Survey of Model-Based Reinforcement Learning: Applications on Robotics

Athanasios S. Polydoros and Lazaros Nalpantidis

Department of Mechanical and Manufacturing Engineering
AC Meyers Vaenge 15, 2450 Copenhagen SV, Denmark
athapoly,lanalpa@m-tech.aau.dk

Abstract. Reinforcement learning is an appealing approach for allowing robots to learn new tasks. Relevant literature reveals a plethora of methods, but at the same time makes clear the lack of implementations for dealing with real life challenges. Current expectations raise the demand for adaptable robots. We argue that, by employing model-based reinforcement learning, the—now limited—adaptability characteristics of robotic systems can be expanded. Also, model-based reinforcement learning exhibits advantages that makes it more applicable to real life use-cases compared to model-free methods. Thus, in this survey, model-based methods that have been applied in robotics are covered. We categorize them based on the derivation of an optimal policy, the definition of the returns function, the type of the transition model and the learned task. Finally, we discuss the applicability of model-based reinforcement learning approaches in new applications, taking into consideration the state of the art in both algorithms and hardware.

Keywords: intelligent robotics, machine learning, model-based reinforcement learning, robot learning, policy search, transition models, reward functions

1 Introduction

Reinforcement Learning (RL) constitutes a significant aspect of the Artificial Intelligence field with numerous applications ranging from finance to robotics and a plethora of proposed approaches. Robotics is a very challenging application for RL since it involves interactions between a mechanical system and its environment. Such interactions can harm both the mechanical system and humans, especially in service and industrial robots which are expected to operate close to humans. On the other hand, RL can increase the adaptability of robotics systems, which is an important feature in order to deal with a complex and dynamic environment. Naturally, the use of RL for

robot control is gaining popularity over the last few years [1, 2] in a broad spectrum of robotics applications [3]. This fact comes as no surprise considering that in the years to come robots are expected to become more intelligent than just being capable of repeating an explicit set of simple actions. The motivation behind this work is based on the observation that numerous survey papers on RL for robotics have been published [1–3], but all of them are mainly focused on model-free approaches. However, during the recent years there is an increasing interest in applications of model-based RL in robotics [4–11]. Another motivating factor for this survey has been the observation that robots themselves are changing; low-cost collaborative manipulators have established themselves as a big part of the robotics market and are expected to gain even more popularity in the years to come. The lower cost of such robots inevitably dictates compromises in their accuracy and repeatability, as well as in the quality of their internal sensing apparatuses. Thus, one needs to reconsider the characteristics of learning techniques to be applied on this new breed of robots.

The goals of this work are two-fold:

1. to give an up-to-date overview of RL for robotics focusing on model-based methods and showcasing their relative advantages;
2. to investigate the appropriateness of RL methods for handling the challenges of low-cost robotic manipulators and to conclude on a suggestion, in Section 7, of a robust and reliable model-based RL approach for performing tasks with low-cost manipulators.

To achieve the first goal, we examine the literature for applications of model-based RL both on simulated and real robotic systems. We present value-function and policy search methods alongside with strategies for setting the returns function and learning the transition model which have been successfully applied on robotic systems. We address our second goal only after the analysis of the state-of-the-art, and in conjunction with the outcomes that derive from our first goal.

This survey is organized as follows. After this introductory section, in Section 2 we present the value function methods used for model-based RL. In Section 3 we discuss returns functions, while in Section 4 the policy search methods are covered. Furthermore, in

Section 5 we present the approaches for modeling the transition dynamics. Section 6 reviews cases of model-based RL methods applied on robotic platforms for learning various tasks. Finally, we summarize the outcomes of this survey in Section 7 and discuss the applicability of model-based RL in applications beyond the ones already reported in the literature.

1.1 Background on model-based RL

The main difference between RL and other types of machine learning, is that the learning procedure involves interactions between the agent and its environment, thus the agent learns the desired task by gathering experience directly from its environment and does not need an external teacher. However, the discrimination between those two components (agent and environment) is not always straightforward and depends on the application. For example, in applications such as bipedal walking and UAV control, the environment is assumed to include the motors of the robot.

The state of the robot s can be described either in a continuous or a discrete manner. In each state the robot controller applies an action a (motor command) that results in a change of its state. Those actions are derived from a policy function $\pi(\cdot)$ which maps – in the deterministic case – the state to a single action $\pi(s) \mapsto a$. In the stochastic case, the policy function depends on a random variable ε and the mapping is written as a probability distribution over actions $\pi(a|s, \varepsilon)$.

The goal of a reinforcement learning algorithm is to find the policy that maximizes the expected return which is defined by a reward function r. The main types of functions that indicate the agent's return can be either discounted or averaged and are calculated on a a finite or infinite horizon. The finite horizon is used when the learned task has a known end state, otherwise infinite functions are more appropriate. The interaction between the robot and the environment is modeled as a Markov Decision Process (MDP), a tuple $[S, A, \varepsilon, P\left(s'|s, a\right), R(s, s', a), \gamma]$. Where S is the set of possible robot states, A is the set of actions, $P\left(s'|s, a\right)$ is the probability of transition at a future state s' when the robot is at state s and applies action a. $R(s, s', a)$ is the reward that the robot expects when it

transits to state s' and depends on the applied action a, state s and is calculated through the reward function. Finally, γ the discount factor of the reward function.

RL approaches are distinguished in two main classes, the model-free (also known as direct) and the model-based (also known as indirect) methods. The main difference between model-based and model free RL is whether a model of the interactions between the robot and the environment is employed. In model-free methods there is not a model and thus the rewards and the optimal actions derive by trial-and-error approach with the physical system. In model-based methods there exists a model of the transition dynamics which is used for the derivation of the rewards and optimal actions. Thus the policies are optimized at the model and the optimal policy is applied at the physical system. Figure 1 illustrates a model-based RL pipeline.

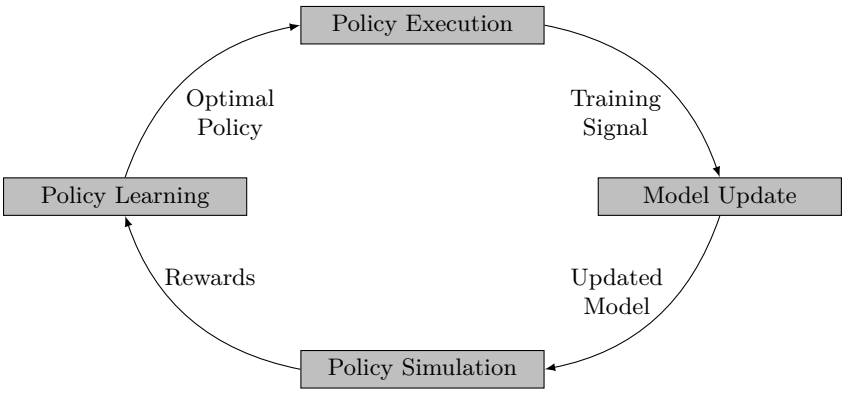


Fig. 1: Pipeline of a model-based RL algorithm.

Model-free methods attracted the most scientific interest, but sampling trajectories for the derivation of the optimal policy can be a disadvantage when applied on real robots. An alternative is the use of a model-based approach. In this context, the optimal policy is derived based on internal simulations of a learned forward model that corresponds to a representation of the robot's dynamics. This characteristic significantly reduces the physical interactions between the robot and its environment, which results in significantly less

mechanical wear. On the other hand, its main disadvantage is that model-based RL algorithms heavily depend on the model's ability to accurately represent the transition dynamics. The pros and cons of the two classes of RL algorithms are summarized in Table 1.

RL Methods	**Advantages**	**Disadvantages**
Model-based RL	– Small number of interactions between robot & environment – Faster convergence to optimal solution.	–Depend on transition models – Model accuracy has a big impact on learning tasks
Model-free RL	– No need for prior knowledge of transitions – Easily implementable	– Slow learning convergence – High wear & tear of the robot – High risk of damage

Table 1: Advantages and disadvantages of model-free and model-based RL methods

The solution of a RL problem can be derived from two alternative method families. The most widely used one is the *value function* approaches, which estimate the future expected return of an agent that is in a given state and performs actions according to a policy.The value functions are distinguished in state-value function $V^{\pi}(s) = E_{\pi}\{R_t|s_t = s\}$, which is the expected return of being in state s following policy π, and action-value function $Q^{\pi}(s,a) = E_{\pi}\{R_t|s_t = s, a_t = a\}$, which is equivalently defined as the future expected return of applying action a in state s and following policy π from then on. The returns function, in the cases of finite learning horizon H, is a sum of rewards $R_t = \sum_{t=0}^{T} r(s_t, a_t)$. In the case of infinite horizons, the returns function can either be in discounted (12) or averaged form (13).

Value function approaches estimate an optimal value function—either state-value or action-value—in order to derive the optimal actions in each state. Thus, the policy that maximizes the long-term reward is derived from the optimal actions in each state. The value function methods are sorted in four classes: *i*) Dynamic Programming (DP) methods that require a model of the transition dynamics, *ii*) Monte Carlo (MC) methods that are based on sampling, *iii*) Temporal Difference Learning (TDL) methods that take into account the

difference of the value function between two state transitions, and *iv*) Differential Dynamic Programming methods (DDP).

An alternative to the use of value function methods is using *policy search* methods. Instead of reconstructing a policy from the optimal value function, in policy search methods the optimal policy is learned directly. This fact allows state-of-the-art policy search methods to converge faster in the case of high-DOF robotic systems, compared to value-function methods [2, Sec. 2.3]. Policies are represented by a wide variety of approaches ranging from simple linear functions to sophisticated Dynamic Movement Primitives [1]. Their common characteristic is that they are all parametrized by a set of parameters, which has to be optimized so as to maximize the cumulative reward. Policy search involves approaches such as: *i*) gradient-based methods that update the parametrized set using hill-climbing approaches on the gradient of the reward function, *ii*) Expectation-Maximization (EM) methods that infer the parameters by maximizing the log-likelihood probability of the rewards, *iii*) Information Theory (Inf.Th.) methods that exploit concepts such as entropy for the derivation of optimized policies, *iv*) Bayesian optimization methods, and *v*) Evolutionary computation. Figure 2 summarizes the main approaches for solving a RL problem. This taxonomy will be followed in the remainder of this survey to explore the state-of-the-art of the field.

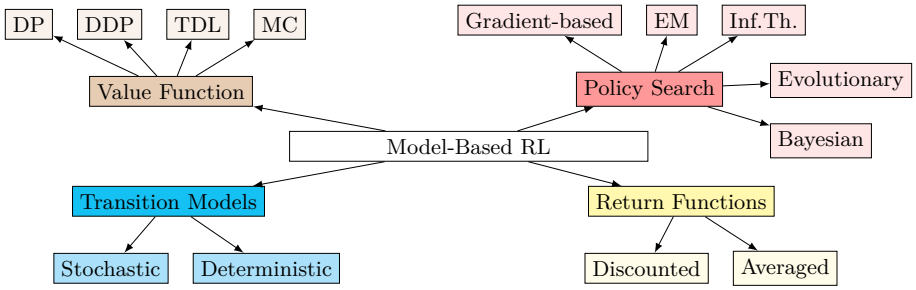


Fig. 2: Categorization of approaches for solving a model-based Reinforcement Learning problem.

2 Value function methods

The state-value function of a RL problem can be written in a recursive form as shown in equation (1), thus the value function of a state depends on the immediate reward of the possible future states and their discounted value function weighted by the transition probabilities.

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P(s'|s,a) \left[R(s,s',a) + \gamma V^{\pi}(s')\right] \tag{1}$$

Equivalently, the action-value function can be written in a recursive form as:

$$Q^{\pi}(s,a) = \sum_{s'} P(s'|s,a) \left[R(s,s',a) + \gamma Q^{\pi}(s',a')\right] \tag{2}$$

Equations (1) and (2) are the Bellman equations for state-value and action-value respectively. The goal of value function methods is to compute the value function and to derive the best policy by maximizing the value functions in each state. It can be shown that there exists an optimal policy π^* for which the value function is maximized, the optimal state-value function V^* is written as and given by:

$$V^{\pi_*}(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s,a) \left[R(s,s',a) + \gamma V^{\pi_*}(s')\right] \tag{3}$$

This corresponds to Bellman's optimality equation for the state-value function. Similarly, the optimal action-value function is a Bellman's optimality equation and is written as:

$$Q^{\pi_*}(s) = \sum_{s'} P(s'|s,a) \left[R(s,s',a) + \gamma \max_{a'} Q^{\pi_*}(s',a')\right] \tag{4}$$

The value function methods provide the means for the derivation of the optimal value function that is used for the reconstruction of the optimal policy. This class of methods is quite popular in the context of model-based RL [9, 10, 12–26] [27–30] and especially dynamic programming methods which require a known model of the transition dynamics. Other methods, such as Monte-Carlo and Temporal

Difference, are mainly used in model-free RL since they don't require a known model. Figure 3 summarizes and categorizes works in the relevant literature, where value function methods have been applied on robotic systems for model-based RL.

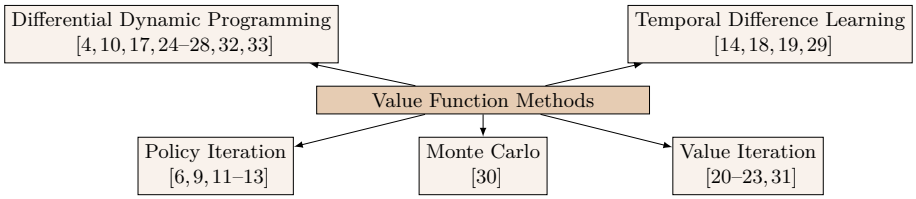


Fig. 3: Overview of value function methods literature.

Dynamic Programming (DP) methods are iterative algorithms and can be distinguished as policy iteration [34] and value iteration approaches [35]. The policy iteration consists of two steps; the first step is the policy evaluation, where the state or action value function is calculated for a given policy. This is derived from an iterative procedure that calculates the value function for each state until a convergence criterion is met. Policy evaluation is followed by the policy improvement step, where the best action for each state is derived.

Policy Iteration In [11, 12] the authors employ a policy iteration approach using the Natural Actor-Critic algorithm [36]. In the critic part, the policy evaluation is performed using a temporal difference approach, namely the LSTD-Q(λ) [37], while the actor part performs the improvement of policy using the natural gradient approach [38]. An actor-critic algorithm is also used in [6] where the authors improve the policy derivation using the gradient of the value function in the critic part of the algorithm. In [13] a modification of the Least-Square Policy Iteration algorithm [39, 40] is used for performing the evaluation and improvement steps. In the evaluation step, a parametric approximation of the action-value function is used in order to reduce the state space. The policy is improved by solving a system of linear equations for the derivation of the parameters that characterize the approximated action-value function. The prediction and time

efficiency of the algorithm are increased by employing the prioritized sweeping (PS) technique, introduced in [41], which focuses the updates of the state values on "interesting" states. In [9] the authors use a Gaussian Process [42] model for approximating the action-value function. The policy evaluation is based on samples generated from the transition model, which are used for the approximation of the action-value. The improvement of the policy can be performed either by a greedy approach using the optimal action-value for each state, or taking into account the entire set of possible action samples for each state.

Value Iteration In policy iteration methods, the policy is updated only after the evaluation step has converged, which can be a time and resource consuming task. In contrast, the value iteration methods do not wait until the convergence of the evaluation procedure for updating the policy. In [20] the value iteration approach is used in conjunction with PS. The algorithm is optimized for online learning due to its parallel execution. The authors of [21] use the R-Max algorithm, introduced in [43]. It uses value iteration for planning on the relocatable action model, which is a decomposed MDP [44].R-MAX is a simple but powerful model-based RL algorithm since it can achieve near-optimal performance with polynomial computational complexity [43]. The algorithm applies the approach of "optimism under uncertainty", assuming that all the unknown states return the maximum possible reward and unknown transitions drive to a fictitious new state. The algorithm consist of two repeated steps, the compute and act and the observe and update. At the first step the agent is based to its knowledge in order to compute and apply the optimal policy which is executed until the end of the episode or until a new state is reached. This step is followed by the observe and update step where the agent updates, for each taken action, the rewards and the transition probabilities of the model and recomputes an optimal policy. Thus, R-MAX forces the agent to explore its domain which leads to an accurate model and a near-optimal policy. An extension of the R-MAX algorithm, the Relational explorer (REX), that can handle relational state-spaces is presented in [45], the authors in [46] expand REX so that it can also handle demonstrations

from experts and also avoid dangerous policy executions and create safer models [47].

In [22], the authors propose a value iteration algorithm, the RL-DT, which reduces the interactions with the environment by performing a targeted exploration. Finally, in [23] the authors also employ value iteration in order to derive the optimal policy.

Differential Dynamic Programing Differential Dynamic Programing (DDP) is a widely used approach in model-based RL for the derivation of an optimal policy. DDP algorithms have their roots in the field of optimal control. They perform local trajectory optimization and thus they require an initial trajectory in order to employ a two-step optimization procedure. DDP algorithms are initialized by specifying a local quadratic model of the value function and a linear model of the policy function that corresponds to the initial trajectory. In the first step of the optimization procedure the current policy is applied on the dynamics model, which generates a simulated trajectory. In the second step, the generated trajectory is used for the calculation of the components of the modeled value function at each point; then the parameters of the policy function are updated accordingly.

DDP is employed in [32], where the local quadratic model of the state-value function is parametrized by the first and second derivative of the value function and the model of the local policy depends on the policy's gradient. In [17] the authors introduce the Minimax DDP where a local model of the action-value function is used. Additionally to traditional DDP, in Minimax DDP the existence of disturbances is represented by a disturbance term in the local model of state value. This approach makes the algorithm capable of dealing with more inaccurate dynamics models compared to traditional DDP. A stochastic extension of DDP, is presented in [33] where the states are described by probability distributions. Its advancement over conventional DDP methods is that the optimal actions derive from a first-order gradient descent algorithm and that their convergence is faster [48]. In [49] the authors perform trajectory optimization in real time by optimizing a constrained utility function using sequential quadratic programming. Their approach allows for real-

time planning of trajectories and does not require the computation or storage of policies.

Linear Quadratic Regulator (LQR) is a special case of DDP where the transition dynamics are modeled by a linear approximation, as in equation (5), parametrized by A and B. Where A is a $m \times m$ matrix and B is a $m \times k$ matrix and m, k are the dimensions of the state s and control a vectors respectively. In LQR, the optimal policy is a linear combination of the state and matrix P_i (6) which derives from (11). The reward function, in the LQR case, represents the cost for being at state s and applying action a at time step t and it is represented in a quadratic form, as in (6), with parameters Q and W which are positive symmetric matrices with dimensions $m \times m$ and $k \times k$ respectively.

$$s' = As + Ba \tag{5}$$

$$r\left(s, a\right) = s^T Q s + a^T W a \tag{6}$$

By applying equations (5) and (6), the value function that has to be optimized can be written as:

$$V^{\pi}\left(s\right) = \underset{a \in A(s)}{min} \left[s^T Q s + a^T W a + V^{\pi}\left(As + Ba\right) \right] \tag{7}$$

which can be optimized by following the value iteration procedure. This yields the optimal policy at each i-step (equation (8)) and the optimal state-value function (equation (9)).

$$\pi\left(s\right) = K_i s \tag{8}$$

$$V^*(s) = s^T P_i s \tag{9}$$

where matrices K and P (10) and (11) are derived using DDP as follows:

$$K_i = -\left(W + B^T P_{i-1} B\right)^{-1} B^T P_{i-1} A \tag{10}$$

$$P_i = Q + K_i^T W K_i + (A + BK_i)^T P_{i-1} (A + BK_i) \tag{11}$$

A more detailed presentation of DDP and LQR can be found in [50, 51] and [52, Ch. 5].

In [28] the authors use a LQR controller for learning the required trajectory with a small amount of data. Furthermore, LQR methods can speed-up the trajectory execution when used in conjunction with iterative learning control approaches [10, 24]. The authors of [27] use demonstrated trajectories from experts in order to further improve the learned optimal policy. The authors in [4] employ a variant of LQR—the iterative LQR—whose objective is to minimize the KL divergence between an optimal trajectory and the trajectory created by the current policy. The minimization is subject to constrains ensuring that the algorithm converges to optimized trajectories and policies that match. A disadvantage of LQR is the assumption of linear deterministic state transitions; this can be overcome by extensions, such as Gauss-Newton LQR, that have been proposed for handling such cases [25].

Other approaches Other alternatives to DP are Temporal Difference (TD) and Monte-Carlo (MC) algorithms. Although they are considered to be model-free methods, since they do not require a model of the transition dynamics, they are used in model-based RL in cases where the transition model is not *a priori* known and instead is learned online. MC algorithms are sample-based and use the generated samples for the estimation of the expected reward. TD learning algorithms estimate the value function based on other estimates in an incremental way (bootstrapping) and thus they don't require the completion of the RL task.

In [30] the authors employ a Monte Carlo Tree search algorithm for the estimation of the action-value function within a multithread architecture for achieving real time learning. A TD learning algorithm—the SARSA introduced in [53]— is used in [29]. There, the model is learned on-line from the state transitions and then is used for the estimation of the action-value function. In [14] the authors are using the Dyna-Q algorithm, which combines the Q-learning [54] with the Dyna framework [55] that integrates model learning, value estimation and the agent's reacting. Dyna-Q has also been employed in [18, 19] for on-line model updates in order to achieve time efficient learning of the required task.

3 Return functions

The selection of the appropriate returns function can have a significant impact on the performance of the model-based RL algorithm. The returns function affects both the convergence and the feasibility of the learned task, since the derivation of the optimal policy relies on it. The goal of RL algorithms is to find the set of actions at each time step that optimizes the returns function in the long run. The returns function aggregates rewards or punishments over a whole episode, i.e. from the initial state s_0 until the end of the task.

The returns functions are classified according to the length of the task that they accumulate into finite and infinite horizon functions. The first ones are more applicable to goal-oriented tasks with an *a priori* known target state; thus, they are commonly used in robotics applications. Another discrimination among returns functions R is whether the rewards/punishments r at each step are accumulated equivalently. Discounted returns functions, as shown in equation (12)

$$R = \sum_{t=0}^{\infty} \gamma^t r\left(s_t, a_t\right) \tag{12}$$

put large influence on early rewards and the influence is exponentially decreasing with respect to the time steps of the horizon. On the other hand, in averaged returns functions, as shown in equation (13)

$$R = \frac{1}{T} \sum_{k=0}^{T-1} r\left(s_t, a_t\right) \tag{13}$$

all the rewards received during the learning episode are equivalently accumulated. Figure 4 organizes the relevant literature of returns functions according to this criterion.

The discount factor represents the uncertainty about future rewards that will be obtained by following a policy and since the uncertainty rises with time, the influence of future reward declines. The discount factor γ has to be selected according to the learned task and represents a trade-off between the convergence time and the quality of the learned task. Thus, small discount factors lead to fast convergence of the RL algorithm. However, then the learned solution can have unsatisfactory quality since the rewards received at the end of the horizon have very small influence on the returns function.

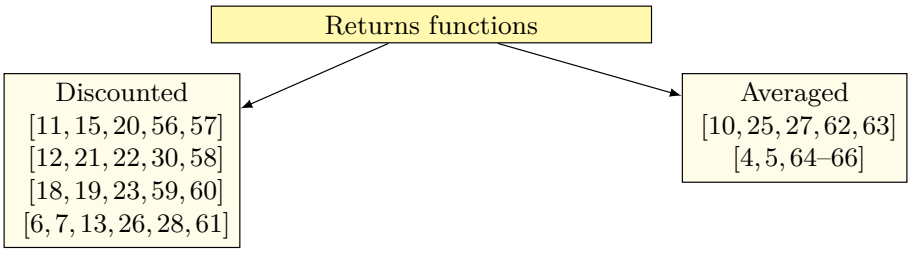


Fig. 4: Overview of returns functions literature.

Discounted returns functions are the most popular on robotics applications of model-based RL. The choice of the reward function strongly depends on the goal of the learned task, the constraints of the robot, and the selection of the policy optimization procedure. In numerous proposed approaches [5,7,10–12,20,58,59,67] the reward is a distance metric between the target and the current state. In some other cases it is a cost function that penalizes the deviation from a demonstrated trajectory [6,16,24,25,27,56]. Furthermore, the reward has to be approximated in quadratic form when applying differential dynamic programing for the derivation of the optimal policy [16, 24]. In applications where the state space is discretized the reward has the form of a step function [13, 20, 22]. The constraints of the robot and the environment can also be introduced in a reward function; e.g. as a cost component that penalizes states that are in collision or close to one [62]. Finally, there exist approaches that add a small penalty on all actions taken by the robot that do not drive it towards the target state [18, 23]. This happens to achieve faster convergence of the learning algorithm.

4 Policy search methods

Policy search methods constitute an alternative way of deriving the optimal policy. While value function methods construct the policy indirectly by finding the actions that maximize the value function at each state, policy search methods directly calculate the optimal policy by modifying its parameters. The algorithms for the derivation of the optimal parameters can be classified in three major groups and in a couple of less popular ones. Gradient-based approaches try to

optimize the parametrized policy function by employing a gradient ascent approach. Their main disadvantage is that the gradient step parameter needs to be set, which can affect the convergence of the algorithm. On the other hand, sampling-based methods do not suffer from this issue—the policy is parametrized with latent variables that are inferred using iterative algorithms. However, their drawbacks are that convergence to the optimal trajectory is not guaranteed and that they can create trajectories far from the training data of the used transition model. On the other hand, Information Theory methods deal with those problems by bounding the trajectory exploration. Last, approaches based on Bayesian optimization and evolutionary computation have also been employed. A taxonomy of policy search methods is illustrated in Figure 5.

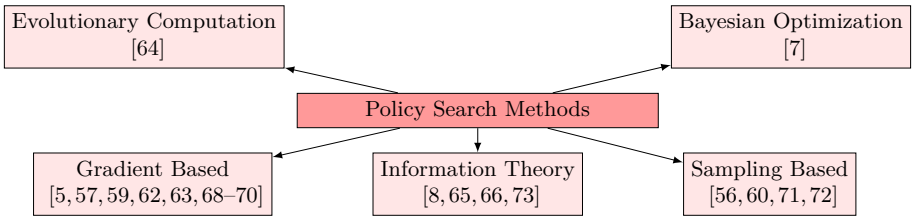


Fig. 5: Overview of policy search approaches literature.

Gradient-based In [57], the direct gradient algorithm [74] is used for policy optimization. The policy is parametrized as an artificial neural network whose weights are modified during policy search. The authors speed-up the learning time by training a simulation-optimal policy and transferring it to the real system which performs further modifications. In [59] the authors use a gradient ascent method for the maximization of the expected reward. The Probabilistic Inference for Learning Control (PILCO) framework, introduced in [67] and applied in [5, 62, 63, 68], is the state of the art approach for solving RL problems since it requires small amount of data for learning the policy and is time efficient. The key characteristic of the algorithm is the ability of the transition dynamics model to handle uncertainty on its inputs [75]. This characteristic makes more accurate

the prediction for long trajectory horizons and therefore the impact of modeling errors is reduced.

PILCO employs a gradient method for the policy improvement. The controller's optimization criterion is the minimization of the long term expected return which penalizes the distance from the target. For the derivation of the gradient the authors exploit the chain rule of partial derivatives.

In more detail, PILCO utilizes Gaussian Process as a method for learning transition dynamics (see further in Section 5) which yields a set of predictive distributions for the state of the robot at each time step of the horizon. Thus, the predictions about the state of the robot are stochastic and described by normal distributions $(p(s_1), p(s_2), \ldots, p(s_k)) \sim \mathcal{N}(s_k|\mu_k, \Sigma_k)$. The next step is to calculate the expected return at each step of the horizon, as in equation (14). Thus, the expected reward of a trajectory derives from the reward function $r(\cdot)$ at each state s_k of the trajectory. The reward is weighted by the probability of being at that state $\mathcal{N}(s_k|\mu_k, \Sigma_k)$ as it derives from the transition model of the algorithm.

$$R_k = \mathbf{E}_s[r(s_k)] = \int r(s_k)\,\mathcal{N}(s_k|\mu_k, \Sigma_k)\,ds_k \tag{14}$$

In this case, the reward function represents the one-step cost of the applied policy and has to be selected in order to make the integral in equation (14) analytically tractable.

Given that the policy is a linear mapping and parametrized by a set of parameters $\theta = [A, b]$, $\pi(s_k|\theta)$ as shown in equation (15), the goal is to apply a gradient-descent approach for the derivation of parameters θ that minimize the expected return R_k.

$$\pi(s_k) = As_k + b \tag{15}$$

Thus, by applying the chain rule, the derivatives in equation (16) remain to be calculated.

$$\frac{dR_k}{d\theta} = \frac{dR_k}{dp(s_k)}\frac{dp(s_k)}{d\theta} \tag{16}$$

Since $p(s_k)$ is normally distributed, it depends on two parameters, the mean μ_k and the covariance matrix Σ_k of the normal distribution.

Thus, equation (16) can be written as:

$$\frac{dR_k}{d\theta} = \frac{\partial R_k}{\partial \mu_k}\frac{d\mu_k}{d\theta} + \frac{\partial R_k}{\partial \Sigma_k}\frac{d\Sigma_k}{d\theta} \tag{17}$$

The derivatives in equation (17) can be computed analytically by iteratively applying the chain rule, which makes the use of gradient-based method applicable. Further details about their derivation can be found in [76].

In [63], PILCO was successfully applied in order to learn a policy on a high dimensional system in conjunction with dimensionality reduction. The authors in [68] employ PILCO for the derivation of a policy within an imitation learning problem. In this context the objective function is based on Kullback–Leibler (KL) divergence, which is a measure of similarity between a trajectory demonstrated by an expert and the trajectory created by the controller. PILCO has also been used in [5] for learning a policy that is able to generalize to unknown tasks. Moreover, the authors in [69] employ a neural network control policy where the network's weights correspond to the learned policy parameters. The goal of the network's stochastic optimization is the derivation of such parameters that best correspond to the optimal trajectory. The optimal trajectory is the one that minimizes a defined cost function using the Newton's method.

Another gradient-based method for policy optimization is presented in [70], where the authors extend the model-free policy gradients with parameter-based exploration (PGPE) method to the model-based scenario (M-PGPE). In this case the policy is deterministic and represented as a linear function. Thus, for dealing with the agent's exploration, the policy's parameters are sampled from a probability distribution which depends on a set of hyper-parameters. The goal of the algorithm is to calculate the hyper-parameters that maximize the expected return by employing a gradient ascent method.

Sampling-based The sampling based approaches that have been applied for policy search on model-based RL employ the PEGASUS (Policy Evaluation-of-Goodness And Search Using Scenarios) algorithm introduced in [77]. PEGASUS deals with the problem of sampling variance by transforming a stochastic MDP or POMDP

into an approximated deterministic one. When a stochastic model is used for the transition dynamics, the model's prediction is a probability distribution over the next state $p(s_{t+1})$. Thus, when a fixed policy is evaluated using the stochastic model, the predictions for the next state vary because they are drawn from a probability distribution. As a result, the expected payoff of the policy is affected. This drawback is solved in PEGASUS by augmenting the (PO)MDP with a set of fixed predefined random numbers (stable seeds) which transform the stochastic to a deterministic transition model. The algorithm is iterative and initialized by the approximated deterministic transition model, the reward function, the random numbers, the distribution that describes the belief about the initial state and the policy parameters. The goal is to identify the policy parameters that maximize the reward function.

PEGASUS has been applied in [60] for learning a neural network controller consisting of four tunable parameters. The policy parameters that maximize the reward derive from a hill-climbing algorithm that can be either a gradient ascent or a random-walk. The performance of the learned policy using this approach was found better compared to the one derived from the demonstrations of a human expert. Moreover, a neural network controller was also presented in [71]. Its weights are the policy parameters that are tuned according to the PEGASUS algorithm. In [72] the authors apply PEGASUS for learning a controller of six parameters based on the output of a vision system that is able to identify collision hazard from a monocular camera. In [56] PEGASUS is used as well for learning a policy with ten free parameters in conjunction with the Nelder-Mead method for updating the parameters.

Information Theory Information theory approaches exploit the concept of entropy which is used as the basis for optimizing the parameters of the policy. Entropy, in the field of information theory, is a measure of the uncertainty related to an event. In the context of model-based RL, entropy is used for the evaluation of a trajectory generated from a policy compared to a reference trajectory—usually obtained from an external demonstrator. The authors in [66] introduce the variational guided policy search algorithm for the derivation of a suitable policy for high-dimensional complex tasks. The algo-

rithm consists of two steps, the policy exploration and the policy optimization step. In the first step, differential dynamic programming is used for trajectory exploration which penalizes large deviations from the current policy. In the next step, the parameters of the policy are optimized based on the trajectory derived from the exploration step. The optimization is based on the minimization of KL divergence, an entropy measure, which is performed using stochastic gradient descent. In [65] and [8] the model-free Relative Entropy Policy Search (REPS) algorithm, as introduced in [78], is extended to model-based RL for learning generalized upper-level policies that can be used for learning tasks with slight variations in their context. Upper-level policies alongside with low-level are applied for learning generalizations of tasks. Thus, low-level policies are responsible for a specific context, such as motion at a specified location, while upper-level policies generalize this specific context of lower-level policies. The use of a Gaussian Process transition model decreases the amount of data required for training, by generating artificial data that are used for updating the policy. Even better, it also reduces the interactions between the robot and the environment. The goal of the REPS algorithm is to optimize the generalized policy without deviating significantly from the observed data.

Furthermore, the Trust Region Policy Optimization (TRPO) algorithm is presented in [61] where an objective function is optimized subject to KL divergence constraints. The objective function represents the expected reward of a stochastic policy while the KL divergence puts an upper-bound constraint on the policy's parameters. Both the objective function and the divergence constraints are approximated by Monte Carlo simulation while the optimization problem is solved by the conjugate gradients method.

The Policy Improvement with Path Integral (PI^2) algorithm is introduced in [79] which derives from the stochastic optimal control framework and employs the quantum mechanic's concept of path integrals for performing policy improvements. The PI^2 algorithm has only one tunable parameter, the exploration noise and also can scale well on high-dimensional robot learning problems, such as learning of humanoid robots. Also its comparison with information theoretic methods indicate that the later can be anticipated as an approximation to stochastic optimal control. Furthermore PI^2 can be used both

on model-based [80,81] and model-free RL approaches [82] where the concept of motor primitives [83] is used for policy parametrization. An extended presentation of the PI^2 algorithm can be found in the survey by Deisenroth et al. [1].

Another path integral based approach is presented in [73], the Model Predictive Path Integral Control (MPPI) for achieving stochastic optimization of trajectories. MPPI, contrary to standard path integral methods, does not assume that there is noise only on the states that the robot controls through its actions but also assumes noisy indirectly control states. In addition, the algorithm provides a clearly formulated expression of the optimal controls for the entire learning horizon. Those advancements are achieved by exploiting the information theory concept of relative entropy. Thus, MPPI minimizes the entropy between the optimal and the followed trajectory of the robot by exploiting the concepts of Radon-Nikodym derivative and Girsanov's theorem.

Other approaches Other methods for policy search include Bayesian optimization [7] and evolutionary computing [64]. In [7] Bayesian optimization is employed for decreasing the number of evaluations that are performed on the objective function for the derivation of the optimal parameters. This can be achieved by modeling the objective function with a Gaussian Process model and use this model for evaluating the objective function. On the other hand, the authors in [64] employ an evolutionary computing approach for the derivation of the optimized policy by optimizing the weights of an artificial neural network.

5 Transition Models

A transition model is a mapping $f(s, a) \mapsto s'$ from the state s of the robot at a given time step of the horizon, and the applied commands a to the state s' at the next time step. Transition models have a large impact on the performance of model-based RL algorithms since the policy learning step relies on the accuracy of their predictions. We distinguish the transition models that have been employed for model-based RL into two main classes—the deterministic and the stochastic

models. Deterministic models do not depend on a random variable for the prediction of the next state, thus the prediction of the model will always be the same for given state and action. On the other hand, stochastic models result in predictions that are defined by a probability distribution over the future state. Figure 6 illustrates and classifies the models that have been used in the literature for model-based RL.

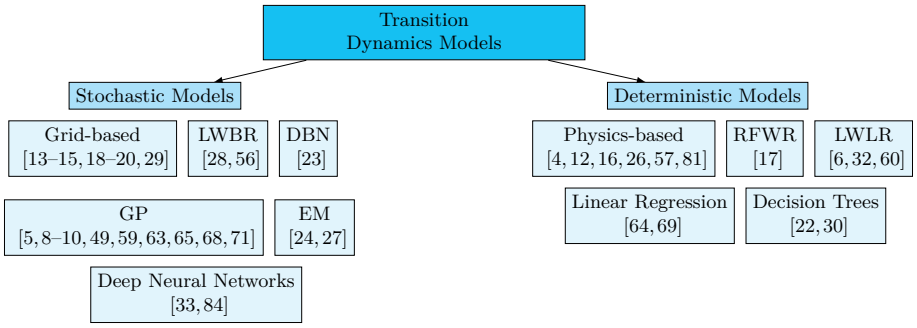


Fig. 6: Overview of transition models literature.

Deterministic Models Physics-based models that describe the transition dynamics are widely used as deterministic models [4, 12, 16, 26, 57, 81]. The main disadvantage of such models is that they contain many factors that are difficult to be analytically expressed, such as friction and dynamics of elastic joints. Another disadvantage is that these models do not take into account potential changes of the robot's environment. To deal with such problems the authors in [15, 64] enhance the physics models of the transition with linear regression for inferring parameters of the models that are hard to be calculated. A combination of a physics-based and learning model is also employed in [17], where a Receptive Field Weighted Regression (RFWR) is used for modeling the error between the physics model and the real robot. RFWR—introduced in [85]—is a local non-parametric model that is updated incrementally. Its predictions are based on triggering receptive fields that define the similarity between the input and the learned data.

Another method, Locally Weighted Linear Regression (LWLR), has been applied for learning the transition model directly from sensory data without any knowledge about the model of the robot [6,32,60]. LWLR was introduced in [86] and is a nonparametric model that fits linear regression models locally on the training data. The predictions of the model's regression coefficients are made using the ordinary least square estimator, weighted by a kernel that provides a measurement of the similarity between new input and learned data. Thus, the data-points that are closer to the input affect the prediction more. Also online updates of a local linear regression model have been proposed employed in [69] for learning contact dynamics of a bipedal robot.

Decision Trees have also been used because of their ability to generalize the learned model well [22, 30]. They comprise a divide-and-conquer approach, since they partition the learned data space in order to make predictions for the inputs. The partition is made according to the C4.5 algorithm [87], which divides the training data using the concept of information gain—an information theory concept—for splitting the data to different nodes until each node is as "pure" as possible.

Stochastic Models The state-of-the-art approach for learning the transition models—and in particular stochastic models—are the Gaussian Processes (GP), applied in [5, 8–10, 49, 59, 63, 65, 68, 71]. Generally, probabilistic models build a distribution over random variables while GP build one over functions. Thus, there is not any assumption about the function f that maps current states and actions to future states. This fact makes GP a powerful learning method. A GP is defined by its mean and a kernel (covariance function). The mean of the GP is assumed to be zero in most cases and a very common choice for kernels are those that belong to the exponential family. There exists a variety of methods for setting the kernel's parameters (hyper-parameters) including greedy search over the hyper-parameters' space and marginal likelihood based methods, which are more widely used.

In more detail, the objective of a GP in the context of transition models, is to infer the function f that generates the noisy observations s'. Assuming Gaussian noise: $s' = f(s, a) + N(0, \sigma_n^2 I)$ and for

notation simplicity $\mathbf{x} = (s, a)$. Quoting Rasmussen in [88], a Gaussian Process can be defined as *"a collection of random variables, any finite number of which have (consistent) joint Gaussian distributions."* Given a dataset of N training samples which are described by D attributes and a set of $N^\star$ testing samples, the mean $\boldsymbol{\mu}(\mathbf{x})$ and the covariance $\boldsymbol{\Sigma}(\mathbf{x}, \mathbf{x}^\star)$ of a GP f are given in equations (18) and (19) respectively.

$$\boldsymbol{\mu}(\mathbf{x}) = \mathbf{E}\left[f(\mathbf{x})\right] \tag{18}$$

$$\boldsymbol{\Sigma}(\mathbf{x}, \mathbf{x}^\star) = \mathbf{E}\left[(f(\mathbf{x}) - \boldsymbol{\mu}(\mathbf{x}))(f(\mathbf{x}^\star) - \boldsymbol{\mu}(\mathbf{x}^\star))\right] \tag{19}$$

In the majority of the applications the mean function of the GP is zero and the focus is given on the selection of the appropriate covariance function — also known as kernel — and its parameters. Kernel functions have an important role since the performance of the GP strongly depends on them. They are classified as stationery or not stationery depending on their invariance to translations in input space. The most used kernel is the squared exponential, given in equation (20), which depends on two parameters; the variance σ^2 and the length-scale λ. The latter controls the distance that the algorithm extrapolates away from the training data.

$$\boldsymbol{\Sigma}(\mathbf{x}, \mathbf{x}^\star) = \sigma^2 exp\left(-\frac{||\mathbf{x} - \mathbf{x}^\star||_2^2}{2\lambda^2}\right) \tag{20}$$

The selection of the kernel's parameter is usually performed by optimizing the log-likelihood of the GP given the training data, as in equation (21) where constant $\beta = -\frac{n}{2}\log 2\pi$. This process corresponds to the minimization of the partial derivative, w.r.t. the covariance parameters θ, of the negative log likelihood. Thus, the minimization of equation (22), where $a = \left(\boldsymbol{\Sigma}(\mathbf{x}, \mathbf{x}) + \sigma_n^2\mathbf{I}\right)\mathbf{s}'$ yields the optimal parameters of the kernel and can be achieved by a variety of function optimization approaches.

$$\log\, p(\mathbf{s}'|\mathbf{X}) = -\frac{1}{2}\mathbf{s}'^T\left(\boldsymbol{\Sigma}(\mathbf{x}, \mathbf{x}) + \sigma_n^2\mathbf{I}\right)^{-1}\mathbf{s}' - \frac{1}{2}\log|\boldsymbol{\Sigma}(\mathbf{x}, \mathbf{x}) + \sigma_n^2\mathbf{I}| - \beta \tag{21}$$

$$-\frac{\partial \log\, p(\mathbf{s}'|\mathbf{X})}{\partial \theta} = -\frac{1}{2}\mathrm{tr}\left(\left(\alpha\alpha^T - \left(\mathbf{\Sigma}(\mathbf{x},\mathbf{x}) + \sigma_n^2\mathbf{I}\right)^{-1}\right)\frac{\partial\left(\mathbf{\Sigma}(\mathbf{x},\mathbf{x}) + \sigma_n^2\mathbf{I}\right)}{\partial\theta}\right) \tag{22}$$

Given a new test input $\mathbf{x}^\star$ its predicted target value $\mathbf{s}'_\star$ corresponds to the mean of the posterior predictive distribution, given in equation (23). On the other hand, the covariance, as given in equation (24), represents the uncertainty of that prediction.

$$\mathbf{s}'_\star = \mathbf{\Sigma}(\mathbf{x}^\star,\mathbf{x})\left(\mathbf{\Sigma}(\mathbf{x},\mathbf{x}) + \sigma_n^2\mathbf{I}\right)^{-1}\mathbf{s} \tag{23}$$

$$\mathrm{cov}\left(\mathbf{s}'_\star\right) = \mathbf{\Sigma}(\mathbf{x}^\star,\mathbf{x}^\star) - \mathbf{\Sigma}(\mathbf{x}^\star,\mathbf{x})\left(\mathbf{\Sigma}(\mathbf{x},\mathbf{x}) + \sigma_n^2\mathbf{I}\right)^{-1}\mathbf{\Sigma}(\mathbf{x}^\star,\mathbf{x}) \tag{24}$$

A common approach for stochastic modeling, especially in the early years of model-based RL, is the partition of the robot's state-space into grids where an action results in a transition on a neighbor grid according to a probability distribution. In [13] the transition model is build online based on empirical estimates derived from the interaction between the robot and the environment. The model is based on statistics about the frequency of visits to each state, the transitions and the immediate rewards. The predictions of the next state are derived by solving a linear system of equations. The Cerebellar Model Articulation Controller (CMAC), a grid-based approach, is used in [15] for learning the transition model. CMAC was introduced in [89] and is a value approximation technique that relies on overlapping grids. The prediction for a new input is based on relevant data of each grid. In [29] the state aggregation method is employed for partitioning the state space. A grid partitions the space and each action can lead to several neighbor grids according to the frequency of occurrence on the training data. The authors in [18, 19] use the CACM algorithm [90] for the discretization of the state space into cells. CACM is based on the adjoining property which ensures transitions only between neighbor cells. Furthermore, table lookup approaches are used in [14, 20].

Locally Weighted Bayesian Regression (LWBR), introduced in [91], is employed in [28, 56] as a model learning algorithm that combines Bayesian inference and LWLR. The regression coefficients are

given a Gaussian prior and the noise is assumed to be described as a gamma distribution. The prediction for each input is made based on its similarity with the trained data as defined by a weight function and is described by a t distribution.

In [24] the authors assume that the transition model is described by a linear combination of the current state and the applied commands with additive Gaussian noise. The unknown parameters of the linear equation are optimized through an Expectation-Maximization (EM) algorithm based on a reference trajectory created by an expert. An EM based approach was also used in [27], where the learning algorithm does not only predict the future state but also the actions that are required in order to follow an optimal trajectory, which is introduced by an expert. The goal of the EM algorithm is to infer the latent variables that describe both the optimal state and actions in each time-step. Another approach that has been used for discrete state-action space is the use of Dynamic Bayesian Networks (DBN) [23]. In that approach the learning of MDPs' transition probabilities corresponds to learning the local structure of DBN. A variant of multinomial logistic regression is used for online training of the DBN.

Deep Learning approaches have been also proposed for learning dynamics models [33, 84]. In [33] the authors propose the use of two deep feed-forward network architectures where the one is responsible for modeling the dynamics and the other for modeling the error and noise. The network's activation function is chooses to be the rectified linear units (ReLU) due to its stability compared to common choices such as the hyperbolic tangent. Another advantage of this approach is the network can handle uncertainty on its inputs and the learning rule derives analytically. Another method for learning forward dynamics of industrial manipulators, the PC-ESN++, is presented in [84] where the authors employ a recurrent neural network with fixed weights as hidden layer. The inputs are decorrelated with an unsupervised learning rule, the predictions derive by performing Bayesian regression and the noise is inferred directly from data. The advantage of the algorithm is that it is memory-less and hence it is much computationally efficient compared to other methods.

A density estimation method for learning transition dynamics has been proposed in [92] and applied in model-based RL [92, 93].

The least-squares conditional density estimation (LSCD) models the transition probabilities as a linear combination of basis-functions and model's parameters. Thus, the algorithm learns those parameters by minimizing a squared difference. The advantage of LSCD is that it allows more generic assumptions about the transition probabilities compared to GP. On the other hand it is not data efficient since it requires much data to learn the transitions. In [93], the authors deal with this problem by performing a supervised dimensionality reduction approach.

6 Applications on Robotics

Model-based RL has been applied to a wide variety of robotics applications ranging from surgical to underwater robotic tasks. Figure 7 organizes the relevant literature around the various types of considered robotic systems.

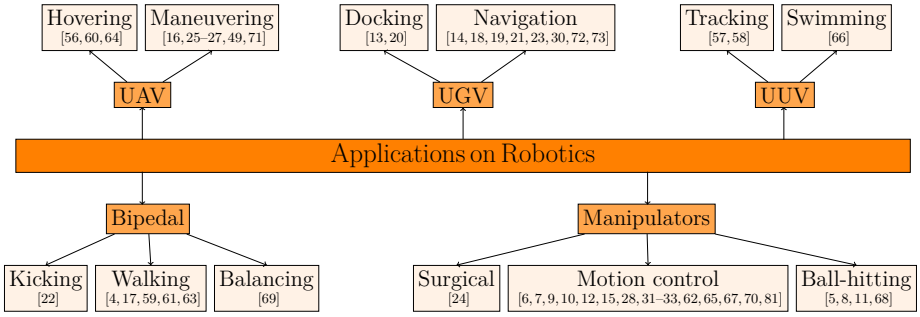


Fig. 7: Overview of literature on applications that use model-based RL for learning robotic tasks.

Autonomous Vehicles The majority of applications deal with autonomous platforms [13, 14, 18–21, 25–27, 30, 56–58, 60, 64, 66, 71–73], including unmanned aerial (UAV), ground (UGV), and underwater (UUV) vehicles. In [56, 60] the authors train a real helicopter's controller to perform hovering and acrobatic maneuvers. The state of the helicopter is described by a 12 dimensional vector and its action space is 6 dimensional. A simulated helicopter is also trained

for learning the hovering task in [26, 64]. Other applications include even more challenging tasks, such as the execution of complex aerobatic maneuvers both on real [25, 27] and simulated UAVs [16, 26]. Furthermore, model-based RL has been used for the control of a blimp [71] whose state is parametrized by a 12 dimensional vector representing the position, orientation and translational and angular velocities. The action space is 3 dimensional and continuous, representing the motor commands on each of the three motors.

Model-based RL has also been used on UGVs for learning navigation and docking tasks. In [14] a Khepera robot is trained to avoid obstacles using 8 infrared sensors. A more complex nonholomonic platform is used in [19] equipped with an on-board microcontroller for processing its infrared and sonar sensors. In [21, 23] the authors used a platform constructed with the Lego Mindstorm NXT kit and tested the navigation ability on different types of terrain. The state of the robot is captured from an external localization system. Furthermore, images from a monocular camera are used in [72] for inferring the distance of obstacles. The control commands are the throttle of the motors and the steering of the car. A real world application is presented in [30], where a car is navigated autonomously in real-time. Besides obstacle avoidance, UGVs are also capable of learning docking tasks. In [13] a Roomba robot is equipped with a camera for receiving information about the environment. Its state is described by two continuous variables—the distance and the angle from the docking location, while the action is a binary variable. Authors in [20], additionally to [13], equipped their platform with sonar sensors for perceiving state information. The state-space of the robot is discrete with 189 states and the action-space is six dimensional. Another interesting application is presented in [73] where an AutoRally vehicle equipped with an on-board GPU learns aggressive driving. Finally, model-based RL has also been used on both real and simulated UAVs for learning cable tracking tasks [57, 58] and swimming gaits [66]. A categorization of the applications on autonomous vehicles is presented in Fig. (8).

Bipedal Robots Bipedal applications include learning penalty kicks [22] and walking gaits [4, 17, 59, 61, 63]. Penalty kicks are learned on an Aldebaran NAO bipedal robot [22] for use at the RoboCup com-

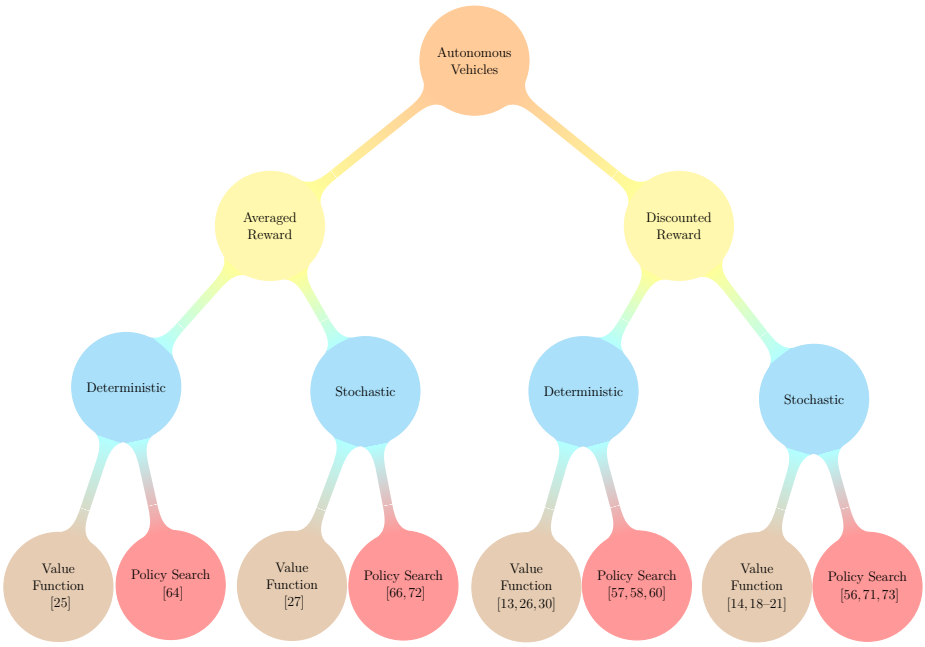


Fig. 8: Categorization of model-based RL applications on autonomous vehicles. Each proposed approach is categorized according to the type of the employed modules.

petition. The action space is three dimensional, two of them move the leg parallel to the ball and the third is the kick action. The state of the robot is two dimensional and described by the relative position of the leg w.r.t the ball which is captured from the robot's head camera. In [4] the authors train a simulated bipedal robot to execute a walking task both on even and uneven terrains; they also tested the ability of the robot to recover after a push. In [31] a model-based RL approach is presented for achieving intrinsically motivated exploration on the iCub humanoid robot. Also in [17], a five degrees of Freedom (DOF) simulated bipedal robot learns a walking task with the ability to cope with disturbances caused from the controller. In [59] and [63] the authors apply the learning algorithms on real bipedal platforms using continuous state and action spaces. Furthermore, in [61] the authors evaluate their proposed algorithm with locomotion experiments where a simulated 18-DOF bipedal learns

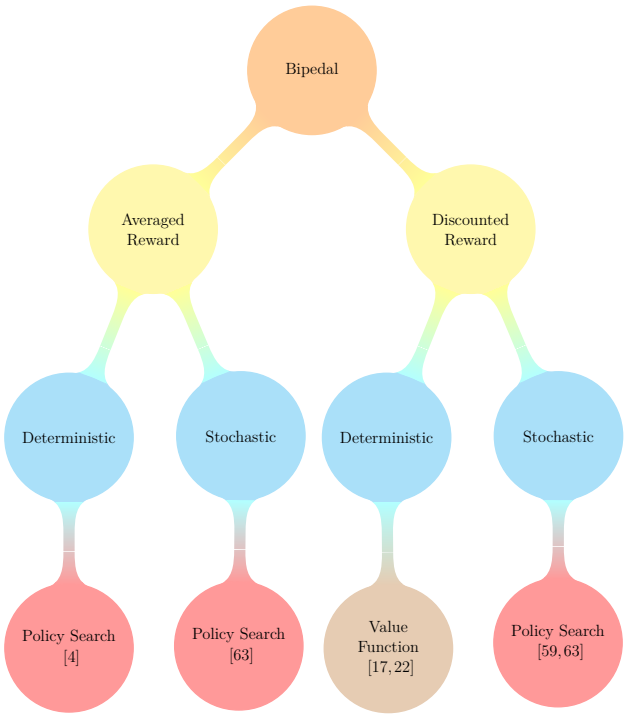


Fig. 9: Categorization of model-based RL applications on bipedal robots. Each proposed approach is categorized according to the type of the employed modules.

walking gaits. A categorization of model-based RL applications on bipedal robots is presented in Fig. (9).

Robotic Manipulators Another popular field of applications deals with the control of robotic manipulators. Task fields range from surgical robotics to motion control of underactuated manipulators [5–12, 15, 24, 28, 32, 62, 65, 67, 68]. The authors in [24] train two Berkeley Surgical Robots to perfom a knot-tie task in high-speed. Furthermore, there exist applications where the authors train robotic manipulators on ball-hitting tasks [5, 8, 11, 68]. In [5, 8] the used platform is a compliant bio-inspired manipulator with spring joints. The state space is continuous and contains positions and velocities of each joint, while the action-space is three dimensional and contains the applied torques. In both cases the robot is trained to perform table-

tennis related tasks. Also, in [11] a 3 DOF robot is trained to perform badminton swings. Learning to perform motion control of a robotic manipulator has been applied on multiple types of platforms such as underactuated manipulators [7, 15, 28], low DOF manipulators (2 or 3 DOF) [6, 7, 9, 10], and more complex high DOF systems [12, 32, 67]. The learned tasks include position control [6, 7, 10, 15], pouring with the PR-2 robot [33], pendulum swinging [28, 32] and more complex manipulation tasks with obstacle avoidance [12, 62]. The manipulators' state-space in the majority of the applications, is continuous and described by the position and acceleration of each joint. The action-space is also continuous and corresponds to the torques applied on actuated joints. A categorization of model-based RL applications on robotic manipulators is presented in Fig. (10).

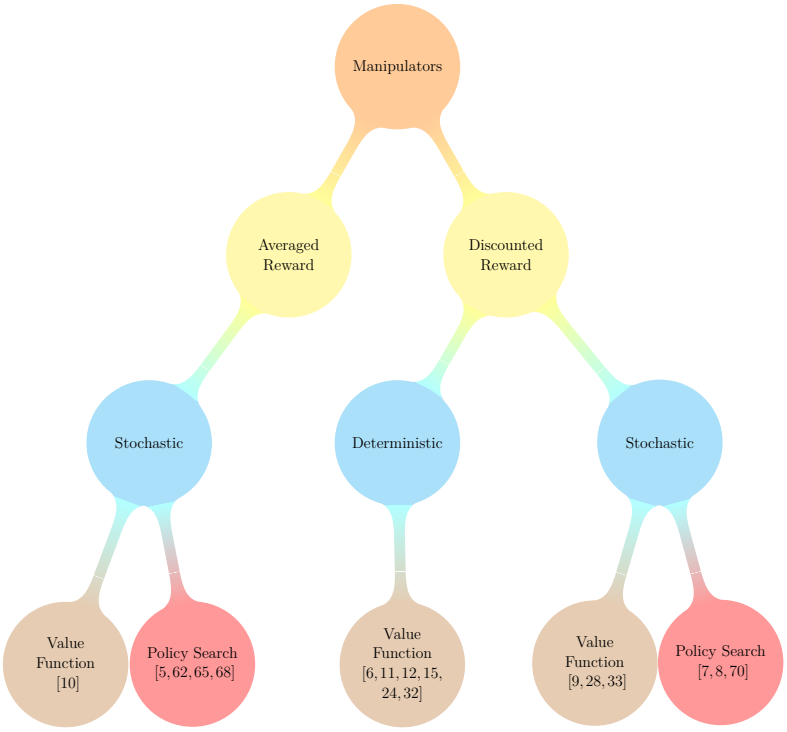


Fig. 10: Categorization of model-based RL applications on robotic manipulators. Each proposed approach is categorized according to the type of the employed modules.

7 Discussion and Conclusion

The previous section reveals the successful deployment of model-based RL in a variety of robotics applications. However, what most—if not all—the surveyed works have in common is that they are dealing with simplistic or "playful" tasks, such as playing table tennis, badminton and pendulum swinging. Even when more challenging tasks are addressed, e.g. maneuvering of autonomous vehicles, what is actually reported is successful tests of very limited practical applicability. Relevant literature evidently lacks usage examples of model-based RL in more "serious" fields requiring reliability and robustness, such as in service or industrial robotics. This observation is not only relevant to model-based RL, but extends to machine learning in general. However, such applications could be greatly benefited by the ability of robotic manipulators to learn and execute tasks in collaboration (or just coexistence) with humans, while always adapting to changes of the environment or the manipulated objects.

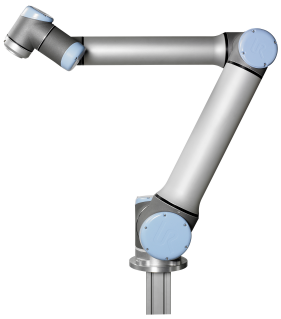
A large portion of the adaptability issues can be resolved by employing RL algorithms for teaching robots. Naturally, the most preferable approach for such tasks appears to be model-based RL, since it requires much fewer interactions with the environment compared to model-free RL. This is due to the use of transition models and is preferable because by minimizing the interactions with the environment the hazard of accidents and the robot's wear and tear are also minimized. As a result, we build upon the analysis of the previous sections to illustrate how model-based RL approaches can be applied to robotic manipulators in the service, industrial and other robotics fields. Take as an example pick and place; a very common (found in many tasks) and potentially challenging (one can consider assembly as a special case of pick and place) operation.

To learn an adaptable pick and place operation the manipulator should be equipped with sensors for perceiving the environment in order to recognize the manipulated object and possible collisions. The collision avoidance is a crucial feature of the system and can be achieved by appropriately defining the returns function which can be a mixture of rewards and costs. The rewards can represent the distance between the current state and the goal state and also the smoothness of the trajectory which derives by the policy optimiza-

tion method. Costs can be introduced for undesired states such as collisions. Another desirable feature is the fast convergence of the policy learning procedure; thus policy iterations are not preferable since they suffer from slow convergence as described in Section 2. On the other hand, the value iteration methods are not appropriate for large continuous state and action spaces because they require computing a state-based or state-action-based value function which is infinite. The same disadvantage also holds for sampling and TD learning methods, in addition to their assumption that there is no prior knowledge about the transition model. DDP from the class of value function methods could be a possible solution, but its requirement for initial demonstrated trajectories reduces the autonomy of the system. Furthermore, Information Theory algorithms, depend on the existence of an initial trajectory which also reduces the autonomy of the system. Thus, policy search methods appear to be more capable of dealing with the requirements posed by collaborative robotics applications. This is due to their ability to reduce the dimensionality of the policy learning problem by parameterizing the policy function and inferring the appropriate parameters. Among these methods, the most promising are the gradient and sampling-based ones.

Given that the transition models are important for the performance of the learning algorithm they should be carefully chosen according to the task. Currently, in the robotics market, there is a trend towards low cost, compliant robotic manipulators. These characteristics make them more affordable and safer for humans, but at the same time pose significant challenges on transition modeling. One of those challenges, is the use of elastic joints that are extremely difficult to model using physics-based approaches—the Baxter robot by Rethink Robotics (Figure 11a) is a typical example of this case. Another challenge is the low quality of the used internal sensors, which—even if it lowers the cost—results in noisy measurements. To make things even more difficult, there also exist manipulators that are not equipped with force/torque sensors, but instead use approximation methods in order to estimate the applied torques—Universal Robots (Figure 11b) models being representative examples. The impact of those issues on model learning is investigated in [94], where the need for a stochastic transition model that is able to learn from noisy sensory data and handle uncertainty in inputs is explored.

(a) Rethnik Robotics Baxter



(b) Universal Robots UR10

Fig. 11: State-of-the-art collaborative robots.

Summarizing, in this paper we presented methods that have been proposed in the literature for learning robotic tasks using model-based RL. Those methods have been classified according to the approach used for the derivation of the policy, which can be based on computing value-functions or policies. Furthermore, different approaches for modeling the transition dynamics have been presented and also methods for defining the returns function. Finally we have proposed a possible outline of a model-based RL approach that could be applied on robotic manipulators for learning new tasks and adapting to changes. The main goal of such an approach is to achieve safer robot-human interaction and lower wear and tear by reducing the interactions between the robot and its environment. It is the strong belief of the authors of this survey that model-based RL can pave the way to more intelligent and adaptive robots. The richness of the relevant literature reveals the maturity of the underlying concepts, as well as their potential in applications requiring more reliability and robustness.

Acknowledgement

This work has been supported by the European Commission through the research project “Sustainable and Reliable Robotics for Part

Handling in Manufacturing Automation (STAMINA)", FP7-ICT-2013-10-610917.

References

1. Deisenroth, M.P.: A Survey on Policy Search for Robotics. Foundations and Trends in Robotics **2** (2011) 1–142
2. Kober, J., Bagnell, J.a., Peters, J.: Reinforcement learning in robotics: A survey. The International Journal of Robotics Research **32** (2013) 1238–1274
3. Kormushev, P., Calinon, S., Caldwell, D.G.: Reinforcement learning in robotics: Applications and real-world challenges. Robotics **2** (2013) 122–148
4. Levine, S., Koltun, V.: Learning complex neural network policies with trajectory optimization. In: Proceedings of the 31st International Conference on Machine Learning (ICML-14). (2014) 829–837
5. Deisenroth, M.P., Englert, P., Peters, J., Fox, D.: Multi-task policy search for robotics. In: IEEE International Conference on Robotics and Automation, IEEE (2014) 3876–3881
6. van Rooijen, J., Grondman, I., Babuška, R.: Learning rate free reinforcement learning for real-time motion control using a value-gradient based policy. Mechatronics **24** (2014) 966–974
7. Wilson, A., Fern, A., Tadepalli, P.: Using trajectory data to improve bayesian optimization for reinforcement learning. The Journal of Machine Learning Research **15** (2014) 253–282
8. Kupcsik, A., Deisenroth, M.P., Peters, J., Loh, A.P., Vadakkepat, P., Neumann, G.: Model-based contextual policy search for data-efficient generalization of robot skills. Artificial Intelligence (2014)
9. Strahl, J., Honkela, T., Wagner, P.: A gaussian process reinforcement learning algorithm with adaptability and minimal tuning requirements. In: Artificial Neural Networks and Machine Learning–ICANN 2014. Springer (2014) 371–378
10. Boedecker, J., Springenberg, J.T., Wulfing, J., Riedmiller, M.: Approximate real-time optimal control based on sparse Gaussian process models. In: 2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), IEEE (2014) 1–8
11. Depraetere, B., Liu, M., Pinte, G., Grondman, I., Babuška, R.: Comparison of model-free and model-based methods for time optimal hit control of a badminton robot. Mechatronics **24** (2014) 1021–1030
12. Guenter, F., Hersch, M., Calinon, S., Billard, A.: Reinforcement Learning for Imitating Constrained Reaching Movements. Advanced Robotics **21** (2007) 1521–1544
13. Shaker, M.R., Yue, S., Duckett, T.: Vision-based reinforcement learning using approximate policy iteration. International Conference on Advanced Robotics (2009)
14. Touzet, C.F.: Neural reinforcement learning for behaviour synthesis. Robotics and Autonomous Systems **22** (1997) 251–281
15. Boone, G.: Efficient reinforcement learning: model-based Acrobot control. Proceedings of International Conference on Robotics and Automation **1** (1997)
16. Abbeel, P., Quigley, M., Ng, A.Y.: Using inaccurate models in reinforcement learning. In: Proceedings of the 23rd international conference on Machine learning - ICML '06, New York, New York, USA, ACM Press (2006) 1–8

17. Morimoto, J., Atkeson, C.G.: Minimax Differential Dynamic Programming: An Application to Robust Biped Walking. Advances in Neural Information Processing Systems 15 (2003) 1539–1546
18. Martínez-Marín, T., Duckett, T.: Fast reinforcement learning for vision-guided mobile robots. Proceedings - IEEE International Conference on Robotics and Automation **2005** (2005) 4170–4175
19. Martinez-Marin, T.: On-line optimal motion planning for nonholonomic mobile robots. In: Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006., IEEE (2006) 512–517
20. Bakker, B., Zhumatiy, V., Gruener, G., Schmidhuber, J.: Quasi-online reinforcement learning for robots. Proceedings - IEEE International Conference on Robotics and Automation **2006** (2006) 2997–3002
21. Leffler, B.R., Littman, M.L., Edmunds, T.: Efficient reinforcement learning with relocatable action models. Proceedings of the 22nd AAAI Conference on Artificial Intelligence (2007) 572–577
22. Hester, T., Quinlan, M., Stone, P.: Generalized model learning for reinforcement learning on a humanoid robot. In: Robotics and Automation (ICRA), 2010 IEEE International Conference on, IEEE (2010) 2369–2374
23. Nguyen, T., Li, Z., Silander, T., Leong, T.Y.: Online Feature Selection for Model-based Reinforcement Learning. Proceedings of the 30th International Conference on Machine Learning (ICML-13) (2013) 498–506
24. Van Den Berg, J., Miller, S., Duckworth, D., Hu, H., Wan, A., Fu, X.Y., Goldberg, K., Abbeel, P.: Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations. Proceedings - IEEE International Conference on Robotics and Automation (2010) 2074–2081
25. Abbeel, P., Coates, A., Ng, A.Y.: Autonomous Helicopter Aerobatics through Apprenticeship Learning. The International Journal of Robotics Research **29** (2010) 1608–1639
26. Ross, S., Bagnell, J.A.: Agnostic system identification for model-based reinforcement learning. Proceedings of the 29th Interational Conference on Machine Learning (2012) 1703–1710
27. Coates, A., Abbeel, P., Ng, A.: Apprenticeship learning for helicopter control. Communications of the ACM (2009)
28. Schneider, J.: Exploiting model uncertainty estimates for safe dynamic control learning. Advances in Neural Information Processing Systems (1997)
29. Kuvayev, L., Sutton, R.: Model-based reinforcement learning with an approximate, learned model. Proceedings of the ninth Yale workshop on adaptive and learning systems (1996) 101–105
30. Hester, T., Quinlan, M., Stone, P.: RTMBA: A real-time model-based reinforcement learning architecture for robot control. IEEE International Conference on Robotics and Automation (2012) 85–89
31. Frank, M., Leitner, J., Stollenga, M., FÃűrster, A., Schmidhuber, J.: Curiosity driven reinforcement learning for motion planning on humanoids. Frontiers in Neurorobotics **7** (2014) 25
32. Atkeson, C.G.: Nonparametric model-based reinforcement learning. In: Advances in Neural Information Processing Systems. (1998) 1008–1014
33. Yamaguchi, A., Atkeson, C.G.: Neural networks and differential dynamic programming for reinforcement learning problems. In: 2016 IEEE International Conference on Robotics and Automation (ICRA), IEEE (2016) 5434–5441

34. Howard, R.: Dynamic Programming and Markov Processes. Technology Press of the Massachusetts Institute of Technology (1960)
35. Bellman, R.E.: Dynamic Programming. Princeton University Press, Princeton (1957)
36. Peters, J., Schaal, S.: Natural actor-critic. Neurocomputing **71** (2008) 1180–1190
37. Peters, J., Vijayakumar, S., Schaal, S.: Reinforcement learning for humanoid robotics. In: Proceedings of the third IEEE-RAS international conference on humanoid robots. (2003) 1–20
38. Amari, S.I.: Natural gradient works efficiently in learning. Neural computation **10** (1998) 251–276
39. Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. Journal of Machine Learning Research **4** (2003) 1107–1149
40. Lagoudakis, M., Parr, R., Littman, M.: Least-squares methods in reinforcement learning for control. In Vlahavas, I., Spyropoulos, C., eds.: Methods and Applications of Artificial Intelligence. Volume 2308 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2002) 249–260
41. Moore, A.W., Atkeson, C.G.: Prioritized sweeping: Reinforcement learning with less data and less time. Machine Learning **13** (1993) 103–130
42. Rasmussen, C.E.: Gaussian processes for machine learning. MIT Press (2006)
43. Brafman, R.I., Tennenholtz, M.: R-max-a general polynomial time algorithm for near-optimal reinforcement learning. The Journal of Machine Learning Research **3** (2003) 213–231
44. Sherstov, A.A., Stone, P.: Improving action selection in mdp's via knowledge transfer. In: AAAI. Volume 5. (2005) 1024–1029
45. Lang, T., Toussaint, M., Kersting, K.: Exploration in relational domains for model-based reinforcement learning. Journal of Machine Learning Research **13** (2012) 3725–3768
46. Martínez, D., Alenya, G., Torras, C.: Relational reinforcement learning with guided demonstrations. Artificial Intelligence (2015)
47. MartÃŋnez, D., AlenyÃă, G., Torras, C.: Safe robot execution in model-based reinforcement learning. In: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on. (2015) 6422–6427
48. Yamaguchi, A., Atkeson, C.G.: Differential dynamic programming with temporally decomposed dynamics. In: Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on. (2015) 696–703
49. Andersson, O., Heintz, F., Doherty, P.: Model-based reinforcement learning in continuous environments using real-time constrained optimization. In: Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI15). (2015)
50. Anderson, B.D., Moore, J.B.: Optimal control: linear quadratic methods. Courier Corporation (2007)
51. Bertsekas, D.P., Bertsekas, D.P., Bertsekas, D.P., Bertsekas, D.P.: Dynamic programming and optimal control. Volume 1. Athena Scientific Belmont, MA (1995)
52. Bradtke, S.J.: Incremental Dynamic Programming for On-line Adaptive Optimal Control. PhD thesis, Amherst, MA, USA (1995) UMI Order No. GAX95-10446.
53. Rummery, G.A., Niranjan, M.: On-line Q-learning using connectionist systems. Technical Report 166, Cambridge University Engineering Department (1994)
54. Watkins, C., Dayan, P.: Technical note: Q-learning. Machine Learning **8** (1992) 279–292
55. Sutton, R.S.: Dyna, an integrated architecture for learning, planning, and reacting. ACM SIGART Bulletin **2** (1991) 160–163

56. Bagnell, J., Schneider, J.: Autonomous helicopter control using reinforcement learning policy search methods. Robotics and Automation, IEEE International Conference on **2** (2001) 1615—-1620
57. El-Fakdi, A., Carreras, M.: Policy gradient based Reinforcement Learning for real autonomous underwater cable tracking. In: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE (2008) 3635–3640
58. El-Fakdi, A., Carreras, M.: Two-step gradient-based reinforcement learning for underwater robotics behavior learning. Robotics and Autonomous Systems **61** (2013) 271–282
59. Morimoto, J., Atkeson, C.G.: Nonparametric representation of an approximated Poincaré map for learning biped locomotion. Autonomous Robots **27** (2009) 131–144
60. Ng, A.Y., Kim, H.J., Jordan, M.I., Sastry, S.: Autonomous helicopter flight via Reinforcement Learning. Advances in Neural Information Processing Systems 16 **16** (2004) 363–372
61. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: Proceedings of The 32nd International Conference on Machine Learning. (2015) 1889–1897
62. Deisenroth, M., Rasmussen, C., Fox, D.: Learning to control a low-cost manipulator using data-efficient reinforcement learning. RSS (2011)
63. Deisenroth, M.P., Calandra, R., Seyfarth, A., Peters, J.: Toward fast policy search for learning legged locomotion. IEEE International Conference on Intelligent Robots and Systems (2012) 1787–1792
64. Koppejan, R., Whiteson, S.: Neuroevolutionary reinforcement learning for generalized helicopter control. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation - GECCO '09, New York, New York, USA, ACM Press (2009) 145
65. Kupcsik, A., Deisenroth, M., Peters, J., Neumann, G.: Data-Efficient Generalization of Robot Skills with Contextual Policy Search. Proceedings of the National Conference on Artificial Intelligence (AAAI) (2013)
66. Levine, S., Koltun, V.: Variational policy search via trajectory optimization. In: Advances in Neural Information Processing. (2013) 207–215
67. Deisenroth, M., Rasmussen, C.E.: PILCO: A model-based and data-efficient approach to policy search. In: 28th International Conference on Machine Learning. (2011) 465–472
68. Englert, P., Paraschos, A., Peters, J., Deisenroth, M.P.: Model-based imitation learning by probabilistic trajectory matching. In: IEEE International Conference on Robotics and Automation. (2013) 1922–1927
69. Mordatch, I., Mishra, N., Eppner, C., Abbeel, P.: Combining model-based policy search with online model learning for control of physical humanoids. In: 2016 IEEE International Conference on Robotics and Automation (ICRA). (2016) 242–248
70. Tangkaratt, V., Mori, S., Zhao, T., Morimoto, J., Sugiyama, M.: Model-based policy gradients with parameter-based exploration by least-squares conditional density estimation. Neural Networks **57** (2014) 128–140
71. Ko, J., Klein, D.J., Fox, D., Haehnel, D.: Gaussian Processes and Reinforcement Learning for Identification and Control of an Autonomous Blimp. Proceedings 2007 IEEE International Conference on Robotics and Automation (2007) 742–747
72. Michels, J., Saxena, A., Ng, A.Y.: High speed obstacle avoidance using monocular vision and reinforcement learning. In: Proceedings of the 22nd international conference on Machine learning, ACM (2005) 593–600

73. Williams, G., Drews, P., Goldfain, B., Rehg, J.M., Theodorou, E.A.: Aggressive driving with model predictive path integral control. In: 2016 IEEE International Conference on Robotics and Automation (ICRA). (2016) 1433–1440
74. Baxter, J., Bartlett, P.L.: Direct gradient-based reinforcement learning. In: Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on. Volume 3., IEEE (2000) 271–274
75. Girard, A., Rasmussen, C.E., Candela, J.Q., Murray-Smith, R.: Gaussian process priors with uncertain inputs application to multiple-step ahead time series forecasting. In Becker, S., Thrun, S., Obermayer, K., eds.: Advances in Neural Information Processing Systems 15. MIT Press (2003) 545–552
76. Deisenroth, M.P.: Efficient reinforcement learning using Gaussian processes. Volume 9. KIT Scientific Publishing (2010)
77. Ng, A.Y., Jordan, M.: PEGASUS: a policy search method for large MDPs and POMDPs. In: Sixteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc. (2000) 406–415
78. Peters, J., Mulling, K., Altun, Y.: Relative entropy policy search. In: Twenty-Fourth AAAI Conference on Artificial Intelligence. (2010)
79. Theodorou, E., Buchli, J., Schaal, S.: A generalized path integral control approach to reinforcement learning. Journal of Machine Learning Research **11** (2010) 3137–3181
80. Pan, Y., Theodorou, E., Kontitsis, M.: Sample efficient path integral control under uncertainty. In Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R., eds.: Advances in Neural Information Processing Systems 28. Curran Associates, Inc. (2015) 2314–2322
81. Colomé, A., Planells, A., Torras, C.: A friction-model-based framework for reinforcement learning of robotic tasks in non-rigid environments. In: 2015 IEEE International Conference on Robotics and Automation (ICRA), IEEE (2015) 5649–5654
82. Theodorou, E., Buchli, J., Schaal, S.: Reinforcement learning of motor skills in high dimensions: A path integral approach. In: Robotics and Automation (ICRA), 2010 IEEE International Conference on, IEEE (2010) 2397–2403
83. Kober, J., Peters, J.R.: Policy search for motor primitives in robotics. In: Advances in neural information processing systems. (2009) 849–856
84. Polydoros, A.S., Nalpantidis, L.: A reservoir computing approach for learning forward dynamics of industrial manipulators. In: Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on, IEEE (2016) 612–618
85. Schaal, S., Atkeson, C.G.: Constructive incremental learning from only local information. Neural Computation **10** (1997) 2047–2084
86. Atkeson, C.G., Moore, A.W., Schaal, S.: Locally weighted learning for control. In: Lazy learning. Springer (1997) 75–113
87. Quinlan, J.: Induction of decision trees. Machine Learning **1** (1986) 81–106
88. Rasmussen, C.E.: Gaussian processes in machine learning. In: Advanced lectures on machine learning. Springer (2004) 63–71
89. Albus, J.S.: A new approach to manipulator control: The cerebellar model articulation controller (CMAC). Journal of Dynamic Systems, Measurement, and Control **97** (1975) 220–227
90. Zufiria, P., Martínez-Marín, T.: Improved optimal control methods based upon the adjoining cell mapping technique. Journal of Optimization Theory and Applications **118** (2003) 657–680

91. Andrew Moore, J.S.: Memory-based stochastic optimization. In Touretzky, D., Mozer, M., Hasselm, M., eds.: Neural Information Processing Systems 8. Volume 8., MIT Press (1996) 1066–1072
92. Sugiyama, M., Takeuchi, I., Suzuki, T., Kanamori, T., Hachiya, H., Okanohara, D.: Least-squares conditional density estimation. IEICE Transactions on Information and Systems **93** (2010) 583–594
93. Tangkaratt, V., Morimoto, J., Sugiyama, M.: Model-based reinforcement learning with dimension reduction. Neural Networks **84** (2016) 1–16
94. Polydoros, A.S., Nalpantidis, L., Kruger, V.: Real-time deep learning of robotic manipulator inverse dynamics. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). (2015) 3442–3448