time, but not backward and forward). The state space can also be based on vastly different features, such as positions, shapes, and colors, when learning object affordances (Paletta et al., 2007) where both the discrete sets and the mapping from sensor values to the discrete values need to be crafted. Kwok and Fox (2004) use a mixed discrete and continuous representation of the state space to learn active sensing strategies in a RoboCup scenario. They first discretize the state space along the dimension with the strongest non-linear influence on the value function and subsequently employ a linear value function approximation (Section 4.2) for each of the regions.

Learned from Data. Instead of specifying the discretizations by hand, they can also be built adaptively during the learning process. For example, a rule based reinforcement learning approach automatically segmented the state space to learn a cooperative task with mobile robots (Yasuda and Ohkura, 2008). Each rule is responsible for a local region of the state-space. The importance of the rules are updated based on the rewards and irrelevant rules are discarded. If the state is not covered by a rule yet, a new one is added. In the related field of computer vision, Piater et al. (2011) propose an approach that adaptively and incrementally discretizes a perceptual space into discrete states, training an image classifier based on the experience of the RL agent to distinguish visual classes, which correspond to the states.

Meta-Actions. Automatic construction of meta-actions (and the closely related concept of options) has fascinated reinforcement learning researchers and there are various examples in the literature. The idea is to have more intelligent actions that are composed of a sequence of movements and that in themselves achieve a simple task. A simple example would be to have a meta-action "move forward 5m." A lower level system takes care of

SMART STATE-ACTION DISCRETIZATION

Approach	Employed by...
Hand crafted	Benbrahim et al. (1992); Kimura et al. (2001); Kwok and Fox (2004); Nemec et al. (2010); Paletta et al. (2007); Tokic et al. (2009); Willgoss and Iqbal (1999)
Learned	Piater et al. (2011); Yasuda and Ohkura (2008)
Meta-actions	Asada et al. (1996); Dorigo and Colombetti (1993); Fidelman and Stone (2004); Huber and Grupen (1997); Kalmár et al. (1998); Konidaris et al. (2011a, 2012); Matarić (1994, 1997); Platt et al. (2006); Soni and Singh (2006); Nemec et al. (2009)
Relational Representation	Cocora et al. (2006); Katz et al. (2008)

VALUE FUNCTION APPROXIMATION

Approach	Employed by...
Physics-inspired Features	An et al. (1988); Schaal (1996)
Neural Networks	Benbrahim and Franklin (1997); Duan et al. (2008); Gaskett et al. (2000); Hafner and Riedmiller (2003); Riedmiller et al. (2009); Thrun (1995)
Neighbors	Hester et al. (2010); Mahadevan and Connell (1992); Touzet (1997)
Local Models	Bentivegna (2004); Schaal (1996); Smart and Kaelbling (1998)
GPR	Gräve et al. (2010); Kroemer et al. (2009, 2010); Rottmann et al. (2007)

PRE-STRUCTURED POLICIES

Approach	Employed by...
Via Points & Splines	Kuindersma et al. (2011); Miyamoto et al. (1996); Roberts et al. (2010)
Linear Models	Tamei and Shibata (2009)
Motor Primitives	Kohl and Stone (2004); Kober and Peters (2009); Peters and Schaal (2008c,b); Stulp et al. (2011); Tamošiūnaitė et al. (2011); Theodorou et al. (2010)
GMM & LLM	Deisenroth and Rasmussen (2011); Deisenroth et al. (2011); Guenter et al. (2007); Lin and Lai (2012); Peters and Schaal (2008a)
Neural Networks	Endo et al. (2008); Geng et al. (2006); Gullapalli et al. (1994); Hailu and Sommer (1998); Bagnell and Schneider (2001)
Controllers	Bagnell and Schneider (2001); Kolter and Ng (2009a); Tedrake (2004); Tedrake et al. (2005); Vlassis et al. (2009); Zucker and Bagnell (2012)
Non-parametric	Kober et al. (2010); Mitsunaga et al. (2005); Peters et al. (2010a)

Table 3: This table illustrates different methods of making robot reinforcement learning tractable by employing a suitable representation.

accelerating, stopping, and correcting errors. For example, in (Asada et al., 1996), the state and action sets are constructed in a way that repeated action primitives lead to a change in the state to overcome problems associated with the discretization. *Q*-learning and dynamic programming based approaches have been compared in a pick-n-place task (Kalmár et al., 1998) using modules. Huber and Grupen (1997) use a set of controllers with associated predicate states as a basis for learning turning gates with a quadruped. Fidelman and Stone (2004) use a policy search approach to learn a small set of parameters that controls the transition between a walking and a capturing meta-action in a RoboCup scenario. A task of transporting a ball with a dog robot (Soni and Singh, 2006) can be learned with semi-automatically discovered options. Using only the subgoals of primitive motions, a humanoid robot can learn a pouring task (Nemec et al., 2009). Other examples include foraging (Matarić, 1997) and cooperative tasks (Matarić, 1994) with multiple robots, grasping with restricted search spaces (Platt et al., 2006), and mobile robot navigation (Dorigo and Colombetti, 1993). If the meta-actions are not fixed in advance, but rather learned at the same time, these approaches are hierarchical reinforcement learning approaches as discussed in Section 5.2. Konidaris et al. (2011a, 2012) propose an approach that constructs a skill tree from human demonstrations. Here, the skills correspond to options and are chained to learn a mobile manipulation skill.

Relational Representations. In a relational representation, the states, actions, and transitions are not represented individually. Entities of the same predefined type are grouped and their relationships are considered. This representation may be preferable for highly geometric tasks (which frequently appear in robotics) and has been employed to learn to navigate buildings with a real robot in a supervised setting (Cocora et al., 2006) and to manipulate articulated objects in simulation (Katz et al., 2008).

4.2 Value Function Approximation

Function approximation has always been the key component that allowed value function methods to scale into interesting domains. In robot reinforcement learning, the following function approximation schemes have been popular and successful. Using function approximation for the value function can be combined with using function approximation for learning a model of the system (as discussed in Section 6) in the case of model-based reinforcement learning approaches.

Unfortunately the max-operator used within the Bellman equation and temporal-difference updates can theoretically make most linear or non-linear approximation schemes unstable for either value iteration or policy iteration. Quite frequently such an unstable behavior is also exhibited in practice. Linear function approximators are stable for policy evaluation, while non-linear function approximation (e.g., neural networks) can even diverge if just used for policy evaluation (Tsitsiklis and Van Roy, 1997).

Physics-inspired Features. If good hand-crafted features are known, value function approximation can be accomplished using a linear combination of features. However, good features are well known in robotics only for a few problems, such as features for
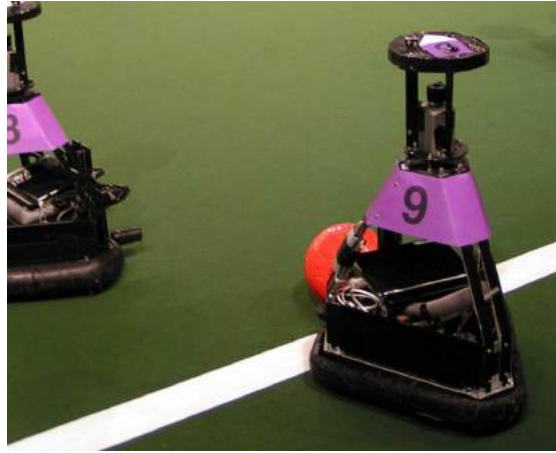
Figure 4: The Brainstormer Tribots won the RoboCup 2006 MidSize League (Riedmiller et al., 2009)(Picture reprint with permission of Martin Riedmiller).

local stabilization (Schaal, 1996) and features describing rigid body dynamics (An et al., 1988). Stabilizing a system at an unstable equilibrium point is the most well-known example, where a second order Taylor expansion of the state together with a linear value function approximator often suffice as features in the proximity of the equilibrium point. For example, Schaal (1996) showed that such features suffice for learning how to stabilize a pole on the end-effector of a robot when within $\pm 15 - 30$ degrees of the equilibrium angle. For sufficient features, linear function approximation is likely to yield good results in an on-policy setting. Nevertheless, it is straightforward to show that impoverished value function representations (e.g., omitting the cross-terms in quadratic expansion in Schaal's set-up) will make it impossible for the robot to learn this behavior. Similarly, it is well known that linear value function approximation is unstable in the off-policy case (Tsitsiklis and Van Roy, 1997; Gordon, 1999; Sutton and Barto, 1998).

Neural Networks. As good hand-crafted features are rarely available, various groups have employed neural networks as global, non-linear value function approximation. Many different flavors of neural networks have been applied in robotic reinforcement learning. For example, multi-layer perceptrons were used to learn a wandering behavior and visual servoing (Gaskett et al., 2000). Fuzzy neural networks (Duan et al., 2008) and explanation-based neural networks (Thrun, 1995) have allowed robots to learn basic navigation. CMAC neural networks have been used for biped locomotion (Benbrahim and Franklin, 1997).

The Brainstormers RoboCup soccer team is a particularly impressive application of value function approximation.(see Figure 4). It used multi-layer perceptrons to learn various sub-tasks such as learning defenses, interception, position control, kicking, motor speed control, dribbling and penalty shots (Hafner and Riedmiller, 2003; Riedmiller et al., 2009). The resulting components contributed substantially to winning the world cup several times in the simulation and the mid-size real robot leagues. As neural networks

are global function approximators, overestimating the value function at a frequently occurring state will increase the values predicted by the neural network for all other states, causing fast divergence (Boyan and Moore, 1995; Gordon, 1999).Riedmiller et al. (2009) solved this problem by always defining an absorbing state where they set the value predicted by their neural network to zero, which "clamps the neural network down" and thereby prevents divergence. It also allows re-iterating on the data, which results in an improved value function quality. The combination of iteration on data with the clamping technique appears to be the key to achieving good performance with value function approximation in practice.

Generalize to Neighboring Cells. As neural networks are globally affected from local errors, much work has focused on simply generalizing from neighboring cells. One of the earliest papers in robot reinforcement learning (Mahadevan and Connell, 1992) introduced this idea by statistically clustering states to speed up a box-pushing task with a mobile robot, see Figure 1a. This approach was also used for a navigation and obstacle avoidance task with a mobile robot (Touzet, 1997). Similarly, decision trees have been used to generalize states and actions to unseen ones, e.g., to learn a penalty kick on a humanoid robot (Hester et al., 2010). The core problem of these methods is the lack of scalability to high-dimensional state and action spaces.

Local Models. Local models can be seen as an extension of generalization among neighboring cells to generalizing among neighboring data points. Locally weighted regression creates particularly efficient function approximation in the context of robotics both in supervised and reinforcement learning. Here, regression errors are weighted down by proximity to query point to train local modelsThe predictions of these local models are combined using the same weighting functions. Using local models for value function approximation has allowed learning a navigation task with obstacle avoidance (Smart and Kaelbling, 1998), a pole swing-up task (Schaal, 1996), and an air hockey task (Bentivegna, 2004).

Gaussian Process Regression. Parametrized global or local models need to pre-specify, which requires a trade-off between representational accuracy and the number of parameters. A non-parametric function approximator like Gaussian Process Regression (GPR) could be employed instead, but potentially at the cost of a higher computational complexity. GPR has the added advantage of providing a notion of uncertainty about the approximation quality for a query point. Hovering with an autonomous blimp (Rottmann et al., 2007) has been achieved by approximation the state-action value function with a GPR. Similarly, another paper shows that grasping can be learned using Gaussian process regression (Gräve et al., 2010) by additionally taking into account the uncertainty to guide the exploration. Grasping locations can be learned by approximating the rewards with a GPR, and trying candidates with predicted high rewards (Kroemer et al., 2009), resulting in an active learning approach. High reward uncertainty allows intelligent exploration in reward-based grasping (Kroemer et al., 2010) in a bandit setting.
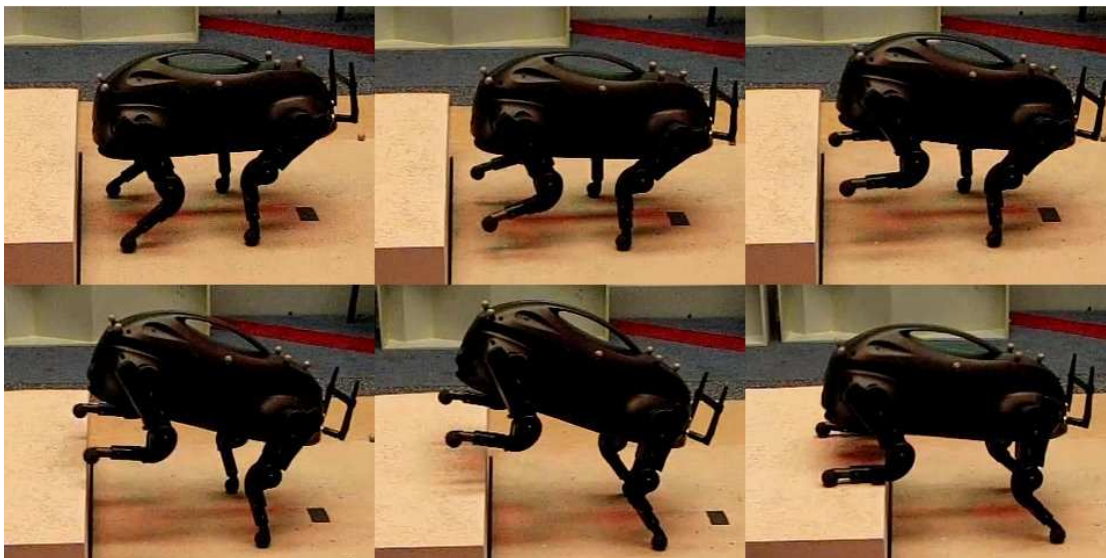
Figure 5: Boston Dynamics LittleDog jumping (Kolter and Ng, 2009a) (Picture reprint with permission of Zico Kolter).

4.3 Pre-structured Policies

Policy search methods greatly benefit from employing an appropriate function approximation of the policy. For example, when employing gradient-based approaches, the trade-off between the representational power of the policy (in the form of many policy parameters) and the learning speed (related to the number of samples required to estimate the gradient) needs to be considered. To make policy search approaches tractable, the policy needs to be represented with a function approximation that takes into account domain knowledge, such as task-relevant parameters or generalization properties. As the next action picked by a policy depends on the current state and action, a policy can be seen as a closed-loop controller. Roberts et al. (2011) demonstrate that care needs to be taken when selecting closed-loop parameterizations for weakly-stable systems, and suggest forms that are particularly robust during learning. However, especially for episodic RL tasks, sometimes open-loop policies (i.e., policies where the actions depend only on the time) can also be employed.

Via Points & Splines. An open-loop policy may often be naturally represented as a trajectory, either in the space of states or targets or directly as a set of controls. Here, the actions are only a function of time, which can be considered as a component of the state. Such spline-based policies are very suitable for compressing complex trajectories into few parameters. Typically the desired joint or Cartesian position, velocities, and/or accelerations are used as actions. To minimize the required number of parameters, not every point is stored. Instead, only important via-points are considered and other points are interpolated. Miyamoto et al. (1996) optimized the position and timing of such

via-points in order to learn a kendama task (a traditional Japanese toy similar to ball-in-a-cup). A well known type of a via point representations are splines, which rely on piecewise-defined smooth polynomial functions for interpolation. For example, Roberts et al. (2010) used a periodic cubic spline as a policy parametrization for a flapping system and Kuindersma et al. (2011) used a cubic spline to represent arm movements in an impact recovery task.

Linear Models. If model knowledge of the system is available, it can be used to create features for linear closed-loop policy representations. For example, Tamei and Shibata (2009) used policy-gradient reinforcement learning to adjust a model that maps from human EMG signals to forces that in turn is used in a cooperative holding task.

Motor Primitives. Motor primitives combine linear models describing dynamics with parsimonious movement parametrizations. While originally biologically-inspired, they have a lot of success for representing basic movements in robotics such as a reaching movement or basic locomotion. These basic movements can subsequently be sequenced and/or combined to achieve more complex movements. For both goal oriented and rhythmic movement, different technical representations have been proposed in the robotics community. Dynamical system motor primitives (Ijspeert et al., 2003; Schaal et al., 2007) have become a popular representation for reinforcement learning of discrete movements. The dynamical system motor primitives always have a strong dependence on the phase of the movement, which corresponds to time. They can be employed as an open-loop trajectory representation. Nevertheless, they can also be employed as a closed-loop policy to a limited extent. In our experience, they offer a number of advantages over via-point or spline based policy representation (see Section 7.2). The dynamical system motor primitives have been trained with reinforcement learning for a T-ball batting task (Peters and Schaal, 2008c,b), an underactuated pendulum swing-up and a ball-in-a-cup task (Kober and Peters, 2009), flipping a light switch (Buchli et al., 2011), pouring water (Tamošiūnaitė et al., 2011), and playing pool and manipulating a box (Pastor et al., 2011). For rhythmic behaviors, a representation based on the same biological motivation but with a fairly different technical implementation (based on half-elliptical locuses) have been used to acquire the gait patterns for an Aibo robot dog locomotion (Kohl and Stone, 2004).

Gaussian Mixture Models and Radial Basis Function Models. When more general policies with a strong state-dependence are needed, general function approximators based on radial basis functions, also called Gaussian kernels, become reasonable choices. While learning with fixed basis function centers and widths often works well in practice, estimating them is challenging. These centers and widths can also be estimated from data prior to the reinforcement learning process. This approach has been used to generalize a open-loop reaching movement (Guenter et al., 2007; Lin and Lai, 2012) and to learn the closed-loop cart-pole swingup task (Deisenroth and Rasmussen, 2011). Globally linear models were employed in a closed-loop block stacking task (Deisenroth et al., 2011).

Neural Networks are another general function approximation used to represent policies. Neural oscillators with sensor feedback have been used to learn rhythmic movements where open and closed-loop information were combined, such as gaits for a two legged robot (Geng et al., 2006; Endo et al., 2008). Similarly, a peg-in-hole (see Figure 1b), a ball-balancing task (Gullapalli et al., 1994), and a navigation task (Hailu and Sommer, 1998) have been learned with closed-loop neural networks as policy function approximators.

Locally Linear Controllers. As local linearity is highly desirable in robot movement generation to avoid actuation difficulties, learning the parameters of a locally linear controller can be a better choice than using a neural network or radial basis function representation. Several of these controllers can be combined to form a global, inherently closed-loop policy. This type of policy has allowed for many applications, including learning helicopter flight (Bagnell and Schneider, 2001), learning biped walk patterns (Tedrake, 2004; Tedrake et al., 2005), driving a radio-controlled (RC) car, learning a jumping behavior for a robot dog (Kolter and Ng, 2009a) (illustrated in Figure 5), and balancing a two wheeled robot (Vlassis et al., 2009). Operational space control was also learned by Peters and Schaal (2008a) using locally linear controller models. In a marble maze task, Zucker and Bagnell (2012) used such a controller as a policy that expressed the desired velocity of the ball in terms of the directional gradient of a value function.

Non-parametric Policies. Polices based on non-parametric regression approaches often allow a more data-driven learning process. This approach is often preferable over the purely parametric policies listed above becausethe policy structure can evolve during the learning process. Such approaches are especially useful when a policy learned to adjust the existing behaviors of an lower-level controller, such as when choosing among different robot human interaction possibilities (Mitsunaga et al., 2005), selecting among different striking movements in a table tennis task (Peters et al., 2010a), and setting the meta-actions for dart throwing and table tennis hitting tasks (Kober et al., 2010).

5 Tractability Through Prior Knowledge

Prior knowledge can dramatically help guide the learning process. It can be included in the form of initial policies, demonstrations, initial models, a predefined task structure, or constraints on the policy such as torque limits or ordering constraints of the policy parameters. These approaches significantly reduce the search space and, thus, speed up the learning process. Providing a (partially) successful initial policy allows a reinforcement learning method to focus on promising regions in the value function or in policy space, see Section 5.1. Pre-structuring a complex task such that it can be broken down into several more tractable ones can significantly reduce the complexity of the learning task, see Section 5.2. An overview of publications using prior knowledge to render the learning problem tractable is presented in Table 4. Constraints may also limit the search space, but often pose new, additional problems for the learning methods. For example,

policy search limits often do not handle hard limits on the policy well. Relaxing such constraints (a trick often applied in machine learning) is not feasible if they were introduced to protect the robot in the first place.

5.1 Prior Knowledge Through Demonstration

People and other animals frequently learn using a combination of imitation and trial and error. When learning to play tennis, for instance, an instructor will repeatedly demonstrate the sequence of motions that form an orthodox forehand stroke. Students subsequently imitate this behavior, but still need hours of practice to successfully return balls to a precise location on the opponent's court. Input from a teacher need not be limited to initial instruction. The instructor may provide additional demonstrations in later learning stages (Latzke et al., 2007; Ross et al., 2011a) and which can also be used as differential feedback (Argall et al., 2008).

This combination of imitation learning with reinforcement learning is sometimes termed *apprenticeship learning* (Abbeel and Ng, 2004) to emphasize the need for learning both from a teacher and by practice. The term "apprenticeship learning" is often employed to refer to "inverse reinforcement learning" or "inverse optimal control" but is intended here to be employed in this original, broader meaning. For a recent survey detailing the state of the art in imitation learning for robotics, see (Argall et al., 2009).

Using demonstrations to initialize reinforcement learning provides multiple benefits. Perhaps the most obvious benefit is that it provides supervised training data of what actions to perform in states that are encountered. Such data may be helpful when used to bias policy action selection.

The most dramatic benefit, however, is that demonstration – or a hand-crafted initial policy – removes the need for *global exploration* of the policy or state-space of the RL problem. The student can improve by locally optimizing a policy knowing what states are important, making local optimization methods feasible. Intuitively, we expect that removing the demands of global exploration makes learning easier. However, we can only find local optima close to the demonstration, that is, we rely on the demonstration to provide a good starting point. Perhaps the textbook example of such in human learning is the rise of the "Fosbury Flop" (Wikipedia, 2013) method of high-jump (see Figure 6). This motion is very different from a classical high-jump and took generations of Olympians to discover. But after it was first demonstrated, it was soon mastered by virtually all athletes participating in the sport. On the other hand, this example also illustrates nicely that such local optimization around an initial demonstration can only find local optima.

In practice, both approximate value function based approaches and policy search methods work best for real system applications when they are constrained to make modest changes to the distribution over states while learning. Policy search approaches implicitly maintain the state distribution by limiting the changes to the policy. On the other hand, for value function methods, an unstable estimate of the value function can lead to drastic changes in the policy. Multiple policy search methods used in robotics are based on this intuition (Bagnell and Schneider, 2003; Peters and Schaal, 2008b; Peters et al.,

Figure 6: This figure illustrates the "Fosbury Flop" (public domain picture from Wikimedia Commons).

2010a; Kober and Peters, 2010).

The intuitive idea and empirical evidence that demonstration makes the reinforcement learning problem simpler can be understood rigorously. In fact, Kakade and Langford (2002); Bagnell et al. (2003) demonstrate that knowing approximately the state-distribution of a good policy[5] transforms the problem of reinforcement learning from one that is provably intractable in both information and computational complexity to a tractable one with only polynomial sample and computational complexity, even under function approximation and partial observability. This type of approach can be understood as a reduction from reinforcement learning to supervised learning. Both algorithms are policy search variants of approximate policy iteration that constrain policy updates. Kollar and Roy (2008) demonstrate the benefit of this RL approach for developing state-of-the-art map exploration policies and Kolter et al., 2008 employed a space-indexed variant to learn trajectory following tasks with an autonomous vehicle and a RC car.

Demonstrations by a Teacher. Demonstrations by a teacher can be obtained in two different scenarios. In the first, the teacher demonstrates the task using his or her own body; in the second, the teacher controls the robot to do the task. The first scenario is limited by the embodiment issue, as the movement of a human teacher usually cannot be mapped directly to the robot due to different physical constraints and capabilities. For example, joint angles of a human demonstrator need to be adapted to account for the kinematic differences between the teacher and the robot. Often it is more advisable to only consider task-relevant information, such asthe Cartesian positions and velocities of the end-effector and the object. Demonstrations obtained by motion-capture have been used to learn a pendulum swingup (Atkeson and Schaal, 1997), ball-in-a-cup (Kober et al., 2008) and grasping (Gräve et al., 2010).

The second scenario obtains demonstrations by a human teacher directly controlling the robot. Here the human teacher first has to learn how to achieve a task with the particular robot's hardware, adding valuable prior knowledge. For example, remotely

---

[5]That is, a probability distribution over states that will be encountered when following a good policy.

controlling the robot initialized a Q-table for a navigation task (Conn and Peters II, 2007). If the robot is back-drivable, kinesthetic teach-in (i.e., by taking it by the hand and moving it) can be employed, which enables the teacher to interact more closely with the robot. This method has resulted in applications including T-ball batting (Peters and Schaal, 2008c,b), reaching tasks (Guenter et al., 2007; Bitzer et al., 2010), ball-in-a-cup (Kober and Peters, 2009), flipping a light switch (Buchli et al., 2011), playing pool and manipulating a box (Pastor et al., 2011), and opening a door and picking up objects (Kalakrishnan et al., 2011). A marble maze task can be learned using demonstrations by a human player (Bentivegna et al., 2004).

One of the more stunning demonstrations of the benefit of learning from a teacher is the helicopter airshows of (Coates et al., 2009). This approach combines initial human demonstration of trajectories, machine learning to extract approximate models from multiple trajectories, and classical locally-optimal control methods (Jacobson and Mayne, 1970) to achieve state-of-the-art acrobatic flight.

Hand-Crafted Policies. When human interaction with the system is not straightforward due to technical reasons or human limitations, a pre-programmed policy can provide alternative demonstrations. For example, a vision-based mobile robot docking task can be learned faster with such a basic behavior than using Q-learning alone, as demonstrated in (Martínez-Marín and Duckett, 2005) Providing hand-coded, stable initial gaits can significantly help in learning robot locomotion, as shown on a six-legged robot (Erden and Leblebicioğlu, 2008) as well as on a biped (Tedrake, 2004; Tedrake et al., 2005). Alternatively, hand-crafted policies can yield important corrective actions as prior knowledge that prevent the robot to deviates significantly from the desired behavior and endanger itself. This approach has been applied to adapt the walking patterns of a robot dog to new surfaces (Birdwell and Livingston, 2007) by Q-learning. Rosenstein and Barto (2004) employed a stable controller to teach the robot about favorable actions and avoid risky behavior while learning to move from a start to a goal position.

5.2 Prior Knowledge Through Task Structuring

Often a task can be decomposed hierarchically into basic components or into a sequence of increasingly difficult tasks. In both cases the complexity of the learning task is significantly reduced.

Hierarchical Reinforcement Learning. A task can often be decomposed into different levels. For example when using meta-actions (Section 4.1), these meta-actions correspond to a lower level dealing with the execution of sub-tasks which are coordinated by a strategy level. Hierarchical reinforcement learning does not assume that all but one levels are fixed but rather learns all of them simultaneously. For example, hierarchical Q-learning has been used to learn different behavioral levels for a six legged robot: moving single legs, locally moving the complete body, and globally moving the robot towards a goal (Kirchner, 1997). A stand-up behavior considered as a hierarchical reinforcement learning task has been learned using Q-learning in the upper-level and a

DEMONSTRATION

Approach	Employed by...
Teacher	Atkeson and Schaal (1997); Bentivegna et al. (2004); Bitzer et al. (2010); Conn and Peters II (2007); Gräve et al. (2010); Kober et al. (2008); Kober and Peters (2009); Latzke et al. (2007); Peters and Schaal (2008c,b)
Policy	Birdwell and Livingston (2007); Erden and Leblebicioğlu (2008); Martínez-Marín and Duckett (2005); Rosenstein and Barto (2004); Smart and Kaelbling (1998); Tedrake (2004); Tedrake et al. (2005); Wang et al. (2006)

TASK STRUCTURE

Approach	Employed by...
Hierarchical	Daniel et al. (2012); Donnart and Meyer (1996); Hart and Grupen (2011); Huber and Grupen (1997); Kirchner (1997); Morimoto and Doya (2001); Muelling et al. (2012); Whitman and Atkeson (2010)
Progressive Tasks	Asada et al. (1996); Randløv and Alstrøm (1998)

DIRECTED EXPLORATION WITH PRIOR KNOWLEDGE

Approach	Employed by...
Directed Exploration	Huang and Weng (2002); Kroemer et al. (2010); Pendrith (1999)

Table 4: This table illustrates different methods of making robot reinforcement learning tractable by incorporating prior knowledge.

continuous actor-critic method in the lower level (Morimoto and Doya, 2001). Navigation in a maze can be learned using an actor-critic architecture by tuning the influence of different control modules and learning these modules (Donnart and Meyer, 1996). Huber and Grupen (1997) combine discrete event system and reinforcement learning techniques to learn turning gates for a quadruped. Hart and Grupen (2011) learn to bi-manual manipulation tasks by assembling policies hierarchically. Daniel et al. (2012) learn options in a tetherball scenario and Muelling et al. (2012) learn different strokes in a table tennis scenario. Whitman and Atkeson (2010) show that the optimal policy for some global systems (like a walking controller) can be constructed by finding the optimal controllers for simpler subsystems and coordinating these.

Progressive Tasks. Often complicated tasks are easier to learn if simpler tasks can already be performed. This progressive task development is inspired by how biological systems learn. For example, a baby first learns how to roll, then how to crawl, then how to walk. A sequence of increasingly difficult missions has been employed to learn a goal shooting task in (Asada et al., 1996) using Q-learning. Randløv and Alstrøm (1998) discuss shaping the reward function to include both a balancing and a goal oriented term for a simulated bicycle riding task. The reward is constructed in such a way that the balancing term dominates the other term and, hence, this more fundamental behavior is learned first.

5.3 Directing Exploration with Prior Knowledge

As discussed in Section 2.1, balancing exploration and exploitation is an important consideration. Task knowledge can be employed to guide to robots curiosity to focus on regions that are novel and promising at the same time. For example, a mobile robot learns to direct attention by employing a modified Q-learning approach using novelty (Huang and Weng, 2002). Using "corrected truncated returns" and taking into account the estimator variance, a six legged robot employed with stepping reflexes can learn to walk (Pendrith, 1999). Offline search can be used to guide Q-learning during a grasping task (Wang et al., 2006). Using upper confidence bounds (Kaelbling, 1990) to direct exploration into regions with potentially high rewards, grasping can be learned efficiently (Kroemer et al., 2010).

6 Tractability Through Models

In Section 2, we discussed robot reinforcement learning from a model-free perspective where the system simply served as a data generating process. Such model-free reinforcement algorithms try to directly learn the value function or the policy without any explicit modeling of the transition dynamics. In contrast, many robot reinforcement learning problems can be made tractable by learning forward models, i.e., approximations of the transition dynamics based on data. Such model-based reinforcement learning approaches jointly learn a model of the system with the value function or the policy and often allow

for training with less interaction with the the real environment. Reduced learning on the real robot is highly desirable as simulations are frequently faster than real-time while safer for both the robot and its environment. The idea of combining learning in simulation and in the real environment was popularized by the Dyna-architecture (Sutton, 1990), prioritized sweeping (Moore and Atkeson, 1993), and incremental multi-step Q-learning (Peng and Williams, 1996) in reinforcement learning. In robot reinforcement learning, the learning step on the simulated system is often called "mental rehearsal". We first discuss the core issues and techniques in mental rehearsal for robotics (Section 6.1), and, subsequently, we discuss learning methods that have be used in conjunction with learning with forward models (Section 6.2). An overview of publications using simulations to render the learning problem tractable is presented in Table 5.

6.1 Core Issues and General Techniques in Mental Rehearsal

Experience collected in the real world can be used to learn a forward model (Åström and Wittenmark, 1989) from data. Such forward models allow training by interacting with a simulated environment. Only the resulting policy is subsequently transferred to the real environment. Model-based methods can make the learning process substantially more sample efficient. However, depending on the type of model, these methods may require a great deal of memory. In the following paragraphs, we deal with the core issues of mental rehearsal: simulation biases, stochasticity of the real world, and efficient optimization when sampling from a simulator.

Dealing with Simulation Biases. It is impossible to obtain a forward model that is accurate enough to simulate a complex real-world robot system without error. If the learning methods require predicting the future or using derivatives, even small inaccuracies can quickly accumulate, significantly amplifying noise and errors (An et al., 1988). Reinforcement learning approaches exploit such model inaccuracies if they are beneficial for the reward received in simulation (Atkeson and Schaal, 1997). The resulting policies may work well with the forward model (i.e., the simulator) but poorly on the real system. This is known as simulation bias. It is analogous to over-fitting in supervised learning – that is, the algorithm is doing its job well on the model and the training data, respectively, but does not generalize well to the real system or novel data. Simulation bias often leads to biased, potentially physically non-feasible solutions while even iterating between model learning and policy will have slow convergence. Averaging over the model uncertainty in probabilistic models can be used to reduce the bias; see the next paragraph for examples. Another result from these simulation biases is that relatively few researchers have successfully demonstrated that a policy learned in simulation can directly be transferred to a real robot while maintaining a high level of performance. The few examples include maze navigation tasks (Bakker et al., 2003; Oßwald et al., 2010; Youssef, 2005), obstacle avoidance (Fagg et al., 1998) for a mobile robot, very basic robot soccer (Duan et al., 2007) and multi-legged robot locomotion (Ilg et al., 1999; Svinin et al., 2001). Nevertheless, simulation biases can be addressed by introducing stochastic models or distributions over models even if the system is very close to deterministic. Artificially

adding a little noise will smooth model errors and avoid policy over-fitting (Jakobi et al., 1995; Atkeson, 1998). On the downside, potentially very precise policies may be eliminated due to their fragility in the presence of noise. This technique can be beneficial in all of the approaches described in this section. Nevertheless, in recent work, Ross and Bagnell (2012) presented an approach with strong guarantees for learning the model and policy in an iterative fashion even if the true system is not in the model class, indicating that it may be possible to deal with simulation bias.

Distributions over Models for Simulation. Model learning methods that maintain probabilistic uncertainty about true system dynamics allow the RL algorithm to generate distributions over the performance of a policy. Such methods explicitly model the uncertainty associated with the dynamics at each state and action. For example, when using a Gaussian process model of the transition dynamics, a policy can be evaluated by propagating the state and associated uncertainty forward in time. Such evaluations in the model can be used by a policy search approach to identify where to collect more data to improve a policy, and may be exploited to ensure that control is safe and robust to model uncertainty (Schneider, 1996; Bagnell and Schneider, 2001). When the new policy is evaluated on the real system, the novel observations can subsequently be incorporated into the forward model. Bagnell and Schneider (2001) showed that maintaining model uncertainty and using it in the inner-loop of a policy search method enabled effective flight control using only minutes of collected data, while performance was compromised by considering a best-fit model. This approach uses explicit Monte-Carlo simulation in the sample estimates.

By treating model uncertainty as if it were noise (Schneider, 1996) as well as employing analytic approximations of forward simulation, a cart-pole task can be solved with less than 20 seconds of interaction with the physical system (Deisenroth and Rasmussen, 2011); a visually driven block-stacking task has also been learned data-efficiently (Deisenroth et al., 2011). Similarly, solving a linearized control problem with multiple probabilistic models and combining the resulting closed-loop control with open-loop control has resulted in autonomous sideways sliding into a parking spot (Kolter et al., 2010). Instead of learning a model of the system dynamics, Lizotte et al. (2007) directly learned the expected return as a function of the policy parameters using Gaussian process regression in a black-box fashion, and, subsequently, searched for promising parameters in this model. The method has been applied to optimize the gait of an Aibo robot.

Sampling by Re-Using Random Numbers. A forward model can be used as a simulator to create roll-outs for training by sampling. When comparing results of different simulation runs, it is often hard to tell from a small number of samples whether a policy really worked better or whether the results are an effect of the simulated stochasticity. Using a large number of samples to obtain proper estimates of the expectations become prohibitively expensive if a large number of such comparisons need to be performed (e.g., for gradient estimation within an algorithm). A common technique in the statistics and simulation community (Glynn, 1987) to address this problem is to re-use the

Figure 7: Autonomous inverted helicopter flight (Ng et al., 2004b)(Picture reprint with permission of Andrew Ng).

series of random numbers in fixed models, hence mitigating the noise contribution. Ng et al. (2004b,a) extended this approach for learned simulators. The resulting approach, PEGASUS, found various applications in the learning of maneuvers for autonomous helicopters (Bagnell and Schneider, 2001; Bagnell, 2004; Ng et al., 2004b,a), as illustrated in Figure 7. It has been used to learn control parameters for a RC car (Michels et al., 2005) and an autonomous blimp (Ko et al., 2007).

While mental rehearsal has a long history in robotics, it is currently becoming again a hot topic, especially due to the work on probabilistic virtual simulation.

6.2 Successful Learning Approaches with Forward Models

Model-based approaches rely on an approach that finds good policies using the learned model. In this section, we discuss methods that directly obtain a new policy candidate directly from a forward model. Some of these methods have a long history in optimal control and only work in conjunction with a forward model.

Iterative Learning Control. A powerful idea that has been developed in multiple forms in both the reinforcement learning and control communities is the use of crude, approximate models to determine gradients, e.g., for an update step. The resulting new policy is then evaluated in the real world and the model is updated. This approach is known as iterative learning control (Arimoto et al., 1984). A similar preceding idea was employed to minimize trajectory tracking errors (An et al., 1988) and is loosely related to feedback error learning (Kawato, 1990). More recently, variations on the iterative learning control has been employed to learn robot control (Norrlöf, 2002; Bukkems et al., 2005), steering a RC car with a general analysis of approximate models for policy search in (Abbeel et al., 2006), a pick and place task (Freeman et al., 2010), and an impressive application of tying knots with a surgical robot at superhuman speeds (van den Berg et al., 2010).

CORE ISSUES AND GENERAL TECHNIQUES IN MENTAL REHEARSAL

Approach	Employed by...
Dealing with Simulation Biases	An et al. (1988); Atkeson and Schaal (1997); Atkeson (1998); Bakker et al. (2003); Duan et al. (2007); Fagg et al. (1998); Ilg et al. (1999); Jakobi et al. (1995); Oßwald et al. (2010); Ross and Bagnell (2012); Svinin et al. (2001); Youssef (2005)
Distributions over Models for Simulation	Bagnell and Schneider (2001); Deisenroth and Rasmussen (2011); Deisenroth et al. (2011); Kolter et al. (2010); Lizotte et al. (2007)
Sampling by Re-Using Random Numbers	Bagnell and Schneider (2001); Bagnell (2004); Ko et al. (2007); Michels et al. (2005); Ng et al. (2004b,a)

SUCCESSFUL LEARNING APPROACHES WITH FORWARD MODELS

Approach	Employed by...
Iterative Learning Control	Abbeel et al. (2006); An et al. (1988); van den Berg et al. (2010); Bukkems et al. (2005); Freeman et al. (2010); Norrlöf (2002)
Locally Linear Quadratic Regulators	Atkeson and Schaal (1997); Atkeson (1998); Coates et al. (2009); Kolter et al. (2008); Schaal and Atkeson (1994); Tedrake et al. (2010)
Value Function Methods with Learned Models	Bakker et al. (2006); Nemec et al. (2010); Uchibe et al. (1998)
Policy Search with Learned Models	Bagnell and Schneider (2001); Bagnell (2004); Deisenroth et al. (2011); Deisenroth and Rasmussen (2011); Kober and Peters (2010); Ng et al. (2004b,a); Peters et al. (2010a)

Table 5: This table illustrates different methods of making robot reinforcement learning tractable using models.

Locally Linear Quadratic Regulators. Instead of sampling from a forward model-based simulator, such learned models can be directly used to compute optimal control policies. This approach has resulted in a variety of robot reinforcement learning applications that include pendulum swing-up tasks learned with DDP (Atkeson and Schaal, 1997; Atkeson, 1998), devil-sticking (a form of gyroscopic juggling) obtained with local LQR solutions (Schaal and Atkeson, 1994), trajectory following with space-indexed controllers trained with DDP for an autonomous RC car (Kolter et al., 2008), and the aerobatic helicopter flight trained with DDP discussed above (Coates et al., 2009).

Value Function Methods with Learned Models. Obviously, mental rehearsal can be used straightforwardly with value function methods by simply pretending that the simulated roll-outs were generated from the real system. Learning in simulation while the computer is idle and employing directed exploration allows Q-learning to learn a navigation task from scratch in 20 minutes (Bakker et al., 2006). Two robots taking turns in learning a simplified soccer task were also able to profit from mental rehearsal (Uchibe et al., 1998). Nemec et al. (2010) used a value function learned in simulation to initialize the real robot learning. However, it is clear that model-based methods that use the model for creating more direct experience should potentially perform better.

Policy Search with Learned Models. Similarly as for value function methods, all model-free policy search methods can be used in conjunction with learned simulators. For example, both pairwise comparisons of policies and policy gradient approaches have been used with learned simulators (Bagnell and Schneider, 2001; Bagnell, 2004; Ng et al., 2004b,a). Transferring EM-like policy search Kober and Peters (2010) and Relative Entropy Policy Search Peters et al. (2010a) appears to be a natural next step. Nevertheless, as mentioned in Section 6.1, a series of policy update methods has been suggested that were tailored for probabilistic simulation Deisenroth et al. (2011); Deisenroth and Rasmussen (2011); Lizotte et al. (2007).

There still appears to be the need for new methods that make better use of the model knowledge, both in policy search and for value function methods.

7 A Case Study: Ball-in-a-Cup

Up to this point in this paper, we have reviewed a large variety of problems and associated solutions within robot reinforcement learning. In this section, we will take a complementary approach and discuss one task in detail that has previously been studied.

This *ball-in-a-cup* task due to its relative simplicity can serve as an example to highlight some of the challenges and methods that were discussed earlier. We do not claim that the method presented is the best or only way to address the presented problem; instead, our goal is to provide a case study that shows design decisions which can lead to successful robotic reinforcement learning.

In Section 7.1, the experimental setting is described with a focus on the task and the reward. Section 7.2 discusses a type of pre-structured policies that has been particu-

(a) Schematic drawings of the ball-in-a-cup motion

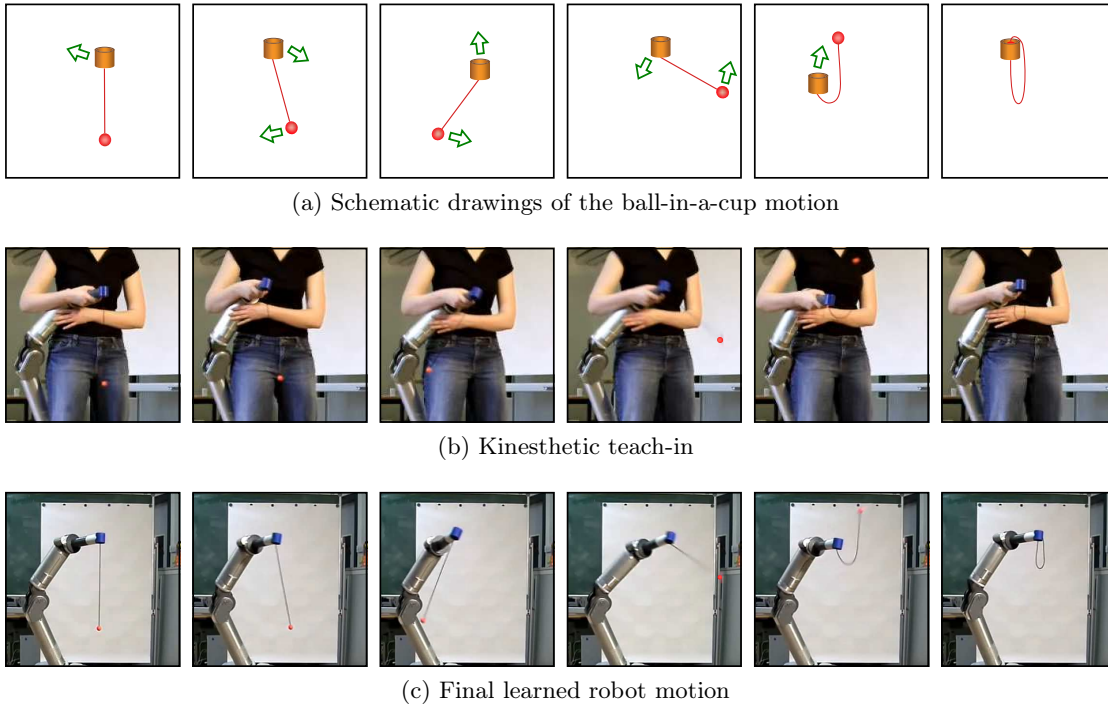(b) Kinesthetic teach-in

(c) Final learned robot motion

Figure 8: This figure shows schematic drawings of the ball-in-a-cup motion (a), the final learned robot motion (c), as well as a kinesthetic teach-in (b). The green arrows show the directions of the current movements in that frame. The human cup motion was taught to the robot by imitation learning. The robot manages to reproduce the imitated motion quite accurately, but the ball misses the cup by several centimeters. After approximately 75 iterations of the Policy learning by Weighting Exploration with the Returns (PoWER) algorithm the robot has improved its motion so that the ball regularly goes into the cup.

larly useful in robotics. Inclusion of prior knowledge is presented in Section 7.3. The advantages of the employed policy search algorithm are explained in Section 7.4. The use of simulations in this task is discussed in Section 7.5 and results on the real robot are described in Section 7.6. Finally, an alternative reinforcement learning approach is briefly explored in Section 7.7.

7.1 Experimental Setting: Task and Reward

The children's game ball-in-a-cup, also known as *balero* and *bilboquet*, is challenging even for adults. The toy consists of a small cup held in one hand (in this case, it is attached to the end-effector of the robot) and a small ball hanging on a string attached to the cup's bottom (for the employed toy, the string is 40cm long). Initially, the ball is at rest, hanging down vertically. The player needs to move quickly to induce motion in the ball through the string, toss the ball in the air, and catch it with the cup. A possible

movement is illustrated in Figure 8a.

The state of the system can be described by joint angles and joint velocities of the robot as well as the Cartesian coordinates and velocities of the ball (neglecting states that cannot be observed straightforwardly like the state of the string or global room air movement). The actions are the joint space accelerations, which are translated into torques by a fixed inverse dynamics controller. Thus, the reinforcement learning approach has to deal with twenty state and seven action dimensions, making discretization infeasible.

An obvious reward function would be a binary return for the whole episode, depending on whether the ball was caught in the cup or not. In order to give the reinforcement learning algorithm a notion of closeness, Kober and Peters (2010) initially used a reward function based solely on the minimal distance between the ball and the cup. However, the algorithm has exploited rewards resulting from hitting the cup with the ball from below or from the side, as such behaviors are easier to achieve and yield comparatively high rewards. To avoid such local optima, it was essential to find a good reward function that contains the additional prior knowledge that getting the ball into the cup is only possible from one direction. Kober and Peters (2010) expressed this constraint by computing the reward as $r(t_c) = \exp\left(-\alpha(x_c - x_b)^2 - \alpha(y_c - y_b)^2\right)$ while $r(t) = 0$ for all $t \neq t_c$. Here, t_c corresponds to the time step when the ball passes the rim of the cup with a downward direction, the cup position is denoted by $[x_c, y_c, z_c] \in \mathbb{R}^3$, the ball position is $[x_b, y_b, z_b] \in \mathbb{R}^3$ and the scaling parameter $\alpha = 100$. This reward function does not capture the case where the ball does not pass above the rim of the cup, as the reward will always be zero. The employed approach performs a local policy search, and hence, an initial policy that brings the ball above the rim of the cup was required.

The task exhibits some surprising complexity as the reward is not only affected by the cup's movements but foremost by the ball's movements. As the ball's movements are very sensitive to small perturbations, the initial conditions, or small arm movement changes will drastically affect the outcome. Creating an accurate simulation is hard due to the nonlinear, unobservable dynamics of the string and its non-negligible weight.

7.2 Appropriate Policy Representation

The policy is represented by dynamical system motor primitives (Ijspeert et al., 2003; Schaal et al., 2007). The global movement is encoded as a point attractor linear dynamical system with an additional local transformation that allows a very parsimonious representation of the policy. This framework ensures the stability of the movement and allows the representation of arbitrarily shaped movements through the primitive's policy parameters. These parameters can be estimated straightforwardly by locally weighted regression. Note that the dynamical systems motor primitives ensure the stability of the movement generation but cannot guarantee the stability of the movement execution. These primitives can be modified through their meta-parameters in order to adapt to the final goal position, the movement amplitude, or the duration of the movement. The resulting movement can start from arbitrary positions and velocities and go to arbitrary final positions while maintaining the overall shape of the trajectory.

While the original formulation in (Ijspeert et al., 2003) for discrete dynamical systems motor primitives used a second-order system to represent the phase z of the movement, this formulation has proven to be unnecessarily complicated in practice. Since then, it has been simplified and, in (Schaal et al., 2007), it was shown that a single first order system suffices

$$\dot{z} = -\tau \alpha_z z. \tag{7}$$

This canonical system has the time constant $\tau = 1/T$ where T is the duration of the motor primitive, a parameter α_z which is chosen such that $z \approx 0$ at T to ensure that the influence of the transformation function, shown in Equation (9), vanishes. Subsequently, the internal state $\boldsymbol{x}$ of a second system is chosen such that positions $\boldsymbol{q}$ of all degrees of freedom are given by $\boldsymbol{q} = \boldsymbol{x}_1$, the velocities $\dot{\boldsymbol{q}}$ by $\dot{\boldsymbol{q}} = \tau \boldsymbol{x}_2 = \dot{\boldsymbol{x}}_1$ and the accelerations $\ddot{\boldsymbol{q}}$ by $\ddot{\boldsymbol{q}} = \tau \dot{\boldsymbol{x}}_2$. Under these assumptions, the learned dynamics of Ijspeert motor primitives can be expressed in the following form

$$\dot{\boldsymbol{x}}_2 = \tau \alpha_x \left(\beta_x \left(\boldsymbol{g} - \boldsymbol{x}_1 \right) - \boldsymbol{x}_2 \right) + \tau \boldsymbol{A} \boldsymbol{f} \left(z \right), \tag{8}$$
$$\dot{\boldsymbol{x}}_1 = \tau \boldsymbol{x}_2.$$

This set of differential equations has the same time constant τ as the canonical system, parameters α_x, β_x set such that the system is critically damped, a goal parameter $\boldsymbol{g}$, a transformation function $\boldsymbol{f}$ and an amplitude matrix $\boldsymbol{A} = \mathrm{diag}(a_1, a_2, \ldots, a_n)$, with the amplitude modifier $\boldsymbol{a} = [a_1, a_2, \ldots, a_n]$. In (Schaal et al., 2007), they use $\boldsymbol{a} = \boldsymbol{g} - \boldsymbol{x}_1^0$ with the initial position $\boldsymbol{x}_1^0$, which ensures linear scaling. Alternative choices are possibly better suited for specific tasks, see e.g., (Park et al., 2008). The transformation function $\boldsymbol{f}(z)$ alters the output of the first system, in Equation (7), so that the second system, in Equation (8), can represent complex nonlinear patterns and it is given by

$$\boldsymbol{f}(z) = \sum_{i=1}^{N} \psi_i(z) \boldsymbol{w}_i z. \tag{9}$$

Here, $\boldsymbol{w}_i$ contains the ith adjustable parameter of all degrees of freedom, N is the number of parameters per degree of freedom, and $\psi_i(z)$ are the corresponding weighting functions (Schaal et al., 2007). Normalized Gaussian kernels are used as weighting functions given by

$$\psi_i(z) = \frac{\exp\left(-h_i \left(z - c_i\right)^2\right)}{\sum_{j=1}^{N} \exp\left(-h_j \left(z - c_j\right)^2\right)}. \tag{10}$$

These weighting functions localize the interaction in phase space using the centers c_i and widths h_i. Note that the degrees of freedom (DoF) are usually all modeled as independent in Equation (8). All DoFs are synchronous as the dynamical systems for all DoFs start at the same time, have the same duration, and the shape of the movement is generated using the transformation $\boldsymbol{f}(z)$ in Equation (9). This transformation function is learned as a function of the shared canonical system in Equation (7).

This policy can be seen as a parameterization of a mean policy in the form $\bar{\boldsymbol{a}} = \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\mu}(\boldsymbol{s}, t)$, which is linear in parameters. Thus, it is straightforward to include prior knowledge from a demonstration using supervised learning by locally weighted regression.

This policy is augmented by an additive exploration $\epsilon(s, t)$ noise term to make policy search methods possible. As a result, the explorative policy can be given in the form $a = \theta^{\mathrm{T}} \mu(s, t) + \epsilon(\mu(s, t))$. Some policy search approaches have previously used state-independent, white Gaussian exploration, i.e., $\epsilon(\mu(s, t)) \sim \mathcal{N}(0, \Sigma)$. However, such unstructured exploration at every step has several disadvantages, notably: (i) it causes a large variance which grows with the number of time-steps, (ii) it perturbs actions too frequently, thus, "washing out" their effects and (iii) it can damage the system that is executing the trajectory.

Alternatively, one could generate a form of structured, state-dependent exploration (Rückstieß et al., 2008) $\epsilon(\mu(s, t)) = \varepsilon_t^{\mathrm{T}} \mu(s, t)$ with $[\varepsilon_t]_{ij} \sim \mathcal{N}(0, \sigma_{ij}^2)$, where σ_{ij}^2 are meta-parameters of the exploration that can also be optimized. This argument results in the policy $a = (\theta + \varepsilon_t)^{\mathrm{T}} \mu(s, t)$ corresponding to the distribution $a \sim \pi(a_t | s_t, t) = \mathcal{N}(a | \theta^{\mathrm{T}} \mu(s, t), \mu(s, t)^{\mathrm{T}} \hat{\Sigma}(s, t))$. Instead of directly exploring in action space, this type of policy explores in parameter space.

7.3 Generating a Teacher Demonstration

Children usually learn this task by first observing another person presenting a demonstration. They then try to duplicate the task through trial-and-error-based learning. To mimic this process, the motor primitives were first initialized by imitation. Subsequently, they were improved them by reinforcement learning.

A demonstration for imitation was obtained by recording the motions of a human player performing kinesthetic teach-in as shown in Figure 8b. Kinesthetic teach-in means "taking the robot by the hand", performing the task by moving the robot while it is in gravity-compensation mode, and recording the joint angles, velocities and accelerations. It requires a back-drivable robot system that is similar enough to a human arm to not cause embodiment issues. Even with demonstration, the resulting robot policy fails to catch the ball with the cup, leading to the need for self-improvement by reinforcement learning. As discussed in Section 7.1, the initial demonstration was needed to ensure that the ball goes above the rim of the cup.

7.4 Reinforcement Learning by Policy Search

Policy search methods are better suited for a scenario like this, where the task is episodic, local optimization is sufficient (thanks to the initial demonstration), and high dimensional, continuous states and actions need to be taken into account. A single update step in a gradient based method usually requires as many episodes as parameters to be optimized. Since the expected number of parameters was in the hundreds, a different approach had to be taken because gradient based methods are impractical in this scenario. Furthermore, the step-size parameter for gradient based methods often is a crucial parameter that needs to be tuned to achieve good performance. Instead, an expectation-maximization inspired algorithm was employed that requires significantly less samples and has no learning rate.

Kober and Peters (2009) have derived a framework of reward weighted imitation. Based on (Dayan and Hinton, 1997) they consider the return of an episode as an improper probability distribution. A lower bound of the logarithm of the expected return is maximized. Depending on the strategy of optimizing this lower bound and the exploration strategy, the framework yields several well known policy search algorithms as well as the novel Policy learning by Weighting Exploration with the Returns (PoWER) algorithm. PoWER is an expectation-maximization inspired algorithm that employs state-dependent exploration (as discussed in Section 7.2). The update rule is given by

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + E\left\{\sum_{t=1}^{T} \boldsymbol{W}\left(\boldsymbol{s}_t, t\right) Q^\pi\left(\boldsymbol{s}_t, \boldsymbol{a}_t, t\right)\right\}^{-1} E\left\{\sum_{t=1}^{T} \boldsymbol{W}\left(\boldsymbol{s}_t, t\right) \boldsymbol{\varepsilon}_t Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, t)\right\},$$

where $\boldsymbol{W}\left(\boldsymbol{s}_t, t\right) = \boldsymbol{\mu}(\boldsymbol{s}, t)\boldsymbol{\mu}(\boldsymbol{s}, t)^{\mathrm{T}}\left(\boldsymbol{\mu}(\boldsymbol{s}, t)^{\mathrm{T}}\hat{\boldsymbol{\Sigma}}\boldsymbol{\mu}(\boldsymbol{s}, t)\right)^{-1}$. Intuitively, this update can be seen as a reward-weighted imitation, (or recombination) of previously seen episodes. Depending on its effect on the state-action value function, the exploration of each episode is incorporated more or less strongly into the updated policy. To reduce the number of trials in this on-policy scenario, the trials are reused through importance sampling (Sutton and Barto, 1998). To avoid the fragility that sometimes results from importance sampling in reinforcement learning, samples with very small importance weights were discarded. In essence, this algorithm performs a local search around the policy learned from demonstration and prior knowledge.

7.5 Use of Simulations in Robot Reinforcement Learning

The robot is simulated by rigid body dynamics with parameters estimated from data. The toy is simulated as a pendulum with an elastic string that switches to a ballistic point mass when the ball is closer to the cup than the string is long. The spring, damper and restitution constants were tuned to match data recorded on a VICON system. The SL framework (Schaal, 2009) allowed us to switch between the simulated robot and the real one with a simple recompile. Even though the simulation matches recorded data very well, policies that get the ball in the cup in simulation usually missed the cup by several centimeters on the real system and vice-versa. One conceivable approach could be to first improve a demonstrated policy in simulation and only perform the fine-tuning on the real robot.

However, this simulation was very helpful to develop and tune the algorithm as it runs faster in simulation than real-time and does not require human supervision or intervention. The algorithm was initially confirmed and tweaked with unrelated, simulated benchmark tasks (shown in (Kober and Peters, 2010)). The use of an importance sampler was essential to achieve good performance and required many tests over the course of two weeks. A very crude importance sampler that considers only the n best previous episodes worked sufficiently well in practice. Depending on the number n the algorithm exhibits are more or less pronounced greedy behavior. Additionally there are a number of possible simplifications for the learning algorithm, some of which work very well in

practice even if the underlying assumption do not strictly hold in reality. The finally employed variant

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \frac{\left\langle \sum_{t=1}^{T} \boldsymbol{\varepsilon}_t Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, t) \right\rangle_{\omega(\boldsymbol{\tau})}}{\left\langle \sum_{t=1}^{T} Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, t) \right\rangle_{\omega(\boldsymbol{\tau})}}$$

assumes that only a single basis function is active at a given time, while there is actually some overlap for the motor primitive basis functions. The importance sampler is denoted by $\langle \cdot \rangle_{\omega(\boldsymbol{\tau})}$. The implementation is further simplified as the reward is zero for all but one time-step per episode.

To adapt the algorithm to this particular task, the most important parameters to tune were the "greediness" of the importance sampling, the initial magnitude of the exploration, and the number of parameters for the policy. These parameters were identified by a coarse grid search in simulation with various initial demonstrations and seeds for the random number generator. Once the simulation and the grid search were coded, this process only took a few minutes. The exploration parameter is fairly robust if it is in the same order of magnitude as the policy parameters. For the importance sampler, using the 10 best previous episodes was a good compromise. The number of policy parameters needs to be high enough to capture enough details to get the ball above the rim of the cup for the initial demonstration. On the other hand, having more policy parameters will potentially slow down the learning process. The number of needed policy parameters for various demonstrations were in the order of 30 parameters per DoF. The demonstration employed for the results shown in more detail in this paper employed 31 parameters per DoF for an approximately 3 second long movement, hence 217 policy parameters total. Having three times as many policy parameters slowed down the convergence only slightly.

7.6 Results on the Real Robot

The first run on the real robot used the demonstration shown in Figure 8 and directly worked without any further parameter tuning. For the five runs with this demonstration, which took approximately one hour each, the robot got the ball into the cup for the first time after 42-45 episodes and regularly succeeded at bringing the ball into the cup after 70-80 episodes. The policy always converged to the maximum after 100 episodes. Running the real robot experiment was tedious as the ball was tracked by a stereo vision system, which sometimes failed and required a manual correction of the reward. As the string frequently entangles during failures and the robot cannot unravel it, human intervention is required. Hence, the ball had to be manually reset after each episode.

If the constraint of getting the ball above the rim of the cup for the initial policy is fulfilled, the presented approach works well for a wide variety of initial demonstrations including various teachers and two different movement strategies (swinging the ball or pulling the ball straight up). Convergence took between 60 and 130 episodes, which largely depends on the initial distance to the cup but also on the robustness of the demonstrated policy.

7.7 Alternative Approach with Value Function Methods

Nemec et al. (2010) employ an alternate reinforcement learning approach to achieve the ball-in-a-cup task with a Mitsubishi PA10 robot. They decomposed the task into two sub-tasks, the swing-up phase and the catching phase. In the swing-up phase, the ball is moved above the cup. In the catching phase, the ball is caught with the cup using an analytic prediction of the ball trajectory based on the movement of a flying point mass. The catching behavior is fixed; only the swing-up behavior is learned. The paper proposes to use SARSA to learn the swing-up movement. The states consist of the cup positions and velocities as well as the angular positions and velocities of the ball. The actions are the accelerations of the cup in a single Cartesian direction. Tractability is achieved by discretizing both the states (324 values) and the actions (5 values) and initialization by simulation. The behavior was first learned in simulation requiring 220 to 300 episodes. The state-action value function learned in simulation was used to initialize the learning on the real robot. The robot required an additional 40 to 90 episodes to adapt the behavior learned in simulation to the real environment.

8 Discussion

We have surveyed the state of the art in robot reinforcement learning for both general reinforcement learning audiences and robotics researchers to provide possibly valuable insight into successful techniques and approaches.

From this overview, it is clear that using reinforcement learning in the domain of robotics is not yet a straightforward undertaking but rather requires a certain amount of skill. Hence, in this section, we highlight several open questions faced by the robotic reinforcement learning community in order to make progress towards "off-the-shelf" approaches as well as a few current practical challenges. Finally, we try to summarize several key lessons from robotic reinforcement learning for the general reinforcement learning community.

8.1 Open Questions in Robotic Reinforcement Learning

Reinforcement learning is clearly not applicable to robotics "out of the box" yet, in contrast to supervised learning where considerable progress has been made in large-scale, easy deployment over the last decade. As this paper illustrates, reinforcement learning can be employed for a wide variety of physical systems and control tasks in robotics. It is unlikely that single answers do exist for such a heterogeneous field, but even for very closely related tasks, appropriate methods currently need to be carefully selected. The user has to decide when sufficient prior knowledge is given and learning can take over. All methods require hand-tuning for choosing appropriate representations, reward functions, and the required prior knowledge. Even the correct use of models in robot reinforcement learning requires substantial future research. Clearly, a key step in robotic reinforcement learning is the automated choice of these elements and having robust algorithms that

limit the required tuning to the point where even a naive user would be capable of using robotic RL.

How to choose representations automatically? The automated selection of appropriate representations remains a difficult problem as the action space in robotics often is inherently continuous and multi-dimensional. While there are good reasons for using the methods presented in Section 4 in their respective domains, the question whether to approximate states, value functions, policies – or a mix of the three – remains open. The correct methods to handle the inevitability of function approximation remain under intense study, as does the theory to back it up.

How to generate reward functions from data? Good reward functions are *always* essential to the success of a robot reinforcement learning approach (see Section 3.4). While inverse reinforcement learning can be used as an alternative to manually designing the reward function, it relies on the design of features that capture the important aspects of the problem space instead. Finding feature candidates may require insights not altogether different from the ones needed to design the actual reward function.

How much can prior knowledge help? How much is needed? Incorporating prior knowledge is one of the main tools to make robotic reinforcement learning tractable (see Section 5). However, it is often hard to tell in advance how much prior knowledge is required to enable a reinforcement learning algorithm to succeed in a reasonable number of episodes. For such cases, a loop of imitation and reinforcement learning may be a desirable alternative. Nevertheless, sometimes, prior knowledge may not help at all. For example, obtaining initial policies from human demonstrations can be virtually impossible if the morphology of the robot is too different from a human's (which is known as the correspondence problem (Argall et al., 2009)). Whether alternative forms of prior knowledge can help here may be a key question to answer.

How to integrate more tightly with perception? Much current work on robotic reinforcement learning relies on subsystems that abstract away perceptual information, limiting the techniques to simple perception systems and heavily pre-processed data. This abstraction is in part due to limitations of existing reinforcement learning approaches at handling inevitably incomplete, ambiguous and noisy sensor data. Learning active perception jointly with the robot's movement and semantic perception are open problems that present tremendous opportunities for simplifying as well as improving robot behavior programming.

How to reduce parameter sensitivity? Many algorithms work fantastically well for a very narrow range of conditions or even tuning parameters. A simple example would be the step size in a gradient based method. However, searching anew for the best parameters for each situation is often prohibitive with physical systems. Instead algorithms

that work fairly well for a large range of situations and parameters are potentially much more interesting for practical robotic applications.

How to deal with model errors and under-modeling? Model based approaches can significantly reduce the need for real-world interactions. Methods that are based on approximate models and use local optimization often work well (see Section 6). As real-world samples are usually more expensive than comparatively cheap calculations, this may be a significant advantage. However, for most robot systems, there will always be under-modeling and resulting model errors. Hence, the policies learned only in simulation frequently cannot be transferred directly to the robot.

This problem may be inevitable due to both uncertainty about true system dynamics, the non-stationarity of system dynamics[6] and the inability of any model in our class to be perfectly accurate in its description of those dynamics, which have led to robust control theory (Zhou and Doyle, 1997). Reinforcement learning approaches mostly require the behavior designer to deal with this problem by incorporating model uncertainty with artificial noise or carefully choosing reward functions to discourage controllers that generate frequencies that might excite unmodeled dynamics.

Tremendous theoretical and algorithmic challenges arise from developing algorithms that are robust to both model uncertainty and under-modeling while ensuring fast learning and performance. A possible approach may be the full Bayesian treatment of the impact of possible model errors onto the policy but has the risk of generating overly conservative policies, as also happened in robust optimal control.

This list of questions is by no means exhaustive, however, it provides a fair impression of the critical issues for basic research in this area.

8.2 Practical Challenges for Robotic Reinforcement Learning

More applied problems for future research result from practical challenges in robot reinforcement learning:

Exploit Data Sets Better. Humans who learn a new task build upon previously learned skills. For example, after a human learns how to throw balls, learning how to throw darts becomes significantly easier. Being able to transfer previously-learned skills to other tasks and potentially to robots of a different type is crucial. For complex tasks, learning cannot be achieved globally. It is essential to reuse other locally learned information from past data sets. While such transfer learning has been studied more extensively in other parts of machine learning, tremendous opportunities for leveraging such data exist within robot reinforcement learning. Making such data sets with many skills publicly available would be a great service for robotic reinforcement learning research.

---

[6]Both the environment and the robot itself change dynamically; for example, vision systems depend on the lighting condition and robot dynamics change with wear and the temperature of the grease.

Comparable Experiments and Consistent Evaluation. The difficulty of performing, and reproducing, large scale experiments due to the expense, fragility, and differences between hardware remains a key limitation in robotic reinforcement learning. The current movement towards more standardization within robotics may aid these efforts significantly, e.g., by possibly providing a standard robotic reinforcement learning setup to a larger community – both real and simulated. These questions need to be addressed in order for research in self-improving robots to progress.

8.3 Robotics Lessons for Reinforcement Learning

Most of the article has aimed equally at both reinforcement learning and robotics researchers as well as practitioners. However, this section attempts to convey a few important, possibly provocative, take-home messages for the classical reinforcement learning community.

Focus on High-Dimensional Continuous Actions and Constant Adaptation. Robotic problems clearly have driven theoretical reinforcement learning research, particularly in policy search, inverse optimal control approaches, and model robustness. The practical impact of robotic reinforcement learning problems (e.g., multi-dimensional continuous action spaces, continuously drifting noise, frequent changes in the hardware and the environment, and the inevitability of undermodeling), may not yet have been sufficiently appreciated by the theoretical reinforcement learning community. These problems have often caused robotic reinforcement learning to take significantly different approaches than would be dictated by theory. Perhaps as a result, robotic reinforcement learning approaches are often closer to classical optimal control solutions than the methods typically studied in the machine learning literature.

Exploit Domain Structure for Scalability. The grounding of RL in robotics alleviates the general problem of scaling reinforcement learning into high dimensional domains by exploiting the structure of the physical world. Prior knowledge of the behavior of the system and the task is often available. Incorporating even crude models and domain structure into the learning approach (e.g., to approximate gradients) has yielded impressive results(Kolter and Ng, 2009a).

Local Optimality and Controlled State Distributions. Much research in classical reinforcement learning aims at finding value functions that are optimal over the entire state space, which is most likely intractable. In contrast, the success of policy search approaches in robotics relies on their implicit maintenance and controlled change of a state distribution under which the policy can be optimized. Focusing on slowly changing state distributions may also benefit value function methods.

Reward Design. It has been repeatedly demonstrated that reinforcement learning approaches benefit significantly from reward shaping (Ng et al., 1999), and particularly from

using rewards that convey a notion of closeness and are not only based on simple binary success or failure. A learning problem is potentially difficult if the reward is sparse, there are significant delays between an action and the associated significant reward, or if the reward is not smooth (i.e., very small changes to the policy lead to a drastically different outcome). In classical reinforcement learning, discrete rewards are often considered, e.g., a small negative reward per time-step and a large positive reward for reaching the goal. In contrast, robotic reinforcement learning approaches often need more physically motivated reward-shaping based on continuous values and consider multi-objective reward functions like minimizing the motor torques while achieving a task.

We hope that these points will help soliciting more new targeted solutions from the reinforcement learning community for robotics.

Acknowledgments & Funding

References

Abbeel, P., Coates, A., Quigley, M., and Ng, A. Y. (2007). An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems (NIPS)*.

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*.

Abbeel, P., Quigley, M., and Ng, A. Y. (2006). Using inaccurate models in reinforcement learning. In *International Conference on Machine Learning (ICML)*.

An, C. H., Atkeson, C. G., and Hollerbach, J. M. (1988). *Model-based control of a robot manipulator*. MIT Press, Cambridge, MA, USA.

Argall, B. D., Browning, B., and Veloso, M. (2008). Learning robot motion control with demonstration and advice-operators. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57:469–483.

Arimoto, S., Kawamura, S., and Miyazaki, F. (1984). Bettering operation of robots by learning. *Journal of Robotic Systems*, 1(2):123–140.

Asada, M., Noda, S., Tawaratsumida, S., and Hosoda, K. (1996). Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23(2-3):279–303.

Atkeson, C. G. (1994). Using local trajectory optimizers to speed up global optimization in dynamic programming. In *Advances in Neural Information Processing Systems (NIPS)*.

Atkeson, C. G. (1998). Nonparametric model-based reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*.

Atkeson, C. G., Moore, A., and Stefan, S. (1997). Locally weighted learning for control. *AI Review*, 11:75–113.

Atkeson, C. G. and Schaal, S. (1997). Robot learning from demonstration. In *International Conference on Machine Learning (ICML)*.

Bagnell, J. A. (2004). *Learning Decisions: Robustness, Uncertainty, and Approximation.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Bagnell, J. A., Ng, A. Y., Kakade, S., and Schneider, J. (2003). Policy search by dynamic programming. In *Advances in Neural Information Processing Systems (NIPS)*.

Bagnell, J. A. and Schneider, J. (2003). Covariant policy search. In *International Joint Conference on Artifical Intelligence (IJCAI)*.

Bagnell, J. A. and Schneider, J. C. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Baird, L. C. and Klopf, H. (1993). Reinforcement learning with high-dimensional continuous actions. Technical Report WL-TR-93-1147, Wright Laboratory, Wright-Patterson Air Force Base, OH 45433-7301.

Bakker, B., Zhumatiy, V., Gruener, G., and Schmidhuber, J. (2003). A robot that reinforcement-learns to identify and memorize important previous observations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Bakker, B., Zhumatiy, V., Gruener, G., and Schmidhuber, J. (2006). Quasi-online reinforcement learning for robots. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Barto, A. G. and Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ.

Bellman, R. E. (1967). *Introduction to the Mathematical Theory of Control Processes*, volume 40-I. Academic Press, New York, NY.

Bellman, R. E. (1971). *Introduction to the Mathematical Theory of Control Processes*, volume 40-II. Academic Press, New York, NY.

Benbrahim, H., Doleac, J., Franklin, J., and Selfridge, O. (1992). Real-time learning: A ball on a beam. In *International Joint Conference on Neural Networks (IJCNN)*.

Benbrahim, H. and Franklin, J. A. (1997). Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems*, 22(3-4):283–302.

Bentivegna, D. C. (2004). *Learning from Observation Using Primitives*. PhD thesis, Georgia Institute of Technology.

Bentivegna, D. C., Atkeson, C. G., and Cheng, G. (2004). Learning from observation and practice using behavioral primitives: Marble maze.

Bertsekas, D. P. (1995). *Dynamic Programming and Optimal Control*. Athena Scientific.

Betts, J. T. (2001). *Practical methods for optimal control using nonlinear programming*, volume 3 of *Advances in Design and Control*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

Birdwell, N. and Livingston, S. (2007). Reinforcement learning in sensor-guided AIBO robots. Technical report, University of Tennesse, Knoxville. advised by Dr. Itamar Elhanany.

Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer.

Bitzer, S., Howard, M., and Vijayakumar, S. (2010). Using dimensionality reduction to exploit constraints in reinforcement learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Boyan, J. A. and Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems (NIPS)*.

Brafman, R. I. and Tennenholtz, M. (2002). R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231.

Buchli, J., Stulp, F., Theodorou, E., and Schaal, S. (2011). Learning variable impedance control. *International Journal of Robotic Research*, 30(7):820–833.

Bukkems, B., Kostic, D., de Jager, B., and Steinbuch, M. (2005). Learning-based identification and iterative learning control of direct-drive robots. *IEEE Transactions on Control Systems Technology*, 13(4):537–549.

Buşoniu, L., Babuška, R., de Schutter, B., and Ernst, D. (2010). *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Boca Raton, Florida.

Coates, A., Abbeel, P., and Ng, A. Y. (2009). Apprenticeship learning for helicopter control. *Communications of the ACM*, 52(7):97–105.

Cocora, A., Kersting, K., Plagemann, C., Burgard, W., and de Raedt, L. (2006). Learning relational navigation policies. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Conn, K. and Peters II, R. A. (2007). Reinforcement learning with a supervisor for a mobile robot in a real-world environment. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*.

Daniel, C., Neumann, G., and Peters, J. (2012). Learning concurrent motor skills in versatile solution spaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Daumé III, H., Langford, J., and Marcu, D. (2009). Search-based structured prediction. *Machine Learning Journal (MLJ)*, 75:297–325.

Dayan, P. and Hinton, G. E. (1997). Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278.

Deisenroth, M. P. and Rasmussen, C. E. (2011). PILCO: A model-based and data-efficient approach to policy search. In *28th International Conference on Machine Learning (ICML)*.

Deisenroth, M. P., Rasmussen, C. E., and Fox, D. (2011). Learning to control a low-cost manipulator using data-efficient reinforcement learning. In *Robotics: Science and Systems (R:SS)*.

Donnart, J.-Y. and Meyer, J.-A. (1996). Learning reactive and planning rules in a motivationally autonomous animat. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(3):381–395.

Donoho, D. L. (2000). High-dimensional data analysis: the curses and blessings of dimensionality. In *American Mathematical Society Conference Math Challenges of the 21st Century*.

Dorigo, M. and Colombetti, M. (1993). Robot shaping: Developing situated agents through learning. Technical report, International Computer Science Institute, Berkeley, CA.

Duan, Y., Cui, B., and Yang, H. (2008). Robot navigation based on fuzzy RL algorithm. In *International Symposium on Neural Networks (ISNN)*.

Duan, Y., Liu, Q., and Xu, X. (2007). Application of reinforcement learning in robot soccer. *Engineering Applications of Artificial Intelligence*, 20(7):936–950.

Endo, G., Morimoto, J., Matsubara, T., Nakanishi, J., and Cheng, G. (2008). Learning CPG-based biped locomotion with a policy gradient method: Application to a humanoid robot. *International Journal of Robotics Research*, 27(2):213–228.

Erden, M. S. and Leblebicioğlu, K. (2008). Free gait generation with reinforcement learning for a six-legged robot. *Robotics and Autonomous Systems*, 56(3):199–212.

Fagg, A. H., Lotspeich, D. L., Hoff, and J. Bekey, G. A. (1998). Rapid reinforcement learning for reactive control policy design for autonomous robots. In *Artificial Life in Robotics*.

Fidelman, P. and Stone, P. (2004). Learning ball acquisition on a physical robot. In *International Symposium on Robotics and Automation (ISRA)*.

Freeman, C., Lewin, P., Rogers, E., and Ratcliffe, J. (2010). Iterative learning control applied to a gantry robot and conveyor system. *Transactions of the Institute of Measurement and Control*, 32(3):251–264.

Gaskett, C., Fletcher, L., and Zelinsky, A. (2000). Reinforcement learning for a vision based mobile robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Geng, T., Porr, B., and Wörgötter, F. (2006). Fast biped walking with a reflexive controller and real-time policy searching. In *Advances in Neural Information Processing Systems (NIPS)*.

Glynn, P. (1987). Likelihood ratio gradient estimation: An overview. In *Winter Simulation Conference (WSC)*.

Goldberg, D. E. (1989). *Genetic algorithms.* Addision Wesley.

Gordon, G. J. (1999). *Approximate Solutions to Markov Decision Processes.* PhD thesis, School of Computer Science, Carnegie Mellon University.

Gräve, K., Stückler, J., and Behnke, S. (2010). Learning motion skills from expert demonstrations and own experience using Gaussian process regression. In *Joint International Symposium on Robotics (ISR) and German Conference on Robotics (ROBOTIK)*.

Greensmith, E., Bartlett, P. L., and Baxter, J. (2004). Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5:1471–1530.

Guenter, F., Hersch, M., Calinon, S., and Billard, A. (2007). Reinforcement learning for imitating constrained reaching movements. *Advanced Robotics*, 21(13):1521–1544.

Gullapalli, V., Franklin, J., and Benbrahim, H. (1994). Acquiring robot skills via reinforcement learning. *Control Systems Magazine, IEEE*, 14(1):13–24.

Hafner, R. and Riedmiller, M. (2003). Reinforcement learning on a omnidirectional mobile robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Hafner, R. and Riedmiller, M. (2007). Neural reinforcement learning controllers for a real robot application. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Hailu, G. and Sommer, G. (1998). Integrating symbolic knowledge in reinforcement learning. In *IEEE International Conference on Systems, Man and Cybernetics (SMC)*.

Hart, S. and Grupen, R. (2011). Learning generalizable control programs. *IEEE Transactions on Autonomous Mental Development*, 3(3):216–231.

Hester, T., Quinlan, M., and Stone, P. (2010). Generalized model learning for reinforcement learning on a humanoid robot. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Hester, T., Quinlan, M., and Stone, P. (2012). RTMBA: A real-time model-based reinforcement learning architecture for robot control. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Huang, X. and Weng, J. (2002). Novelty and reinforcement learning in the value system of developmental robots. In *2nd International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*.

Huber, M. and Grupen, R. A. (1997). A feedback control structure for on-line learning tasks. *Robotics and Autonomous Systems*, 22(3-4):303–315.

Ijspeert, A. J., Nakanishi, J., and Schaal, S. (2003). Learning attractor landscapes for learning motor primitives. In *Advances in Neural Information Processing Systems (NIPS)*.

Ilg, W., Albiez, J., Jedele, H., Berns, K., and Dillmann, R. (1999). Adaptive periodic movement control for the four legged walking machine BISAM. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Jaakkola, T., Jordan, M. I., and Singh, S. P. (1993). Convergence of stochastic iterative dynamic programming algorithms. In *Advances in Neural Information Processing Systems (NIPS)*.

Jacobson, D. H. and Mayne, D. Q. (1970). *Differential Dynamic Programming*. Elsevier.

Jakobi, N., Husbands, P., and Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. In *3rd European Conference on Artificial Life*.

Kaelbling, L. P. (1990). *Learning in Embedded Systems*. PhD thesis, Stanford University, Stanford, California.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Kakade, S. (2003). *On the Sample Complexity of Reinforcement Learning*. PhD thesis, Gatsby Computational Neuroscience Unit. University College London.

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning (ICML)*.

Kalakrishnan, M., Righetti, L., Pastor, P., and Schaal, S. (2011). Learning force control policies for compliant manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Kalman, R. E. (1964). When is a linear control system optimal? *Journal of Basic Engineering*, 86(1):51–60.

Kalmár, Z., Szepesvári, C., and Lőrincz, A. (1998). Modular reinforcement learning: An application to a real robot task. *Lecture Notes in Artificial Intelligence*, 1545:29–45.

Kappen, H. (2005). Path integrals and symmetry breaking for optimal control theory. *Journal of Statistical Mechanics: Theory and Experiment*, page P11011.

Katz, D., Pyuro, Y., and Brock, O. (2008). Learning to manipulate articulated objects in unstructured environments using a grounded relational representation. In *Robotics: Science and Systems (R:SS)*.

Kawato, M. (1990). Feedback-error-learning neural network for supervised motor learning. *Advanced Neural Computers*, 6(3):365–372.

Kearns, M. and Singh, S. P. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232.

Keeney, R. and Raiffa, H. (1976). *Decisions with multiple objectives: Preferences and value tradeoffs*. J. Wiley, New York.

Kimura, H., Yamashita, T., and Kobayashi, S. (2001). Reinforcement learning of walking behavior for a four-legged robot. In *IEEE Conference on Decision and Control (CDC)*.

Kirchner, F. (1997). Q-learning of complex behaviours on a six-legged walking machine. In *EUROMICRO Workshop on Advanced Mobile Robots*.

Kirk, D. E. (1970). *Optimal control theory*. Prentice-Hall, Englewood Cliffs, New Jersey.

Ko, J., Klein, D. J., Fox, D., and Hähnel, D. (2007). Gaussian processes and reinforcement learning for identification and control of an autonomous blimp. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Kober, J., Mohler, B., and Peters, J. (2008). Learning perceptual coupling for motor primitives. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Kober, J., Oztop, E., and Peters, J. (2010). Reinforcement learning to adjust robot movements to new situations. In *Robotics: Science and Systems (R:SS)*.

Kober, J. and Peters, J. (2009). Policy search for motor primitives in robotics. In *Advances in Neural Information Processing Systems (NIPS)*.

Kober, J. and Peters, J. (2010). Policy search for motor primitives in robotics. *Machine Learning*, 84(1-2):171–203.

Kohl, N. and Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Kollar, T. and Roy, N. (2008). Trajectory optimization using reinforcement learning for map exploration. *International Journal of Robotics Research*, 27(2):175–197.

Kolter, J. Z., Abbeel, P., and Ng, A. Y. (2007). Hierarchical apprenticeship learning with application to quadruped locomotion. In *Advances in Neural Information Processing Systems (NIPS)*.

Kolter, J. Z., Coates, A., Ng, A. Y., Gu, Y., and DuHadway, C. (2008). Space-indexed dynamic programming: Learning to follow trajectories. In *International Conference on Machine Learning (ICML)*.

Kolter, J. Z. and Ng, A. Y. (2009a). Policy search via the signed derivative. In *Robotics: Science and Systems (R:SS)*.

Kolter, J. Z. and Ng, A. Y. (2009b). Regularization and feature selection in least-squares temporal difference learning. In *International Conference on Machine Learning (ICML)*.

Kolter, J. Z., Plagemann, C., Jackson, D. T., Ng, A. Y., and Thrun, S. (2010). A probabilistic approach to mixed open-loop and closed-loop control, with application to extreme autonomous driving. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Konidaris, G. D., Kuindersma, S., Grupen, R., and Barto, A. G. (2011a). Autonomous skill acquisition on a mobile manipulator. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Konidaris, G. D., Kuindersma, S., Grupen, R., and Barto, A. G. (2012). Robot learning from demonstration by constructing skill trees. *International Journal of Robotics Research*, 31(3):360–375.

Konidaris, G. D., Osentoski, S., and Thomas, P. (2011b). Value function approximation in reinforcement learning using the Fourier basis. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Kroemer, O., Detry, R., Piater, J., and Peters, J. (2009). Active learning using mean shift optimization for robot grasping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Kroemer, O., Detry, R., Piater, J., and Peters, J. (2010). Combining active learning and reactive control for robot grasping. *Robotics and Autonomous Systems*, 58(9):1105–1116.

Kuhn, H. W. and Tucker, A. W. (1950). Nonlinear programming. In *Berkeley Symposium on Mathematical Statistics and Probability*.

Kuindersma, S., Grupen, R., and Barto, A. G. (2011). Learning dynamic arm motions for postural recovery. In *IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS)*.

Kwok, C. and Fox, D. (2004). Reinforcement learning for sensing strategies. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Langford, J. and Zadrozny, B. (2005). Relating reinforcement learning performance to classification performance. In *22nd International Conference on Machine Learning (ICML)*.

Latzke, T., Behnke, S., and Bennewitz, M. (2007). Imitative reinforcement learning for soccer playing robots. In *RoboCup 2006: Robot Soccer World Cup X*, pages 47–58. Springer-Verlag.

Laud, A. D. (2004). *Theory and Application of Reward Shaping in Reinforcement Learning*. PhD thesis, University of Illinois at Urbana-Champaign.

Lewis, M. E. and Puterman, M. L. (2001). *The Handbook of Markov Decision Processes: Methods and Applications*, chapter Bias Optimality, pages 89–111. Kluwer.

Lin, H.-I. and Lai, C.-C. (2012). Learning collision-free reaching skill from primitives. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Lizotte, D., Wang, T., Bowling, M., and Schuurmans, D. (2007). Automatic gait optimization with Gaussian process regression. In *International Joint Conference on Artifical Intelligence (IJCAI)*.

Mahadevan, S. and Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55(2-3):311 – 365.

Martínez-Marín, T. and Duckett, T. (2005). Fast reinforcement learning for vision-guided mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Matarić, M. J. (1994). Reward functions for accelerated learning. In *International Conference on Machine Learning (ICML)*.

Matarić, M. J. (1997). Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4:73–83.

Michels, J., Saxena, A., and Ng, A. Y. (2005). High speed obstacle avoidance using monocular vision and reinforcement learning. In *International Conference on Machine Learning (ICML)*.

Mitsunaga, N., Smith, C., Kanda, T., Ishiguro, H., and Hagita, N. (2005). Robot behavior adaptation for human-robot interaction based on policy gradient reinforcement learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Miyamoto, H., Schaal, S., Gandolfo, F., Gomi, H., Koike, Y., Osu, R., Nakano, E., Wada, Y., and Kawato, M. (1996). A Kendama learning robot based on bi-directional theory. *Neural Networks*, 9(8):1281–1302.

Moldovan, T. M. and Abbeel, P. (2012). Safe exploration in markov decision processes. In *29th International Conference on Machine Learning (ICML)*, pages 1711–1718.

Moore, A. W. and Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130.

Morimoto, J. and Doya, K. (2001). Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. *Robotics and Autonomous Systems*, 36(1):37–51.

Muelling, K., Kober, J., Kroemer, O., and Peters, J. (2012). Learning to select and generalize striking movements in robot table tennis. *International Journal of Robotics Research*.

Nakanishi, J., Cory, R., Mistry, M., Peters, J., and Schaal, S. (2008). Operational space control: A theoretical and emprical comparison. *International Journal of Robotics Research*, 27:737–757.

Nemec, B., Tamošiūnaitė, M., Wörgötter, F., and Ude, A. (2009). Task adaptation through exploration and action sequencing. In *IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS)*.

Nemec, B., Zorko, M., and Zlajpah, L. (2010). Learning of a ball-in-a-cup playing robot. In *International Workshop on Robotics in Alpe-Adria-Danube Region (RAAD)*.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., and Liang, E. (2004a). Autonomous inverted helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics (ISER)*.

Ng, A. Y., Harada, D., and Russell, S. J. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning (ICML)*.

Ng, A. Y., Kim, H. J., Jordan, M. I., and Sastry, S. (2004b). Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*.

Norrlöf, M. (2002). An adaptive iterative learning control algorithm with experiments on an industrial robot. *IEEE Transactions on Robotics*, 18(2):245–251.

Oßwald, S., Hornung, A., and Bennewitz, M. (2010). Learning reliable and efficient navigation with a humanoid. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Paletta, L., Fritz, G., Kintzler, F., Irran, J., and Dorffner, G. (2007). Perception and developmental learning of affordances in autonomous robots. In Hertzberg, J., Beetz, M., and Englert, R., editors, *KI 2007: Advances in Artificial Intelligence*, volume 4667 of *Lecture Notes in Computer Science*, pages 235–250. Springer.

Park, D.-H., Hoffmann, H., Pastor, P., and Schaal, S. (2008). Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In *IEEE International Conference on Humanoid Robots (HUMANOIDS)*.

Pastor, P., Kalakrishnan, M., Chitta, S., Theodorou, E., and Schaal, S. (2011). Skill learning and task outcome prediction for manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Pendrith, M. (1999). Reinforcement learning in situated agents: Some theoretical problems and practical solutions. In *European Workshop on Learning Robots (EWRL)*.

Peng, J. and Williams, R. J. (1996). Incremental multi-step q-learning. *Machine Learning*, 22(1):283–290.

Perkins, T. J. and Barto, A. G. (2002). Lyapunov design for safe reinforcement learning. *Journal of Machine Learning Research*, 3:803–832.

Peters, J., Muelling, K., and Altun, Y. (2010a). Relative entropy policy search. In *National Conference on Artificial Intelligence (AAAI)*.

Peters, J., Muelling, K., Kober, J., Nguyen-Tuong, D., and Kroemer, O. (2010b). Towards motor skill learning for robotics. In *International Symposium on Robotics Research (ISRR)*.

Peters, J. and Schaal, S. (2008a). Learning to control in operational space. *International Journal of Robotics Research*, 27(2):197–212.

Peters, J. and Schaal, S. (2008b). Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190.

Peters, J. and Schaal, S. (2008c). Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697.

Peters, J., Vijayakumar, S., and Schaal, S. (2004). Linear quadratic regulation as benchmark for policy gradient methods. Technical report, University of Southern California.

Piater, J. H., Jodogne, S., Detry, R., Kraft, D., Krüger, N., Kroemer, O., and Peters, J. (2011). Learning visual representations for perception-action systems. *International Journal of Robotics Research*, 30(3):294–307.

Platt, R., Grupen, R. A., and Fagg, A. H. (2006). Improving grasp skills using schema structured learning. In *International Conference on Development and Learning*.

Powell, W. B. (2012). AI, OR and control theory: A rosetta stone for stochastic optimization. Technical report, Princeton University.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience.

Randløv, J. and Alstrøm, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. In *International Conference on Machine Learning (ICML)*, pages 463–471.

Rasmussen, C. and Williams, C. (2006). *Gaussian Processes for Machine Learning*. Adaptive Computation And Machine Learning. MIT Press.

Åström, K. J. and Wittenmark, B. (1989). *Adaptive control*. Addison-Wesley.

Ratliff, N., Bagnell, J. A., and Srinivasa, S. (2007). Imitation learning for locomotion and manipulation. In *IEEE-RAS International Conference on Humanoid Robots (HU-MANOIDS)*.

Ratliff, N., Bradley, D., Bagnell, J. A., and Chestnutt, J. (2006a). Boosting structured prediction for imitation learning. In *Advances in Neural Information Processing Systems (NIPS)*.

Ratliff, N. D., Bagnell, J. A., and Zinkevich, M. A. (2006b). Maximum margin planning. In *International Conference on Machine Learning (ICML)*.

Riedmiller, M., Gabel, T., Hafner, R., and Lange, S. (2009). Reinforcement learning for robot soccer. *Autonomous Robots*, 27(1):55–73.

Rivlin, T. J. (1969). *An Introduction to the Approximation of Functions*. Courier Dover Publications.

Roberts, J. W., Manchester, I., and Tedrake, R. (2011). Feedback controller parameterizations for reinforcement learning. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*.

Roberts, J. W., Moret, L., Zhang, J., and Tedrake, R. (2010). *From Motor to Interaction Learning in Robots*, volume 264 of *Studies in Computational Intelligence*, chapter Motor learning at intermediate Reynolds number: experiments with policy gradient on the flapping flight of a rigid wing, pages 293–309. Springer.

Rosenstein, M. T. and Barto, A. G. (2004). Reinforcement learning with supervision by a stable controller. In *American Control Conference*.

Ross, S. and Bagnell, J. A. (2012). Agnostic system identification for model-based reinforcement learning. In *International Conference on Machine Learning (ICML)*.

Ross, S., Gordon, G., and Bagnell, J. A. (2011a). A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artifical Intelligence and Statistics (AISTATS)*.

Ross, S., Munoz, D., Hebert, M., and AndrewBagnell, J. (2011b). Learning message-passing inference machines for structured prediction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Rottmann, A., Plagemann, C., Hilgers, P., and Burgard, W. (2007). Autonomous blimp control using model-free reinforcement learning in a continuous state and action space. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Rubinstein, R. Y. and Kroese, D. P. (2004). *The Cross Entropy Method: A Unified Approach To Combinatorial Optimization, Monte-carlo Simulation (Information Science and Statistics)*. Springer-Verlag New York, Inc.

Rückstieß, T., Felder, M., and Schmidhuber, J. (2008). State-dependent exploration for policy gradient methods. In *European Conference on Machine Learning (ECML)*.

Russell, S. (1998). Learning agents for uncertain environments (extended abstract). In *Conference on Computational Learning Theory (COLT)*.

Rust, J. (1997). Using randomization to break the curse of dimensionality. *Econometrica*, 65(3):487–516.

Sato, M., Nakamura, Y., and Ishii, S. (2002). Reinforcement learning for biped locomotion. In *International Conference on Artificial Neural Networks*.

Schaal, S. (1996). Learning from demonstration. In *Advances in Neural Information Processing Systems (NIPS)*.

Schaal, S. (1999). Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6):233–242.

Schaal, S. (2009). The SL simulation and real-time control software package. Technical report, University of Southern California.

Schaal, S. and Atkeson, C. G. (1994). Robot juggling: An implementation of memory-based learning. *Control Systems Magazine*, 14(1):57–71.

Schaal, S., Atkeson, C. G., and Vijayakumar, S. (2002). Scalable techniques from non-parameteric statistics for real-time robot learning. *Applied Intelligence*, 17(1):49–60.

Schaal, S., Mohajerian, P., and Ijspeert, A. J. (2007). Dynamics systems vs. optimal control – A unifying view. *Progress in Brain Research*, 165(1):425–445.

Schneider, J. G. (1996). Exploiting model uncertainty estimates for safe dynamic control learning. In *Advances in Neural Information Processing Systems (NIPS)*.

Schwartz, A. (1993). A reinforcement learning method for maximizing undiscounted rewards. In *International Conference on Machine Learning (ICML)*.

Silver, D., Bagnell, J. A., and Stentz, A. (2008). High performance outdoor navigation from overhead data using imitation learning. In *Robotics: Science and Systems (R:SS)*.

Silver, D., Bagnell, J. A., and Stentz, A. (2010). Learning from demonstration for autonomous navigation in complex unstructured terrain. *International Journal of Robotics Research*, 29(12):1565–1592.

Smart, W. D. and Kaelbling, L. P. (1998). A framework for reinforcement learning on real robots. In *National Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence (AAAI/IAAI)*.

Smart, W. D. and Kaelbling, L. P. (2002). Effective reinforcement learning for mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Soni, V. and Singh, S. P. (2006). Reinforcement learning of hierarchical skills on the Sony AIBO robot. In *International Conference on Development and Learning (ICDL)*.

Sorg, J., Singh, S. P., and Lewis, R. L. (2010). Reward design via online gradient ascent. In *Advances in Neural Information Processing Systems (NIPS)*.

Spall, J. C. (2003). *Introduction to Stochastic Search and Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1 edition.

Strens, M. and Moore, A. (2001). Direct policy search using paired statistical tests. In *International Conference on Machine Learning (ICML)*.

Stulp, F. and Sigaud, O. (2012). Path integral policy improvement with covariance matrix adaptation. In *International Conference on Machine Learning (ICML)*.

Stulp, F., Theodorou, E., Kalakrishnan, M., Pastor, P., Righetti, L., and Schaal, S. (2011). Learning motion primitive goals for robust manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *International Machine Learning Conference (ICML)*.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning*. MIT Press, Boston, MA.

Sutton, R. S., Barto, A. G., and Williams, R. J. (1991). Reinforcement learning is direct adaptive optimal control. In *American Control Conference*.

Sutton, R. S., Koop, A., and Silver, D. (2007). On the role of tracking in stationary environments. In *International Conference on Machine Learning (ICML)*.

Sutton, R. S., McAllester, D., Singh, S. P., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*.

Svinin, M. M., Yamada, K., and Ueda, K. (2001). Emergent synthesis of motion patterns for locomotion robots. *Artificial Intelligence in Engineering*, 15(4):353–363.

Tadepalli, P. and Ok, D. (1994). H-learning: A reinforcement learning method to optimize undiscounted average reward. Technical Report 94-30-1, Department of Computer Science, Oregon State University.

Tamei, T. and Shibata, T. (2009). Policy gradient learning of cooperative interaction with a robot using user's biological signals. In *International Conference on Neural Information Processing (ICONIP)*.

Tamošiūnaitė, M., Nemec, B., Ude, A., and Wörgötter, F. (2011). Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives. *Robotics and Autonomous Systems*, 59:910 – 922.

Tedrake, R. (2004). Stochastic policy gradient reinforcement learning on a simple 3D biped. In *International Conference on Intelligent Robots and Systems (IROS)*.

Tedrake, R., Manchester, I. R., Tobenkin, M. M., and Roberts, J. W. (2010). LQR-trees: Feedback motion planning via sums of squares verification. *International Journal of Robotics Research*, 29:1038–1052.

Tedrake, R., Zhang, T. W., and Seung, H. S. (2005). Learning to walk in 20 minutes. In *Yale Workshop on Adaptive and Learning Systems*.

Tesch, M., Schneider, J. G., and Choset, H. (2011). Using response surfaces and expected improvement to optimize snake robot gait parameters. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Theodorou, E. A., Buchli, J., and Schaal, S. (2010). Reinforcement learning of motor skills in high dimensions: A path integral approach. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Thrun, S. (1995). An approach to learning mobile robot navigation. *Robotics and Autonomous Systems*, 15:301–319.

Tokic, M., Ertel, W., and Fessler, J. (2009). The crawler, a class room demonstrator for reinforcement learning. In *International Florida Artificial Intelligence Research Society Conference (FLAIRS)*.

Toussaint, M., Storkey, A., and Harmeling, S. (2010). *Inference and Learning in Dynamic Models*, chapter Expectation-Maximization methods for solving (PO)MDPs and optimal control problems. Cambridge University Press.

Touzet, C. (1997). Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems, Special Issue on Learning Robot: The New Wave*, 22(3-4):251–281.

Tsitsiklis, J. N. and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690.

Uchibe, E., Asada, M., and Hosoda, K. (1998). Cooperative behavior acquisition in multi mobile robots environment by reinforcement learning based on state vector estimation. In *IEEE International Conference on Robotics and Automation (ICRA)*.

van den Berg, J., Miller, S., Duckworth, D., Hu, H., Wan, A., Fu, X.-Y., Goldberg, K., and Abbeel, P. (2010). Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations. In *International Conference on Robotics and Automation (ICRA)*.

Vlassis, N., Toussaint, M., Kontes, G., and Piperidis, S. (2009). Learning model-free robot control by a Monte Carlo EM algorithm. *Autonomous Robots*, 27(2):123–130.

Wang, B., Li, J., and Liu, H. (2006). A heuristic reinforcement learning for robot approaching objects. In *IEEE Conference on Robotics, Automation and Mechatronics*.

Whitman, E. C. and Atkeson, C. G. (2010). Control of instantaneously coupled systems applied to humanoid walking. In *IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS)*.

Wikipedia (2013). Fosbury flop. http://en.wikipedia.org/wiki/Fosbury Flop.

Willgoss, R. A. and Iqbal, J. (1999). Reinforcement learning of behaviors in mobile robots using noisy infrared sensing. In *Australian Conference on Robotics and Automation*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.

Yamaguchi, J. and Takanishi, A. (1997). Development of a biped walking robot having antagonistic driven joints using nonlinear spring mechanism. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Yasuda, T. and Ohkura, K. (2008). A reinforcement learning technique with an adaptive action generator for a multi-robot system. In *International Conference on Simulation of Adaptive Behavior (SAB)*.

Youssef, S. M. (2005). Neuro-based learning of mobile robots with evolutionary path planning. In *ICGST International Conference on Automation, Robotics and Autonomous Systems (ARAS)*.

Zhou, K. and Doyle, J. C. (1997). *Essentials of Robust Control*. Prentice Hall.

Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Zucker, M. and Bagnell, J. A. (2012). Reinforcement planning: RL for optimal planners. In *IEEE International Conference on Robotics and Automation (ICRA)*.