Parametric Policy Gradients for Robotics

Frank Sehnke, Thomas Rückstieß, Martin Felder and Jürgen Schmidhuber

Abstract— **Slow convergence is a major problem for policy gradient methods. It is a consequence of the fact that the state-action histories used to estimate the gradient are obtained by repeatedly sampling from a probabilistic policy. Given that histories vary greatly even for a fixed policy, gradient estimates obtained by perturbing the policy are bound to be noisy. Lately, an alternative approach was proposed, in which the policy perturbations are replaced by perturbations of the parameters of a controller. This greatly reduces the variance in the gradient estimates, since a complete history can be generated from a single sample in parameter space. Moreover, because the parameters are perturbed directly, the method can be applied to non-differentiable controllers. We will show that this new method is much better suited for realistic learning tasks of humanoid robots than other methods usually used. We will examine the relationship of the new method to other fields of machine learning in robotics. Experiments on two kinds of simulations demonstrate the superior performance to evolution strategies, finite difference methods and existing policy gradient methods such as REINFORCE and natural actor critic. Finally we show the successful learning of grasping an object from different positions with a 7 DoF robot arm simulation as a proof of our claims.**

I. Introduction

Policy gradient methods are among the most effective optimization strategies for complex, high dimensional reinforcement learning tasks [1], [2], [3], [4]. However, a significant problem with policy gradient algorithms such as REINFORCE [5] and natural actor critic (NAC) [4] is that the high variance in their gradient estimates leads to slow convergence. Various approaches have been proposed to reduce the variance [6], [7], [2], [8].

However, none of these methods address the fundamental cause of high variance, which is that repeatedly sampling from a probabilistic policy leads to noisy rewards. Furthermore, the degree of noise increases with the length of the history, since each state depends on the entire past of the trajectory. The approach of Policy Gradients with Parameter-based Exploration (PGPE) [9], is to replace the usual explicit policy with an implicit policy defined by a distribution over the parameters of a controller. The parameters are sampled from this distribution at the start of each sequence, and are kept constant throughout the whole sequence. Because the reward for each sequence depends on only one sample, the gradient estimates are far less noisy. Obviously, this holds for stochastic environments as well.

All authors are with Faculty of Robotics and Embedded Systems, Computer Science, Technische Universität München, Germany

Jürgen Schmidhuber is also with IDSIA, Manno-Lugano, Switzerland sehnke@in.tum.de rueckste@in.tum.de felder@in.tum.de juergen.schmidhuber@in.tum.de

Another advantage of PGPE is that it optimizes the controller parameters directly. By contrast, a standard policy gradient method must first determine the reward gradient with respect to the policy, then differentiate the parameters with respect to that. This has two drawbacks. Firstly, it assumes that the controller is differentiable, which our approach does not. Secondly it makes optimization more difficult since, for example, two very different parameter settings could determine very similar policies.

The contents of this paper are as follows. Section II summarizes the derivation of PGPE from the general framework of episodic reinforcement learning, illustrating the differences between it and standard policy gradient methods such as REINFORCE [5]. Finally it presents pseudocode for the complete algorithm. Section III evaluates PGPE on two reinforcement learning experiments and compares its performance with REINFORCE, evolution strategies (ES) [10], Simultaneous Perturbation Stochastic Adaptation (SPSA) [13] and NAC. It finally shows the solution of a learning task on the CCRL [11] robot simulation. A conclusion and outlook is given in Section IV.

II. Method

In what follows we summarize the derivation of the PGPE algorithm presented in [9] from the general framework of episodic reinforcement learning in a Markovian environment. In doing so we highlight the differences between PGPE and policy gradient methods such as REINFORCE, and discuss why these differences lead to more accurate parameter gradient estimates. We also highlight the special advantages of PGPE in humanoid robot reinforcement learning.

A. Policy Gradients with Parameter-Based Exploration

Consider an agent interacting with an environment. Denote the state of the environment at time t as s_t and the action at time t as a_t. Because we are interested in continuous state and action spaces (usually required for control tasks), we represent both a_t and s_t with real valued vectors. We assume that the environment is Markovian, i.e. that the current state-action pair defines a probability distribution over the possible next states $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$. We further assume that the actions depend stochastically on the current state and some real valued vector θ of agent parameters: $a_t \sim p(a_t|s_t, \theta)$. Lastly, we assume that each state-action pair produces a scalar reward $r_t(a_t, s_t)$. We refer to a length T sequence of state-action pairs produced by an agent as a *history* $h = [s_{1:T}, a_{1:T}]$ (elsewhere in the literature such sequences are also referred to as *trajectories* or *roll-outs*).

Given the above formulation we can associate a cumulative reward r with each history h by summing over the rewards at each time step: $r(h) = \sum_{t=1}^{T} r_t$. In this setting, the goal of reinforcement learning is to find the parameters θ that maximize the agent's expected reward

$$J(\theta) = \int_H p(h|\theta) r(h) dh \tag{1}$$

An obvious way to maximize $J(\theta)$ is to find $\nabla_\theta J$ and use it to carry out gradient ascent. Noting that the reward for a particular history is independent of θ, and using the standard identity $\nabla_x y(x) = y(x) \nabla_x \log x$, we can write

$$\nabla_\theta J(\theta) = \int_H p(h|\theta) \nabla_\theta \log p(h|\theta) r(h) dh \tag{2}$$

Since the environment is Markovian, and since the states are conditionally independent of the parameters given the agent's choice of actions, we can write $p(h|\theta) = p(s_1) \Pi_{t=1}^{T} p(s_{t+1}|s_t, a_t) p(a_t|s_t, \theta)$. Substituting this into (2) gives

$$\nabla_\theta J(\theta) = \int_H p(h|\theta) \sum_{t=1}^{T} \nabla_\theta p(a_t|s_t, \theta) r(h) dh \tag{3}$$

Clearly, integrating over the entire space of histories is unfeasible, and we therefore resort to sampling methods:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} \nabla_\theta p(a_t^n|s_t^n, \theta) r(h^n) \tag{4}$$

where the histories h^i are chosen according to $p(h^i|\theta)$. The question then is how to model $p(a_t|s_t, \theta)$. In policy gradient methods such as REINFORCE, the parameters θ are used to determine a probabilistic *policy* $\pi_\theta(a_t|s_t) = p(a_t|s_t, \theta)$. A typical policy model would be a parametric function approximator whose outputs define the probabilities of taking different actions. In this case the histories can be sampled by choosing an action at each time step according to the policy distribution, and the final gradient can be calculated by differentiating the policy with respect to the parameters. However, the problem is that sampling from the policy on every time step leads to an excessively high variance in the sample over histories, and therefore in the estimated gradient.

PGPE addresses the variance problem by replacing the probabilistic policy with a probability distribution over θ, that is

$$p(a_t|s_t, \rho) = \int_\Theta p(\theta|\rho) \delta_{F_\theta(s_t), a_t} d\theta, \tag{5}$$

where ρ are the parameters determining the distribution over θ, $F_\theta(s_t)$ is the (deterministic) action chosen by the model with parameters θ in state s_t, and δ is the usual Dirac delta function. The advantage of this approach is that, because the actions are deterministic, an entire history can be generated using a single parameter sample, thereby reducing the variance in the gradient estimate. As an added benefit the parameter gradient is estimated by direct perturbations, without having to backpropagate any derivatives, which allows the use of non-differentiable controllers.

The expected reward with a given ρ is

$$J(\rho) = \int_\Theta \int_H p(h, \theta|\rho) r(h) dh d\theta \tag{6}$$

Differentiating this with respect to ρ and applying the log trick as before we get

$$\nabla_\rho J(\rho) = \int_\Theta \int_H p(h, \theta|\rho) \nabla_\rho \log p(h, \theta|\rho) r(h) dh d\theta \tag{7}$$

Noting that h is conditionally independent of ρ given θ, we have $p(h, \theta|\rho) = p(h|\theta) p(\theta|\rho)$ and therefore $\nabla_\rho \log p(h, \theta|\rho) = \nabla_\rho p(\theta|\rho)$. Substituting this into (6) we get

$$\nabla_\rho J(\rho) = \int_\Theta \int_H p(h|\theta) p(\theta|\rho) \nabla_\rho \log p(\theta|\rho) r(h) dh d\theta \tag{8}$$

In principle we can again use sampling methods, this time by first choosing θ from $p(\theta|\rho)$, then running the agent to generate h from $p(h|\theta)$:

$$\nabla_\rho J(\rho) \approx \frac{1}{N} \sum_{n=1}^{N} \nabla_\rho \log p(\theta|\rho) r(h^n) \tag{9}$$

In this paper we assume that ρ consists of a set of means $\{\mu_i\}$ and standard deviations $\{\sigma_i\}$ that determine an independent normal distribution for each parameter θ_i in θ. (More complex forms for the dependency of θ on ρ could be used, at the expense of higher computational cost). Some rearrangement gives the following forms for the derivative of $\log p(\theta|\rho)$ with respect to μ_i and σ_i:

$$\nabla_{\mu_i} \log p(\theta|\rho) = \frac{(\theta_i - \mu_i)}{\sigma_i^2} \tag{10}$$

$$\nabla_{\sigma_i} \log p(\theta|\rho) = \frac{(\theta_i - \mu_i)^2 - \sigma_i^2}{\sigma_i^3}, \tag{11}$$

which can then be substituted into (9) to approximate the μ and σ gradients.

B. Sampling with a baseline

Given enough samples, (9) can determine the reward gradient to arbitrary accuracy. However each sample requires rolling out an entire state-action history, which is expensive. Following Williams [5], we obtain a cheaper gradient estimate by drawing a single sample θ and comparing its reward r to a baseline reward b given by a moving average over previous samples. Intuitively, if $r > b$ we adjust ρ so as to increase the probability of θ, and $r < b$ we do the opposite. If (again following Williams) we use a step size $\alpha_i = \alpha \sigma_i^2$ in the direction of positive gradient, where α is a constant, we get the following parameter update equations:

$$\Delta\mu_i = \alpha(r - b)(\theta_i - \mu_i) \tag{12}$$

$$\Delta\sigma_i = \alpha(r - b) \frac{(\theta_i - \mu_i)^2 - \sigma_i^2}{\sigma_i} \tag{13}$$

Algorithm 1 The PGPE Algorithm: $\mathbf{T}$ and $\mathbf{S}$ are $P \times N$ matrices with P the number of parameters and N the number of histories. The baseline is updated accordingly after each step.

Initialize μ to μ_{init}
Initialize σ to σ_{init}

while TRUE **do**
  for $n = 1$ to N **do**
    draw perturbation $\epsilon^n \sim \mathcal{N}(\mathbf{0}, \mathbf{I}\sigma^2)$
    $\theta^n = \mu + \epsilon^n$
    evaluate $r^n = r(h(\theta^n))$
  end for

  $\mathbf{T} = [\, t_{ij} \,]_{ij}$ with $t_{ij} := \epsilon_i^j$
  $\mathbf{S} = [\, s_{ij} \,]_{ij}$ with $s_{ij} := \frac{(\epsilon_i^j)^2 - \sigma_i^2}{\sigma_i}$
  $\mathbf{r} = [(r^1 - b), \ldots, (r^N - b)]^{\mathrm{T}}$

  update $\mu = \mu + \alpha \mathbf{T}\mathbf{r}$
  update $\sigma = \sigma + \alpha \mathbf{S}\mathbf{r}$
  update baseline b accordingly
end while

C. Relationship to Other Algorithms

In this section we compare the properties of PGPE with REINFORCE, SPSA and ES. Figure 1 shows an overview of the relationship of PGPE to the other fields of compared learning methods.

a) Evolution Strategies with local mutation operator: There exist some similarities of PGPE to ES with the local mutation operator. In both cases exploration is done in parameter space with a standard deviation per dimension. Also control over exploration is given by adapting these standard deviations through the learning process. However,

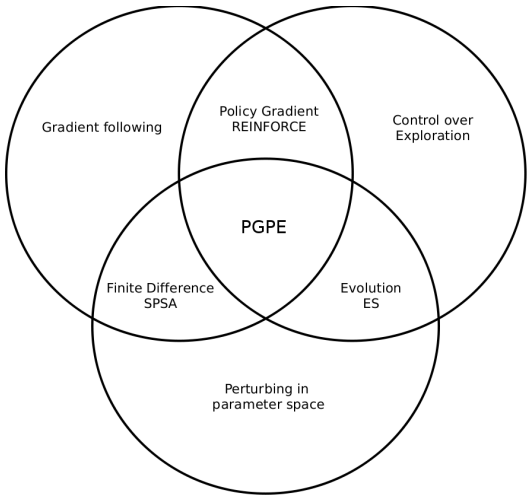

Fig. 1. Relationship of PGPE to other methodological fields.

the convergence speed in the given experiments is slower for ES. This is partly because, as well as having stochastic mutations:

$$\theta_i(t) = \theta_i(t-1) + \sigma_i(t) N(0,1), \tag{14}$$

ES has stochastic updates to the standard deviations of the mutations:

$$\sigma_i(t) = \sigma_i(t-1) e^{\tau_g N(0,1) + \tau_l N(0,1)} \tag{15}$$

and the coupling of these two stochastic processes slows down convergence. Derandomized ES [12] addresses that problem by replacing Eqn. (15) with a deterministic standard deviation update rule, based on the change in parameters between the parent and child generation.

Tracking a population has advantages in the early phase of search, when broad, relatively undirected exploration is desirable. This is particularly true for the multimodal fitness spaces typical of realistic control tasks. However in later phases convergence is usually more efficient with gradient based methods.

b) Simultaneous Perturbation Stochastic Adaptation: Two main differences separate SPSA from PGPE. First the uniform sampling of perturbations from SPSA is replaced by Gaussian sampling, with the finite differences gradient correspondingly replaced by the likelihood gradient. Second, the variances of the perturbations per parameter dimension are turned into free parameters and trained with the rest of the model.

The parameter update rule for SPSA is:

$$\theta_i(t+1) = \theta_i(t) - \alpha \frac{y_+ - y_-}{2\epsilon} \tag{16}$$

with $y_+ = r(\theta + \Delta\theta)$ and $y_- = r(\theta - \Delta\theta)$, where $r(\theta)$ is the evaluation function and $\Delta\theta$ is drawn from a Bernoulli distribution scaled by the time dependent step size $\epsilon(t)$, i.e. $\Delta\theta_i(t) = \epsilon(t) \cdot rand[-1, 1]$ In addition, a set of metaparameters is used to help SPSA converge. ϵ decays according to $\epsilon(t) = \frac{\epsilon(0)}{t^\gamma}$ with $0 < \gamma < 1$. Similarly, α decreases over time, with $\alpha = a/(t + A)^E$ for some fixed a, A and E [13]. The choice of initial parameters $\epsilon(0), \gamma$, a, A and E is critical to the performance of SPSA. [14] provides some guidance on picking these coefficients (note that the nomenclature differs from that used here).

By using PGPE (applying the above mentioned changes) the number of free parameters in the algorithm is reduced drastically to 3. Additionally we found that the parameters $\alpha_\mu = 0.2$, $\alpha_\sigma = 0.1$ and $\sigma_{init} = 2.0$ worked very well for a wide variety of tasks in contrast to the metaparametrs of SPSA that vary enormous dependend on the actual problem.

c) REINFORCE: We previously asserted that the lower variance of PGPE's gradient estimates is partly due to the fact that PGPE requires only one parameter sample per history, whereas REINFORCE requires samples every time step. This suggests that reducing the frequency of REINFORCE perturbations should improve its gradient estimates.

Fig. 2 shows the performance of episodic REINFORCE with a perturbation probability of 1, 0.5, 0.25, and 0.125 per

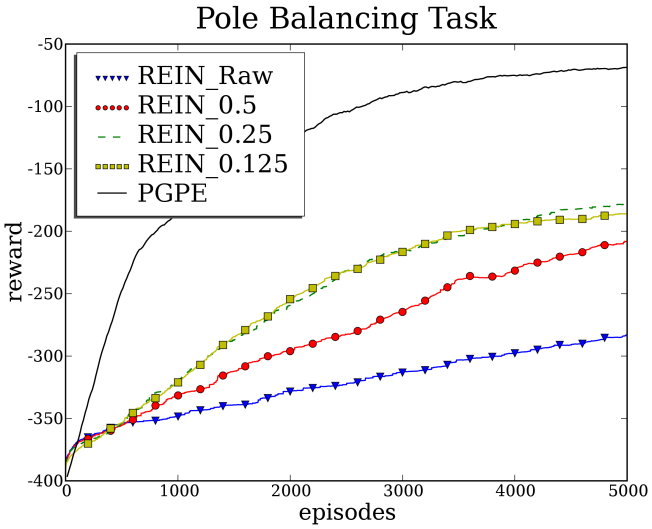

Fig. 2. REINFORCE on the pole balancing task, with various frequencies of action perturbation. PGPE is shown for reference.

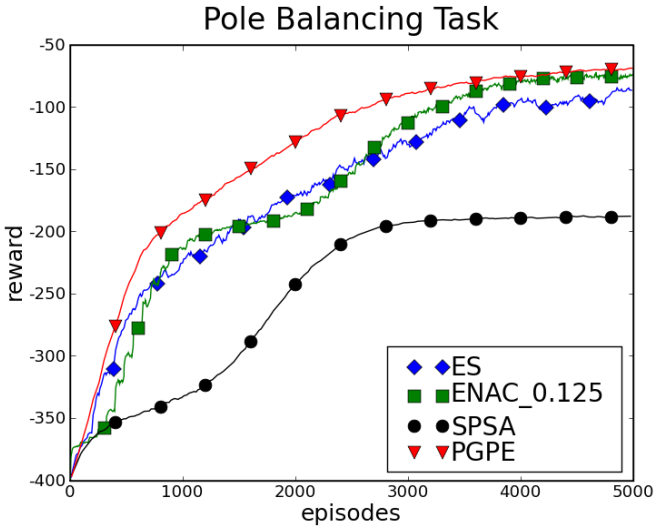

Fig. 3. PGPE compared to ES, SPSA and NAC on the extended pole balancing benchmark.

time step. In general, performance improved with decreasing perturbation probability. However the difference between 0.25 and 0.125 is negligible. This is because reducing the number of perturbations constrains the range of exploration at the same time as it reduces the variance of the gradient, leading to a saturation point beyond which performance does not increase. Note that the above trade off does not exist for PGPE, because a single perturbation of the parameters can lead to a large change in behaviour. In all experiments of section III we used REINFORCE and NAC with the best found perturbation probability instead of the original algorithms to reach an useful comparison.

III. Experiments

In this section we compare PGPE with REINFORCE, NAC, SPSA and ES on two simulated control scenarios to show the practicability of PGPE for robot learning. In all experiments we used ES with a local mutation operator. We did not examine correlated mutation and CMA-ES because both mutation operators add $n(n-1)$ strategy parameters to the genome: this would lead to prohibitive memory costs, if we consider controllers with a realistic number of parameters (e.g. more than 1000 parameters for the controller in the robust standing task). In addition, the local mutation operator of ES is more similar to the perturbations in PGPE. All plots show the average results of 40 independent runs. All experiments were conducted with the optimal meta-parameters found in preliminary experiments for each algorithm.

A. Pole balancing

The first scenario is the extended pole balancing benchmark as described in [15]. In contrast to [15], however, we do not initialize the controller with a previously chosen stabilizing policy but start with random controller policies. We introduce this scenario for reference and comparison reasons. In this task the agent's goal is to maximize the length of time a movable cart can balance a pole upright in the center of a track. The agent's inputs are the angle and angular velocity of the pole and the position and velocity of the cart. The agent is represented by a linear controller with four inputs and one output unit. The simulation is updated 50 times a second. The initial position of the cart and angle of the pole are chosen randomly.

Figure 3 shows the performance of the various methods on the pole balancing task. All algorithms quickly learned to balance the pole, and all eventually learned to do so in the center of the track. PGPE was both the fastest to learn and the most effective algorithm on this benchmark.

B. Biped Robot Standing Task

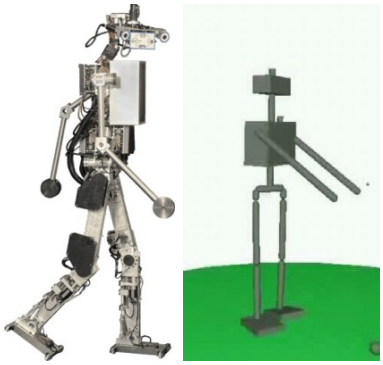

Fig. 4. The real Johnnie robot and its simulation. Courtesy Institute of Applied Mechanics [16]

The task in this scenario was to keep a simulated biped robot standing while perturbed by external forces. The simulation, based on the biped robot Johnnie [16] was implemented using the Open Dynamics Engine. The lengths and masses of the body parts, the location of the connection points, and the range of allowed angles and torques in the joints were matched with those of the original robot. Due to the difficulty of accurately simulating the robot's feet,

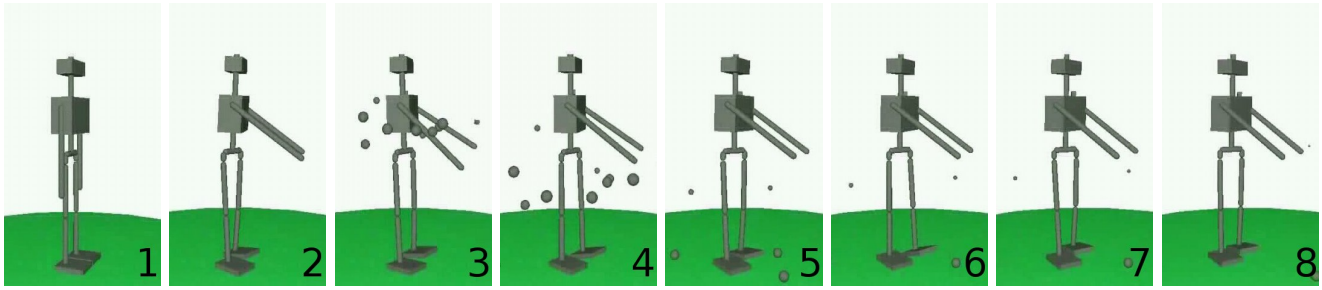

Fig. 6. From left to right, a typical solution which worked well in the simple robust standing task is shown: 1. Initial posture. 2. Stable posture. 3. Perturbation. 4. - 7. Backsteps right, left, right, left. 8. Stable posture regained.

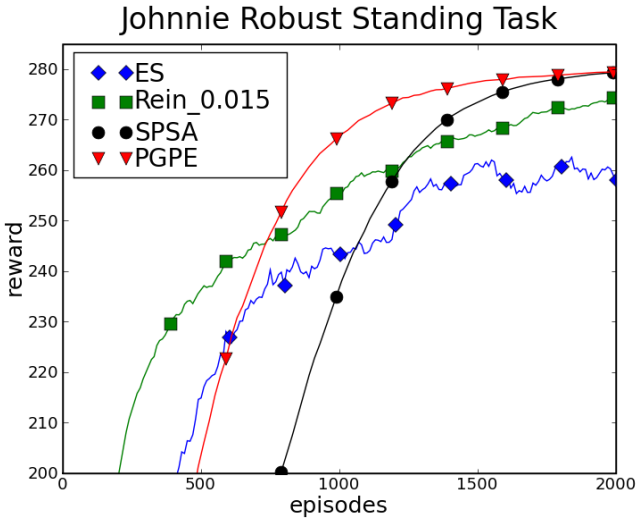

Fig. 5. PGPE compared to ES, SPSA and REINFORCE on the robust standing benchmark.

the friction between them and the ground was approximated with Coulomb friction. The framework has 11 degrees of freedom and a 41 dimensional observation vector (11 angles, 11 angular velocities, 11 forces, 2 pressure sensors in feet, 3 degrees of orientation and 3 degrees of acceleration in the head). The controller was a Jordan network [17] with 41 inputs, 20 hidden units and 11 output units. The aim of the task is to maximize the height of the robot's head, up to the limit of standing completely upright. The robot is continually perturbed by random forces that would knock it over if it did not react. As can be seen from the results in Fig. 5, the task was relatively easy, and all the methods were able to quickly achieve a high reward. REINFORCE learned especially quickly, and outperformed PGPE in the early stages of learning. However PGPE overtook it after about 600 training episodes.

Figure 6 shows a typical solution of the robust standing task with a reward outcome of 279. For more detailed views of the solution please refer to the submitted video.

C. Grasping Task

The task in this scenario was to grasp an object from different positions. The simulation, based on the CCRL robot [11] was implemented using the Open Dynamics

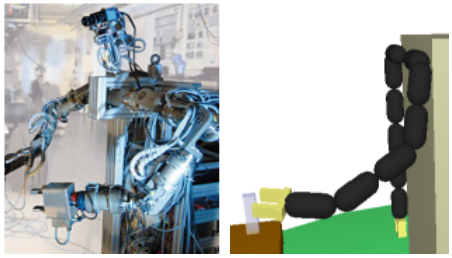

Fig. 7. The real CCRL robot and its simulation. Courtesy Institute of Automatic Control Engineering [11]

Engine. The lengths and masses of the body parts and the location of the connection points were matched with those of the original robot. Friction was again approximated with Coulomb friction. The framework has 8 degrees of freedom and a 35 dimensional observation vector (8 angles, 8 angular velocities, 8 forces, 2 pressure sensors in hand, 3 degrees of orientation and 3 values of position in hand, 3 values of position of object). The controller was a Jordan network [17] with 35 inputs, 10 hidden units and 8 output units. The aim of the task is to get hold on the object and lift it up from the table. The object is located at random positions in the reachable area of the table. The task was learned in 4 phases: First phase was to grasp the object that is located at a fixed position at the edge of the table. Second phase was to grasp the object located on the table apart from the edge. Third phase was to grasp the object at positions normally distributed around the center of the reachable region (standard deviation of 10cm). The last phase was to grasp the object from equally distributed positions in the reachable area. The task was learned incrementally. Every phase was learned with 10.000 episodes and used the final controller of the preceding phase. Figure 8 shows a typical solution of the grasping task. For more detailed views of the solution please refer to the submitted video.

D. Discussion

One general observation from our experiments was that the longer the episodes the more PGPE outperformed policy gradient methods. This is not surprising, since the variance of the REINFORCE gradient estimates increases with the number of action samples. However it is an important benefit, since most interesting real-world problems require much

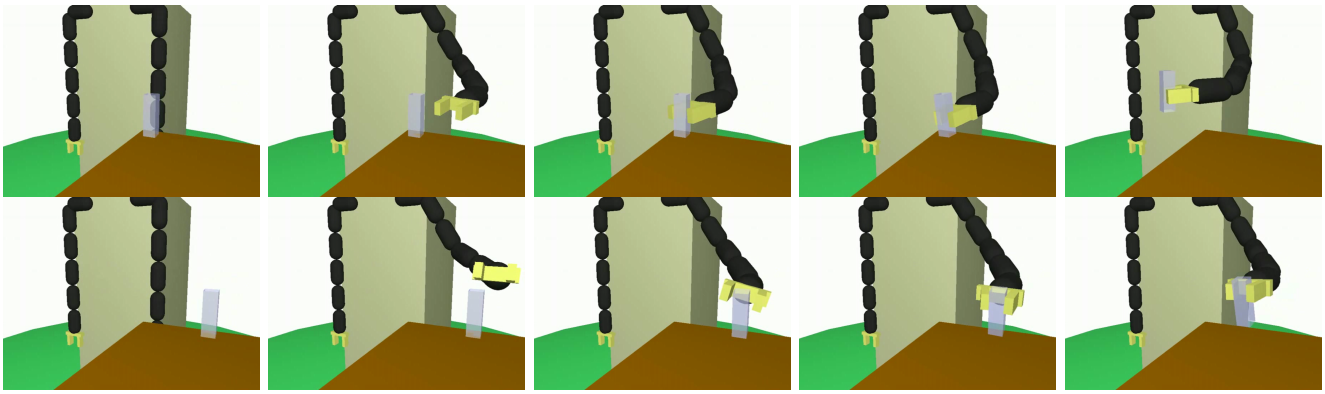

Fig. 8. From left to right, a typical solution which worked well in the grasping task is shown for 2 different positions of the object with the same controller: 1. Initial posture. 2. Approach. 3. Enclose. 4. Take hold. 5. Lift.

longer episodes than our experiments [1], [2], [18].

IV. Conclusion and Outlook

We have evaluated PGPE, an algorithm for episodic reinforcement learning based on a gradient based search through model parameter space on sophisticated robotic learning tasks. We showed how PGPE compares to other related fields. and explained why PGPE leads to lower variance gradient estimates than those obtained by policy gradient methods. We compared PGPE to a range of stochastic optimization algorithms on two control tasks, and found that it gave superior performance in every case.

A possible objection to PGPE is that the parameter space is generally higher dimensional than the action space, and therefore has higher sampling complexity. However, standard policy gradient methods in fact train the same number of parameters — in PGPE they are just trained explicitly instead of implicitly. Additionally, recent results [15] indicate that this drawback was overestimated in the past. In this paper, we presented experiments were PGPE successfully trains a controller with more than 1000 parameters. Another issue is that PGPE, at least in its present form, is episodic, because the parameter sampling is carried out once per history. This contrasts with policy gradient methods, which can be applied to infinite horizon settings as long as frequent rewards can be computed.

In the grasping task the position of the object is assumed to be given by the robot vision system. Such position data is usually noisy. This noisiness is not included in the simulation up to now, but will be part of future experiments.

An interesting future approach is to use a classifier system as controller, since PGPE does not require a differentiable controller A classifier system can resemble an easier controller for learning these tasks and is readable by humans. A trained classifier system could even be easily translated in program code to resemble a module for a hierarchical control mechanism.

V. Acknowledgement

This work was funded within the Excellence Cluster *Cognition for Technical Systems* (CoTeSys) by the German Research Foundation (DFG).

References

[1] H. Benbrahim and J. Franklin, "Biped dynamic walking using reinforcement learning," *Robotics and Autonomous Systems Journal*, 1997.
[2] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *IROS-2006*, Beijing, China, 2006, pp. 2219 – 2225.
[3] N. Schraudolph, J. Yu, and D. Aberdeen, "Fast online policy gradient learning with smd gain vector adaptation," in *Advances in Neural Information Processing Systems 18*, Y. Weiss, B. Schölkopf, and J. Platt, Eds. Cambridge, MA: MIT Press, 2006.
[4] J. Peters, S. Vijayakumar, and S. Schaal, "Natural actor-critic," in *ECML-2005*, 2005, pp. 280–291.
[5] R. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning." *Machine Learning*, vol. 8, pp. 229–256, 1992.
[6] J. Baxter and P. L. Bartlett, "Reinforcement learning in POMDPs via direct gradient ascent," in *Proc. 17th International Conf. on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 2000, pp. 41–48.
[7] D. Aberdeen, "Policy-gradient algorithms for partially observable markov decision processes," Ph.D. dissertation, Australian National University, 2003.
[8] R. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *NIPS-1999*, pp. 1057–1063, 2000.
[9] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber, "Policy gradients with parameter-based exploration for control," in *Springer LNCS proceedings of ICANN (in print)*, 2008.
[10] H. Schwefel, *Evolution and optimum seeking*. Wiley New York, 1995.
[11] M. Buss and S. Hirche, "Institute of Automatic Control Engineering, TU München, Germany," 2008, http://www.lsr.ei.tum.de/.
[12] N. Hansen and A. Ostermeier, "Completely Derandomized Self-Adaptation in Evolution Strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.
[13] J. Spall, "An overview of the simultaneous perturbation method for efficient optimization," *Johns Hopkins APL Technical Digest*, vol. 19, no. 4, pp. 482–492, 1998.
[14] ——, "Implementation of the simultaneous perturbation algorithm for stochastic optimization," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 34, no. 3, pp. 817–823, 1998.
[15] M. Riedmiller, J. Peters, and S. Schaal, "Evaluation of policy gradient methods and variants on the cart-pole benchmark," in *ADPRL-2007*, 2007.
[16] H. Ulbrich, "Institute of Applied Mechanics, TU München, Germany," 2008, http://www.amm.mw.tum.de/.
[17] M. Jordan, "Attractor dynamics and parallelism in a connectionist sequential machine," *Proc. of the Eighth Annual Conference of the Cognitive Science Society*, vol. 8, pp. 531–546, 1986.
[18] H. Müller, M. Lauer, R. Hafner, S. Lange, A. Merke, and M. Riedmiller, "Making a robot learn to play soccer," *KI-2007*, 2007.