Robot Beerpong: Model-Based Learning for Shifting Targets

Fabian Wagner
Bsc. Stud. Computer Science TUDa

Felix Schmitt
Msc. Stud. Mathematics TUDa

Supervisor: Oliver Kroemer
IAS TUDa

Abstract

Defining controls for robot to achieve precise goal-directed movements can be hard when using hand crafted solutions. *Reinforcement Learning*, particularly policy-search methods provides a promising alternative which has already been successfully used for robot learning. Here the task is learned using a function that rewards desired movements and an algorithm that seeks to maximize the reward. In this paper we address the motor-task of Beerpong, that was solved using reinforcement learning in the simulation-environment SL. We extended a model-based method to the setting of bandit-type reward, showing that it is able to out perform two benchmark algorithms in terms of rollouts on the setting of a stationary cup. Also, we proofed capability of our approach to generalize to changing environment.

1 Introduction

Imagining a future use of robot companions in daily live, there is a high number of abilities needed. As one would want to give as less feedback as possible to the robot, e.g. only a appraisal after execution of the movements for a desired task, the robot has to deal with single rewards in reinforcement learning. Typical task could be throwing movements where only the only the coordinates of the landing point are given back, hammering a nail using a evaluation of the stability at the end of the task, fast reaching movements or even the right stopping of a autonomous car at a desired location. All these task can in principal be transformed to our test-problem, hence a successful algorithm for the distinct problem is expected to solve also the other.

Appearing in Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS) 2012, La Palma, Canary Islands. Volume 22 of JMLR: W&CP 22. Copyright 2012 by the authors.

Another challenge is, that the real word is subject to changes that occur during learning and might as well effect the desired task. For example the target for the throwing task could change its position. However the robot should be able to quickly adapt to the new situation and generalize the past experience. Desirable would be that even unsuccessful attempts are remembered for future use in more appropriate situation.
Our proposed One Step PILCO (OSP) algorithm uses a flexible parametrization and a sophisticated model, which is trained during learning, to deal with poor reward information and efficiently uses the gained experience in a changing environment.

1.1 Robot Environment and Task

We worked on the Barrett^{TM} robot arm with seven degrees of freedom using the open-source robot-simulator SL [1].
All learning algorithms were executed in Matlab and trajectories were sent to the simulator using a shared memory communication between Matlab and SL. We also passed an additional variable to change the cup position between the trials of a running algorithm. The given trajectory was followed by the simulator using a model-based controller with PD feedback.

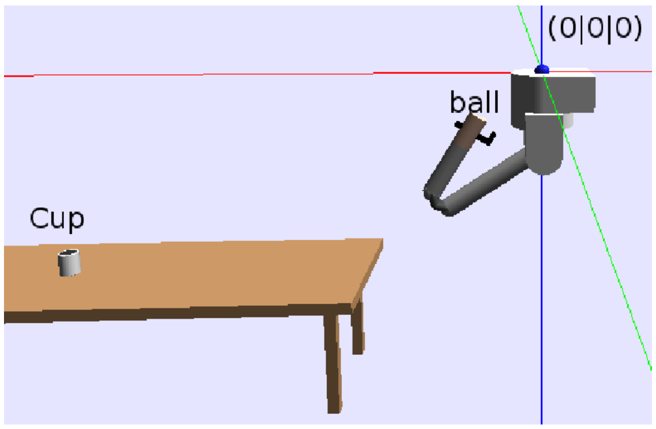

Figure 1: Beerpong environment

As shown in figure 1, the Barrett^{TM} is hanging from the ceiling, facing one table and a cup placed on the table. At the beginning of a rollout a ball is placed in ball-holder mounted on the end-effector.
The robot was given the task to throw the ball such that it bounces **once** on the table and then lands in the cup according to the game Beerpong. The cup position was initially set randomly and kept for the first two experiments. In the last experiment the cup position was changed after every 10 rollouts.

The release of the ball is not explicitly controlled but occurs when:

$$\|\ddot{x}_{\text{ball}} - \ddot{x}_{\text{rack}}\| \geq f_{\text{release}}/m_{\text{ball}},$$

where $\ddot{x}_{.}$ is the acceleration of ball and rack respectively, and the right-hand side is a release threshold. Hence, there is no release time of the ball, instead the ball lies loosely in the end-effector. The robot controls the ball only with the trajectory in joint-space until it leaves the rack.
Without loss of generality, we restricted the variations in the cup-position in the distance from the robot while keeping the height and orientation fixed. It is therefore only necessary to control the robot-movements in a y-z plane which can be done by only actuating the joints *ShoulderAbductionAdduction*, *EllBow*, *WristFlexionExtension* and keeping the rest fixed. All other positioning of the cup would be manageable by setting the shoulder-joint appropriately as shown in figure 2.

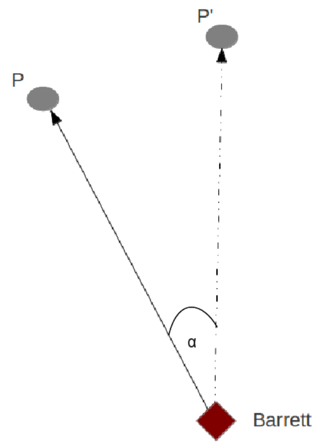

Figure 2: Variating in x-direction can be solved by changing the shoulder-joint. $\Theta(SFE) = \Theta(SFE) + \alpha$

Beerpong is a difficult skill for robots, as a high precision of the 2nd bounce is needed to get the ball into the cup. Although only three joints have to be controlled a successful execution of the task is not easy:
First, the release is not directly controllable, so that a step-wise parametrisation of the trajectory has to deal with the minor relevance of control-points past and a high importance of the **entire** movement before the release, that varies in time.
Second, rewards will have a discontinuity appearing at movements where the second bounce hits the cup's boundary from outside, which makes it a harder skill to learn.
Third, there is a variety of possible solutions to the task,resulting the reward landscape to be highly multi-modal. You can see two different ball trajectories for the same task in figure 3.

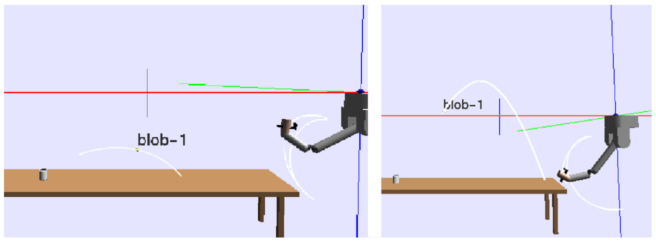

Figure 3: Two possible, yet strongly differing ball trajectories of a high-reward rollout.

1.2 Reward

As an expressive and yet simple reward-function is crucial for reinforcement-learning applications, we used a reward function inspired by Jens Kober's work on ball-in-a-cup [4] and [7]:

$$r = \exp\left(-10 \cdot \boldsymbol{\Delta}\right)$$

$$\begin{aligned}\boldsymbol{\Delta} =& \|x_{\text{2nd bounce}} - c_{bottom}\|^2 \\ &+ 0.1\|x_{\text{2nd bounce}} - c_{mid}\|^2 \\ &+ 0.1\|x_{\text{2nd bounce}} - c_{top}\|^2,\end{aligned}$$

where $x_{2ndbounce}$ is the position of the second bounce, c_{bottom} is the bottom of the cup, c_{mid} is the middle of the cup and c_{top} ist the top of the cup.

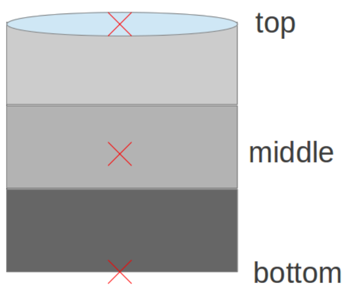

Figure 4: Location of c_{bottom}, c_{mid} and c_{top}

To calculated the reward during a rollout, the positions of all bounces on the table and their corresponding time-points are stored in SL. After 2000 ms (1000 time steps) the simulation is ended and the reward function is called. If only one bounce occurred, the reward is set to 0, otherwise we use 10 time steps before and 2 after the second bounce to calculate the nearest ball position to c_{bottom}, c_{mid} and c_{top}. The weighted minimal squared distances are summed up in $\mathbf{\Delta}$ and transformed by a radial-basis function $\exp(-10 \cdot \mathbf{\Delta})$. The motivation for this saturating reward comes from experiments in human learning [7] and has several advantages which are utilized in one of the evaluated methods.

1.3 Shifting Cup Scenario

For our last experiment we changed the learning environment to the harder task of learning to throw the ball in a cup with a varying position. The true position is always given to the robot at before each rollout, but is changed during the between rollouts. This is an example for *Transfer-Learning* where the agent (robot) has to learn how to generalize from a (half) learned policy to another one due to changes in the environment. For future use of robots in daily life these kinds of scenarios will be of high importance. Hence advanced algorithms have to be developed to solve the resulting difficulties.

2 Description of Used Methods

From the vast variety of possible algorithms for solving the Beerpong task we decided on the off-the-shelf method *Finite Difference Gradient Sampling*(FD) and the state-of-the-art method *Policy Learning by Weighting Exploration with the Returns* (POWeR) as benchmarks, as well as a Gaussian Process model-based approach One Step PILCO [8] (OSP).

2.1 Dynamic Movement Primitives

In policy search, the robot needs a suitable parametrized policy for representing throwing movements. This policy should be flexible and suitable for robot applications, e.g. also usable for kinesthetic teach-in to initialize the learning process

A well established method for parametrising motor tasks [5] are *discrete Dynamic Movement Primitives* (DMP). Instead of being explicitly specified, the trajectory $y(t), \dot{y}(t)$ of every joint is implicitly given by a *Ordinary Differential Equation* (ODE).
The first part consists of a 2nd order linear ODE, that has a point attractor g, the so called *goal state.*

$$\tau \dot{z} = \alpha_y \left(\beta_y \left(g - y\right) - z\right) + f, \quad \tau \dot{y} = z.$$

By appropriately setting α_y, β_y (default are $\alpha_y, \beta_y = [25, 25/4]$) , desired stability properties of the dynamic system, e.g. critical damping $y \to g$, can be achieved. Additionally, an external force f is added to the system, specified by

$$\begin{aligned} f(x) &= \frac{\sum_i \psi_i(x) w_i x}{\sum_i \psi_i(x)} \\ \psi_i(x) &= \exp\left(-h_i \left(x - c_i\right)^2\right) \\ \tau \dot{x} &= -\alpha_x x \quad [\text{usually } \alpha_x = 25/4]. \end{aligned}$$

This forces is steered by a 1st order ODE on the *phase* x which controls the activation of the *force basis-funcions* ψ_i that are weighted by w_i.

$$\begin{aligned} c_i &= \exp\left(-\alpha_x \left(i - 1\right) + \frac{1}{2 \cdot (N - 1)}\right) \\ h_i &= \frac{10}{13} \frac{1}{\|c_{i+1} - c_i\|^2} \end{aligned}$$

The trajectories used in the task are therefore parametrized by the weights w_i and the goal-state g only. Fixing all but g, w_i also facilitates the learning process, as the transition between time-steps depends **linearly** on the remaining open parameters which makes it also usable for imitation learning.

2.2 FD Gradient Sampling

Finite Difference Gradient Sampling is the basic method for optimization of a black-box, which is a reward function $r : \Theta \to \mathbb{R}$ that is only given by evaluation on a real system and assumed to be stochastic. Motivated by the fact that $f(\theta) - f(\theta_0) = \nabla_\theta f(\theta_0) \cdot (\theta - \theta_0) + \mathcal{O}(\|\Delta\theta\|)$ for a deterministic, differentiable function, the gradient of $\mathbb{E}(r)(.)$ in θ_0 is approximated by evaluating a set of samples $\theta_i \sim \mathcal{N}(\theta_0, \sigma^2 I)$ and computing least-squares estimation on the rewards $r(\theta_i)$. The gradient estimate G_{θ_0} is given by:

$$G_{\theta_0} = \left[\mathbf{\Delta\Theta}^T \mathbf{\Delta\Theta}\right]^{-1} \mathbf{\Delta\Theta}^T \mathbf{\Delta r}.$$

The computedG_{θ_0} can be used for stochastic steepest ascent. Although gradient estimation is relatively simple and medium scalable (one just needs enough samples), the algorithm needs in application a tuning of the step-size, which can be difficult.

2.3 POWeR

Additionally to FD Gradient Sampling we also implemented the *Policy Learning by Weighting Exploration with the Returns* algorithm, as presented in [4]. The method overcomes the problems of defining a good step-size using an *Expectation Maximization* EM approach to performing the policy updates in the learning algorithm.
In [4] it is shown, that for a (DMP) trajectory τ, $\pi_{\theta_0}(\tau)$ a given policy (in our case a sampling rule with mean θ_0 for the parameters, that results in different trajectories), $D(.\|.)$ the *Kullback-Leiber divergence* and C a positive constant

$$\log \mathbb{E}^{\pi}_{\theta}(R(\tau)) \geq -C \cdot D\left(\pi_{\theta_0}(\tau)R(\tau)\|\pi_{\theta}(\tau)\right) = L_{\theta_0}(\theta).$$

Therefore maximization of $L_{\theta_0}(\theta)$ increases the lower bound and results, as the bound is tight, in a parameter update with higher expected reward.

Using the DMPs results in trajectories of the form $\tau = (\theta + \epsilon)^T \mathbf{\Phi}(\mathbf{t})$, where $\mathbf{t} = [0, \ldots, t, \ldots, t_{\text{end}}]$ is the discretized time and ϵ additive noise used for exploration, if one would explicitly calculated the steps y_t of the ODE-solver.
In combination with a Gaussian sampling rule $\theta \mapsto \theta + \varepsilon^i$, $\forall_{i=1}^{N} \varepsilon^i \sim \mathcal{N}(0, \epsilon^2)$, N number of samples, the POWeR-algorithm simplifies to the update-rule:

$$\theta_{i+1} = \theta_i + \frac{\sum_{j=1}^{N} \varepsilon^j \cdot r(\tau^{\pi}_{\theta_i + \varepsilon^j})}{\sum_{j=1}^{N} r(\tau^{\pi}_{\theta_i + \varepsilon^j})}$$

2.4 One Step Pilco

In the context of the Transfer-Learning Scenario, it is reasonable to learn a forward model $M : \Theta \to \mathbb{R} : \theta \mapsto d_2$ form parameters θ to the resulting y-coordinate of the 2nd bounce d_2. Assume that there is already some data of parameters $\mathbf{\Theta}$ and resulting distance of the 2nd second bounce $\mathbf{d_2}$ given. As the reward $r \approx \exp\left(-\frac{1}{2b} \cdot (d_2 - t)^2\right)$, $d_2 \sim GP(\theta|\mathbf{\Theta}, \mathbf{d_2})$ can be used to guide the parameter-search. Additionally, as the state of the environment will change during learning, using information gained on a different target results in a more efficient learning process. One could also learn an inverse Model, but that is likely to lead into problems when the mapping is not one-to-one as in our **highly** multi-modal case.

The *GP*-regressor is especially suited for the task, as it gives not only a prediction mean, but also the prediction variance expressing the model uncertainty. Following [6], the *GP*-regressor for approximation of a mapping $y = f(\mathbf{x}) + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, \epsilon^2)$ by sample data $\mathbf{X}, \mathbf{y}$ at point $\mathbf{x}^\star$ is given by

$$\begin{pmatrix} \mathbf{y} \\ y^\star \end{pmatrix} \sim \mathcal{N}\left(0, \begin{pmatrix} \mathbf{K}(\mathbf{X},\mathbf{X}) + \epsilon^2_{\text{noise}}\delta_{i,j} & \mathbf{k}(\mathbf{x}^\star, \mathbf{X}) \\ \mathbf{k}(\mathbf{X}, \mathbf{x}^\star) & k(\mathbf{x}^\star, \mathbf{x}^\star) \end{pmatrix}\right)$$

with a positive definite *Kernel* $K : \mathbb{X} \times \mathbb{X} \to \mathbb{R}^+$.
Therefore, the predictive distribution
$p(y^\star|\mathbf{x}^\star, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\mu_{GP}(\mathbf{x}), \sigma^2_{GP}(\mathbf{x}))$ is defined by:

$$\begin{aligned} \mu_{GP}(\mathbf{x}) &= \mathbf{k}(\mathbf{X}, \mathbf{x}^\star) \left[\mathbf{K}(\mathbf{X},\mathbf{X}) + \epsilon^2\delta_{i,j}\right]^{-1} \mathbf{y}, \\ \sigma^2_{GP}(\mathbf{x}) &= k(\mathbf{x}^\star, \mathbf{x}^\star) \\ &\quad - \mathbf{k}(\mathbf{X}, \mathbf{x}^\star) \left[\mathbf{K}(\mathbf{X},\mathbf{X}) + \epsilon^2\delta_{i,j}\right]^{-1} \mathbf{k}(\mathbf{x}^\star, \mathbf{X}). \end{aligned}$$

We used the common *Squared-Exponential Kernel with automatic relevance detection* (SEard) given by

$$k(\theta_1, \theta_2) = s \cdot \exp\left(-\frac{1}{2}(\theta_1 - \theta_2)^T \Sigma^{-1} (\theta_1 - \theta_2)\right),$$

where s defines the scale of the basis-function and Σ is a diagonal-matrix containing the length-scales l_i for every input dimension.

Iterations

Given the current forward-model, set the next parameters to evaluate on the robot to the optimum of the stochastic *Non-Linear Program* (NLP) resulting from maximizing the expected reward $\mathbb{E}(r|\theta)$ given the GP-model:

$$\begin{aligned} \theta^\star = \arg_\theta \max\ &\mathbb{E}\left[\exp\left(-\frac{1}{2b} \cdot (d_2 - t)^2\right)\right] \\ \text{s.t.}\quad &d_2 \sim \mathcal{N}\left(\mu_{GP}(\theta), \sigma^2_{GP}(\theta)\right). \end{aligned}$$

In [7] an analytic formula for the expected reward in case of a Gaussian input d_2 is given by

$$F(\mu, \sigma^2) = ([1 + \sigma^2/b]^{-\frac{1}{2}} \cdot \exp\left(-\frac{1}{2}\frac{1}{\sigma^2 + b} \cdot (\mu - t)^2\right),$$

that turns the stochastic NLP into the deterministic:

$$\begin{aligned} &\theta^\star = \arg_\theta \max\ F\left(\mu, \sigma^2\right) \\ &\quad \text{s.t.} \\ &\quad \mu = \mathbf{k}(\mathbf{\Theta}, \theta) \left[K(\mathbf{\Theta}, \mathbf{\Theta}) + \epsilon^2\delta_{i,j}\right]^{-1} \mathbf{d_2}, \\ &\quad \sigma^2 = k(\theta, \theta) - \mathbf{k}(\mathbf{\Theta}, \theta) \left[K(\mathbf{\Theta}, \mathbf{\Theta}) + \epsilon^2\delta_{i,j}\right]^{-1} \mathbf{k}(\theta, \mathbf{\Theta}). \end{aligned}$$

Additionally [7] shows that this *saturating* reward is well-suited for reinforcement-learning, as it balances *exploitation* and *exploration* with respect to the model uncertainty and the achieved reward. One can interpret our approach as a significant easier to implement and still powerful one-step version of the *PILCO* algorithm for reinforcement-learning [8].

As the derivatives can also be calculated by means of the derivative the of kernel-function $\nabla_\theta \left[K(\theta, \mathbf{\Theta})\right]$,

$$\nabla_{\mu,\sigma^2} F\left(\mu, \sigma^2\right) = \begin{pmatrix} -F\left(\mu,\sigma^2\right)\frac{1}{\sigma^2+b}(\mu - d) \\ \frac{1}{2}F\left(\mu,\sigma^2\right)\left[\frac{1}{\sigma^2+b}\right]^2\left((\mu-d)^2 - 1\right) \end{pmatrix}$$

$$\begin{aligned}\nabla_\theta \mu(\theta) =& \nabla_\theta\left[\mathbf{k}(\theta,\mathbf{\Theta})\right]\cdot \\ & \left[\mathbf{K}(\mathbf{\Theta},\mathbf{\Theta}) + \epsilon^2\delta_{i,j}\right]^{-1}\mathbf{d_2} \\ \nabla_\theta \sigma^2(\theta) =& -2\nabla_\theta\left[\mathbf{k}(\theta,\mathbf{\Theta})\right]\cdot \\ & \left[\mathbf{K}(\mathbf{\Theta},\mathbf{\Theta}) + \epsilon^2\delta_{i,j}\right]^{-1}\mathbf{k}(\theta,\mathbf{\Theta})\end{aligned}$$

Sequential Quadratic Programming SQP can be applied to solve the NLP and the optimal parameters $\theta^\star$ can be evaluated on the robot.

Afterwards, the data is updated wit the new evaluation data $\theta \rightarrow \mathbf{\Theta}$, $d_i(\theta) \rightarrow \mathbf{d}_i$ and the forward-model is retrained. This process is iterated until the 2nd bounce reaches the desired position.

3 Evaluation

In this section, we demonstrated the ability of presented methods to solve the Beerpong-task and the application of OSP to the shifting cup scenario. Therefore we first explain implementation details of the three algorithms. Furthermore numerical results on the SL simulation environment are illustrated at two figures and finally we give an interpretation of the evaluation.

3.1 Experimental setup

We now describe the setting of the hyperparameters of the algorithms and especially show several implementation details of our presented OSP method.

3.1.1 FD Gradient Sampling

In our implementation of the *FD Gradient Sampling* we used one independent DMP for each of the three active joints and 4 basis-functions, for reasons of comparison to OSP, to generate the movements. The number of basis-functions did not play a big role for FD nor for POWeR, as computation and convergence scaled with the increase of samples due to higher dimension.
The solution of the ODE defined by the DMPs was done with a explicit *2/3 Bogacki-Shampine Runge-Kutta integrator* which is the standard in both Matlab/Octave. Initialization was for all methods $\theta_0 = [1.25 + 0.2\epsilon, 1.2 + 0.2\epsilon, 0.4\epsilon, \epsilon_{1:3}] \quad \epsilon \sim \mathcal{U}([-0.5, 0.5])$.
As mentioned earlier it was necessary to get a good step-size rule and our best results were achieved using $\sigma = \min\left(30(1 - r(i))\sigma_0, 10\sigma_0\right)$ where $\sigma_0 = 0.01$ for **20 parameter samples** $\sim \mathcal{N}(\theta, \sigma^2)$ and step-size for the normalized gradient.

3.1.2 POWeR

We used the POWeR-algorithm in combination with the DMP trajectory generator as proposed in [4], in the same setting as FD for matters of comparability. The policies where set to $\tau[\pi(\theta)]$, $\pi(\theta) \sim \mathcal{N}(\theta, [(1-R)/40]^2)$.
We sampled **10 parameters** in each iteration and additionally followed [4] and reused the best 5% of rollouts in each update.

3.1.3 OSP

Although OSP seems very sound from theoretical view, it is the most complex of the methods to implement, but still considerably more straightforward then full *Pilco*. First, it is important to set the hyper-parameters s, l_i, ϵ to a appropriate size. Hence we used [3] to maximize the marginal likelihood [6] on a set of data collected in rollouts. In order to use only one sample to initialize the OSP, the ϵ had to be scaled down, to not wash out data from real evaluation in early stages of the algorithm. During iteration ϵ was slowly increased with the growth of data.
Given the small amount of data, the dimension of Θ had to be reduced (initially 16) using only 4 basis functions. In this manner, the forward model had to be learned in only a 15-dimensional space

One natural choice for initializing the policy search is to search all collected data for the best θ_0 and apply descent from there. However, this approach usually ended in getting stuck near the initialisation, as the low noise made the variance of the GP-model increases too strong away from the initial point that SQP did not produce a expressive solution.
First, we added the Variance of the reward $k^2 \cdot \mathbb{V}\left[\exp\left(-\frac{1}{2b}\cdot(d_2 - t)^2\right)\right]$ to the objective as proposed in [7] to increase the exploration by rewarding rollouts in uncertain regions.
Second, we initialized $\theta_0 + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, 0.035 \cdot I)$ and discarded NLP solutions when to distant for reasons of safety. These parameters were not executed but incorporated in the model setting $d_2 = 0$ to prevent the robot from damaging itself and trying again.

3.2 Results

For evaluation we executed 20 trials, each with a random cup position $\sim \mathcal{U}([2, 3.7])$. FD and POWeR were run for 30 parameter-updates and OSP for 50 iterations. As the different methods use a varying number of rollouts per parameter-update, the total number differs as it can be seen in figure 5.

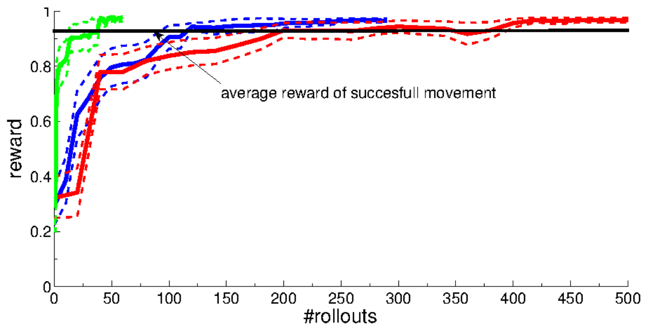

Figure 5: Evaluation on static cup scenario: red FD, blue POWeR and green OSP.
Thick line $\mathbb{E}(r)$, thin lines normal error ν, black line average reward for successful throw.

The black line at 0.93 indicates the average reward over the different cup positions, a successful throwing movement achieves.

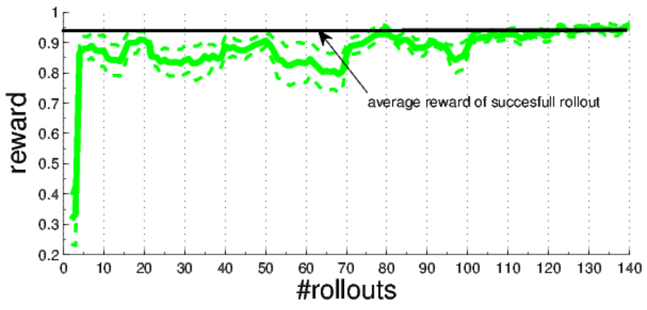

Figure 6: Evaluation of OSP on shifting cup scenario changing the cup postion every 10 rollouts

For the shifting cup scenario figure 6, the cup was moved to a random cup position $\sim \mathcal{U}([2, 3.7])$ every 10 iterations of the OSP algorithm.

3.3 Discussion

As a rather unspecific FD needed the most rollouts and had also the highest variance in the rewards.

POWeR performed significantly better and needed half of the rollouts to reach a successful policy and was the fastest method in terms of computation. However, it first seemed as if the discontinuity in the reward function caused problems when reusing old good rollouts, as roll outs which hit the cup's outside where not discarded from the best 5% and stop the algorithm from reaching higher rewards. But finally this issue was solved by the use of a more elaborate reuse-rule incorporating the current reward.

OSP, although needing by far the most computation time, clearly outperformed the other methods, needing on average only 40 rollouts ($\frac{1}{3}$ of POWeR and less than $\frac{1}{10}$ of FD) to achieve a successful movement. The reason, can be found in the fact that the method uses an explicit forward model to decide on parameters to evaluated. The stochastic nature of the GP-regressor, combined with the functional, consisting of expected reward and variance, automatically produces efficient exploration, steered by the present model uncertainty, as reported by [7].
In the scenario of the shifting cup, figure 6 shows, that OSP is capable of quickly generalizing the throwing movements. Directly after moving the cup, the reward breaks in for the first two to three iterations and recovers after a few more iterations in the first half of the experiment. In the second half the model is good enough to directly perform a high reward movement in the new situation. Finally in the end OSP produces successful rollouts even if the cup position changes.

4 Conclusion and further work

In this paper we presented the motor-task of Beerpong and explained why it is a relevant and interesting problem for reinforcement learning. Furthermore we showed that the task is solvable using a DMP parametrisation and the presented three methods. We also extended the previous work of [8] to the special single, bandit-type reward setting in Beerpong which resulted in the proposed OSP-algorithm, that we derived directly from a GP-forward-model-based approach and the saturating reward function. Our simulated experiments showed that OSP can learn faster than the two benchmark algorithms FD and POWeR.
Finally we changed the Beerpong-task to the more difficult shifting cup scenario and successfully evaluated OSP.

In future work we are planning to investigate how to simultaneously track the GP-hyperparameters during the learning process iterations. Therefore, it is also worth considering the use of a prior on the hyperparameters.
Additionally, we will perform a comparison to *Cost regularized Kernel Regression* [2], a kernelized extension of POWeR, on similar generalization tasks.

Acknowledgements

Thanks to Chris for his help on SL and the matlab interface and to the barber who took care of the marks SL left in Felix hair.

References

[1] S. Schaal. *The SL Simulation and Real-Time Control Software Package* Computational Learning and Motor Control Laboratory, Computer Science and Neuroscience, University of Southern California, Los Angeles

[2] J. Kober, E. Oztop and J. Peters. Reinforcement Learning to adjust Robot Movements to New Situations. Proceedings of Robotics: Science and Systems (R:SS), 2010

[3] C. E. Rasmussen and H. Nickisch. *GAUSSIAN PROCESS REGRESSION AND CLASSIFICATION Toolbox version 3.2 for GNU Octave 3.2.x and Matlab 7.x.* from `http://www.gaussianprocess.org/.../code/`, 2005-2013

[4] J. Kober and J. Peters. Policy Search for Motor Primitives in Robotics. Advances in Neural Information Processing Systems 22 , The MIT Press, Cambridge, MA, USA, NIPS 2007

[5] S. Schaal, P. Mohajerian, A. Ijspeert. Dynamic Systems versus Optimal Control - A unifying view. Progress in Brain Research, Vol. 165, pages 425-445. Elsevier B.V. 2007

[6] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, MA, USA, 2006.

[7] M. P. Deisenroth. *Efficient Reinforcement Learning using Gaussian Processes* PhD thesis, in Uwe D. Hanebeck (eds.), 9, KIT Scientific Publishing.

[8] M. P. Deisenroth and C. E. Rasmussen PILCO: A Model-Based and Data-Efficient Approach to Policy Search. International Conference on Machine Learning (ICML 2011)