Learning to bounce a ball with a robotic arm

Eric Wolter
TU Darmstadt

Thorsten Baark
TU Darmstadt

Abstract

Bouncing a ball is a fun and challenging task for humans. It requires fine and complex motor controls and thus is an interesting problem for robot learning. We investigate how to find a policy that is able to keep a ball in the air using a paddle. To solve the bounce ball task we use Relative Entropy Policy Search (REPS) to learn the parameters of Rhythmic Dynamical Movement Primitives (RDMP). We evaluate the policies learned for different configurations. The results show that REPS is successful at solving configurations in which the ball initialization is point-wise, but does not converge to a good optimum when the ball is initialized from a distribution.

1 Introduction

The task of bouncing a free-floating ball off a paddle is both a fun and challenging game for humans. Everybody who has played table tennis has at one point attempted the bounce ball task. The complex motor control skills necessary for completing this task make it an interesting and challenging problem for robot learning.

The bounce ball task can be described as a simple scenario where a paddle is used to hit a small ball. The goal is to not let the ball fall down. To achieve the goal the ball has to be hit carefully so that it doesn't deviates too far from the center position as it otherwise gets harder and harder to reach the ball without moving the whole body.

The goal of robot learning is to find a task-specific control policy that is capable of performing the required movement for the task. Learning the policies by trial and error using reinforcement learning algorithms "allows the usage [...] of pre-structured policies, the [...] integration of a teacher's presentations as well as fast online learning" [3]. Policy search methods thus have become an accepted alternative to other reinforcement learning algorithms [3].

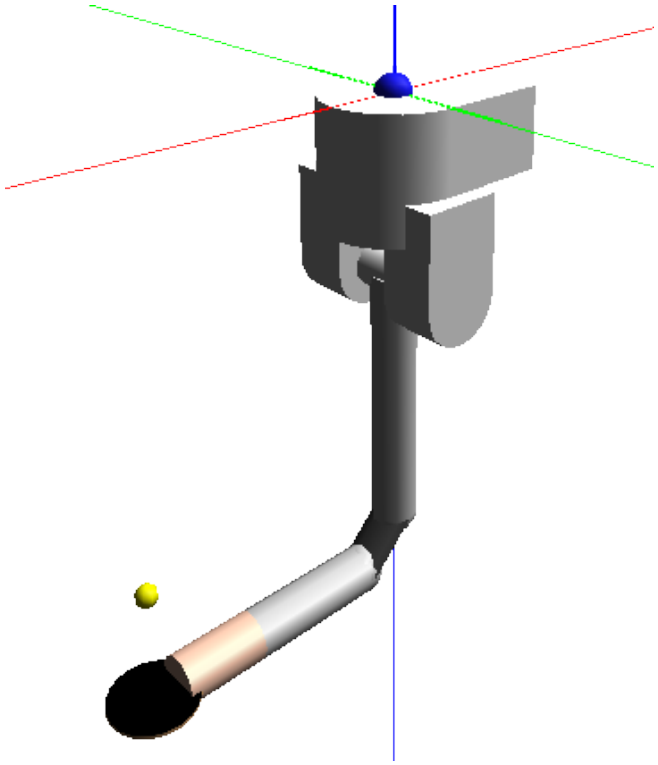

Figure 1: The robotic arm used to learn the bounce ball task.

In this paper we focus on Relative Entropy Policy Search (REPS) to train a simulated robot to perform the bounce ball task. REPS already exhibits good and robust results for a number of different problems [4][1], such as playing tetherball. Efficiently learning the control policy for the bounce ball task is complex, as the state-action space is high dimensional, so Dynamical Movement Primitives (DMP) are used to take advantage of parameterized policies that can achieve a complete movement behavior [7]. We observe that the required paddle movement is cyclic and thus we rely on Rhythmic Dynamical Movement Primitives (RDMP) to model the path of the paddle. RDMP requires a fixed number of parameters to control the desired

movement of the paddle. Each joint is controlled by its own RDMP and therefore has its own set of parameters. We use REPS to learn a policy to compute their values.

To experimentally validate the performance of REPS and RDMP we use a simulated robotic arm. The robotic arm, shown in Figure 1 consists of 7 joints, which enable it to closely model the movement of a human arm.

In order to better understand the performance of REPS we test and evaluate a number of different configurations. Each configurations varies by the number of joints used and the initial ball movement. We ensure that the initial ball velocities remain in a reasonable range so that the ball is reachable.

Experimental results show that REPS is capable of solving the tasks, where the initial ball state is fixed. For configurations where the initial ball velocity is a distribution, REPS fails to converge to a good optimum. It still manages to find reasonable results, however with large variance. Similar results in [1] suggest that our model has a multimodal reward and therefore can not converge.

The remainder of this paper is structured as follows. In Section 2 we describe the theory behind RDMP and REPS. In Section 3 we describe the experimental setup and in Section 4 the evaluation of the results. We conclude in Section 5 with giving an outlook on possible future improvements and extensions.

2 Background

The goal of this paper is to train a robot arm to repeatedly hit a ball to keep it in the air. We model this task as a Markov Decision Process (MDP) [6]. A MDP consist of possible states s_t, possible actions a, a reward function $\mathcal{R}_s^a$ for the state and chosen action, as well as a transition function $f(s, a)$. This transition function takes a given action and moves the MDP from state s_t into a new state s_{t+1}, i.e. $s_{t+1} = f(s_t, a)$. In the bounce ball task we choose the states to be the current ball position and velocity and the actions to be the parameters used to describe the motion for hitting the ball. The transition function f is the simulated behavior of the ball and robot arm. The reward function, as described in Section 3, is based on the amount of times the ball is hit above a certain threshold.

To describe the movement of the robot arm we use Dynamic Movement Primitives (DMP), as it is a parametrized approach for modeling motions. However the bounce ball task is intrinsically cyclic, in that we want to repeatedly hit a ball with a paddle and thereby keep it up in the air. To model this periodic behavior we use Rhythmic Dynamic Movement Primitives (RDMP), a periodic version of DMPs as described in [2] and [6]. It allows us to model the necessary periodic movement of a robot arm by learning its parameters.

In order to learn the parameters of the RDMPs we use the method of Relative Entropy Policy Search (REPS), as previous work in [4] and [1] showed its successful learning of multiple tasks, including playing tetherball.

The two techniques are combined by using the actions, that are sampled from the policy estimated by REPS as the parameters for the RDMPs that model the paddle motion.

2.1 Dynamical Movement Primitives

Dynamic Movement Primitives (DMP) [6] represent a point attractive system, i.e. they try to close the gap between the current and a goal position. Their instantiation consist of two sets of equations, the canonical system and the transformation system.

In the case of a discrete acceleration DMP, as described in [6], the canonical system is expressed as

$$\tau \dot{x} = v \text{ and } \tau \dot{v} = \alpha_v(\beta_v(g - y) - v), \tag{1}$$

where g is the goal position, α_v and β_v are time constants, τ is a time scaling factor, y is the desired position, x is a phase variable and v a phase velocity. x and v are used by the transformation system instead of an explicit time representation as it allows for the manipulation of the time evolution.

The transformation system on the other hand is

$$\tau \dot{z} = \alpha_z(\beta_z(r - y) - z) + f, \tag{2}$$
$$\tau \dot{y} = z, \tag{3}$$
$$\tau \dot{r} = \alpha_g(g - r), \tag{4}$$

where α_z and β_z are time constants, r is introduced to allow for a first order filter to ensure a continuous acceleration profile and $y, z, \dot{z}$ are the desired position, velocity and acceleration respectively. The function f works as a non-linear approximator and is based on the phase variable and velocity given by the canonical system. In the discrete acceleration DMP it is defined as

$$f(x, v, g) = \frac{\sum_{i=1}^{N} \Phi_i w_i v_i}{\sum_{i=1}^{N} \Phi_i}, \tag{5}$$

$$\text{where } \Phi_i = \exp\left(-h_i \left(\frac{x}{g} - c_i\right)^2\right). \tag{6}$$

This formulation allows us to learn a motion by estimating the weights w_i.

2.2 Rhythmic Dynamical Movement Primitives

Although DMPs work well as point attractor systems, they are not able to capture cyclic motions as their phase variables are strictly monotonic [6]. However they can be adapted to Rhythmic Dynamical Movement Primitives (RDMP) [6, 2] by changing $\tau\dot{r}$ to

$$\tau\dot{r} = \alpha_r(A - r), \tau\dot{\phi} = 1 \tag{7}$$

where r is now the amplitude of an oscillator, A is the desired amplitude and $\dot{\phi}$ is the phase. Further the non-linear approximator f is defined as

$$f(r, \phi) = \frac{\sum_{i=1}^{N} \Phi_i \mathbf{w_i} \mathbf{v}}{\sum_{i=1}^{N} \Phi_i}, \tag{8}$$

$$\text{where } \mathbf{v} = [r\cos\phi, r\sin\phi]^T \tag{9}$$

$$\text{and } \Phi_i = \exp\left(-h_i \left(mod(\phi, 2\pi) - c_i\right)^2\right). \tag{10}$$

The definition of v in f indicates the periodic nature of RDMP. In RDMP as well as in the basic DMP the weights w_i determine the trajectory that the DMP follows and therefore they need to be estimated to solve the task of bouncing a ball above a paddle.

2.3 Relative Entropy Policy Search

Relative Entropy Policy Search (REPS) is a method for finding policies for Markov Decision Processes (MDP), which given a current state s compute an action a bringing the MDP into state s'.

REPS, as shown in Algorithm 1, aims to find the optimal policy that maximizes the expected return, while also bounding the loss of information during the search. Although it can be applied to continuous actions and states, [4] is focused on discrete state machines. Therefore we also use some of the notation given by [1] as it contends directly with finding continuous policies. However, as we are interested in the learning performance of the basic REPS algorithm we do not adopt the main improvement of [1] which extends REPS to Hierarchical Relative Entropy Policy Search (HiREPS) to deal with multimodal reward functions.

The maximization of the expected return is performed based on the observed states s, actions a, rewards $\mathcal{R}_s^a$ and the relative entropy between the observed data distribution $q(s, a)$ and the data distribution $p^\pi(s, a) = \mu^\pi(s)\pi(a|s)$. The relative entropy is used to bound the information loss. Hence [4] arrives at the maximization problem of

Algorithm 1: REPS (adapted from [1])

Input : Information loss tolerance ϵ
Initialize: π as a gaussian
while *not converged* **do**
  Sample: collect samples from the policy
    $\{s_i \sim a,\ a_i \sim \pi(a|s_i), \mathcal{R}_s^a\}\ i \in \{1, ..., N\}$
  Minimize the dual function:
    $[\theta, \eta] = \arg\min_{[\theta,\eta]} g(\theta, \eta)$
  Policy update:
    Calculate weighting for each sample
    $w_{s,a} = \exp\left(\frac{\mathcal{R}_s^a - \theta^T\phi}{\eta}\right)$
    Estimate π by weighted ML estimates
end
Output : Policy $\pi(a|s)$

$$\max J(\pi) = \sum_{s,a} \mu^\pi(s)\pi(a|s)\mathcal{R}_s^a, \tag{11}$$

$$s.t.\ \epsilon \geq \sum_{s,a} \mu^\pi(s)\pi(a|s)\log\frac{\mu^\pi(s)\pi(a|s)}{q(s,a)}, \tag{12}$$

$$\sum_{s'} \mu^\pi(s')\phi_{s'} = \sum_{s,a,s'} \mu^\pi(s)\pi(a|s)\mathcal{P}_{s,s'}^a\phi_{s'}, \tag{13}$$

$$1 = \sum_{s,a} \mu^\pi(s)\pi(a|s) \tag{14}$$

where $\mathcal{R}_s^a$ is the reward achieved by action a in state s, $\mathcal{P}_{s,s'}^a$ the probability of a transition from state s into s' given the action a, ϕ_s represents features of a state and ϵ limits the loss of information.

In our implementation the equation is solved by optimizing the dual function g given by

$$g(\theta, \eta) = \epsilon\eta + \theta^T\phi_{\mathbf{s}} + \eta\log\left(Z_{s,a}\right), \tag{15}$$

$$\text{where } Z_{s,a} = \sum_{s,a} q(s,a)\exp\left(\frac{\mathcal{R}_s^a}{\eta}\right) \tag{16}$$

which is adapted from [1] instead of [4], as it gives the same results but is more intuitive when continuous states are used.

The arguments θ and η, that result from maximizing g, are then used to estimate a new policy by weighting each action and state pair with

$$w_{s,a} = \exp\left(\frac{\mathcal{R}_s^a - \theta^T\phi}{\eta}\right). \tag{17}$$

The process of maximization and policy estimation is subsequently repeated until the policy converges.

3 Experimental Setup

In order to evaluate the performance of the proposed methods we use the SL Simulation and Real-Time

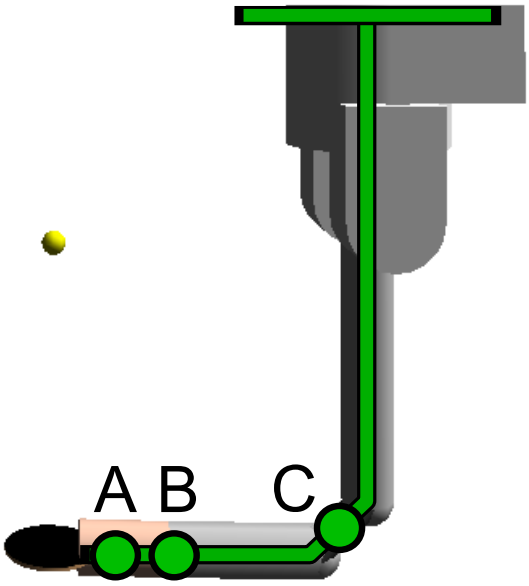

Figure 2: The three joints used during learning for a total of three degrees of freedom. Joint C is the elbow and allows the forearm to move up and down. Joint A and B enable the pitch and roll of the paddle.

Control Software Package [5]. It provides the framework in which the robot arm and the bounce ball task have been implemented.

The robot arm we use in our setup consists of 7 joints of which we only use three, as is illustrated in Figure 2. Joint C resembles the elbow and allows the forearm to move up and down. Joint A and B model a human hand and enable pitching and rolling of the paddle attached at the end-effector.

The initial position of the ball, i.e. $[0, -0.545, -0.6]$, is directly above the center of the paddle. The paddle itself is positioned at $z = -0.9$. The robot is rewarded for each hit which lets the ball fly above a certain threshold $t = -0.7$, as listed in Algorithm 2.

Algorithm 2: The reward function

if $ball\dot{Z}_{old} < 0$ ***and*** $ball\dot{Z}_{current} > 0$ **then**
    hasHit = TRUE
end
if $hasHit = TRUE$ ***and*** $ballZ_{current} > t$ **then**
    reward += 1
end

The reward is both intuitive to understand and showed reliable results during our experimentations. It prevents a number of naive solutions like catching the ball or following the ball to the floor. In addition to that an extra reward is given if the ball is still above the paddle at the end of the simulation period, as otherwise solutions which drop the ball right before the end can achieve the same good reward. In order to always start with a reasonable initial action distribution we use a sample from the previously learned policy as the initialization vector.

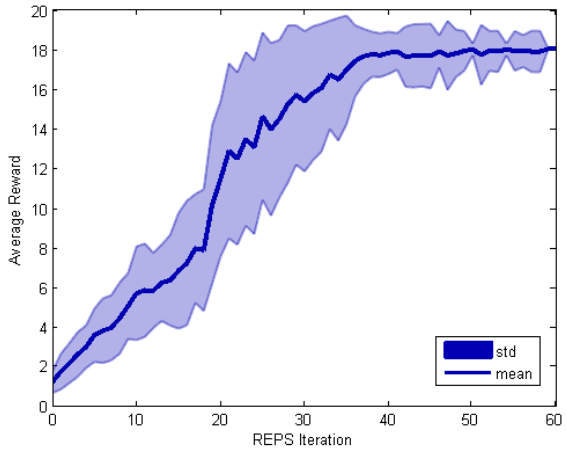

Figure 3: The figure shows the average reward on the bounce ball task when only using the elbow joint and point-wise ball initialization. The learning converges and achieves the optimum of 18.

4 Results

We look at a number of increasingly difficult configurations of the task. For each configuration the number of joints used and the initial ball state is varied. For the simulation period used during the experimentation the best reward that can be achieved is 18, it indicates that the ball stays above the paddle for the entire period.

4.1 Point-wise ball initialization and one joint

The first configuration places the ball directly above the center of the paddle with no initial velocity or acceleration. We only learn the RDMP for the elbow joint using five weights to modulate the movement. The graph in Figure 3 shows the mean and standard deviation of the reward for each iteration REPS is learning the policy. The reward reaches the optimum of 18 after about 40 iterations and then continues to converge. The final policy is able to successfully hold the ball above the paddle for the entire simulation period.

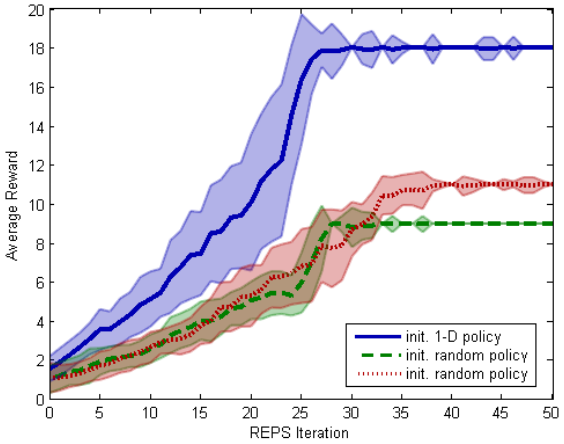

Figure 4: The average reward on the bounce ball task using the elbow joint and joints for the pitch and roll of the hand and a point-wise ball initialization. If the final policy found in configuration 4.1 is used as the initial policy the optimum of 18 is reached. For the random initial policy the solution converges prematurly without finding a good optimal solution.

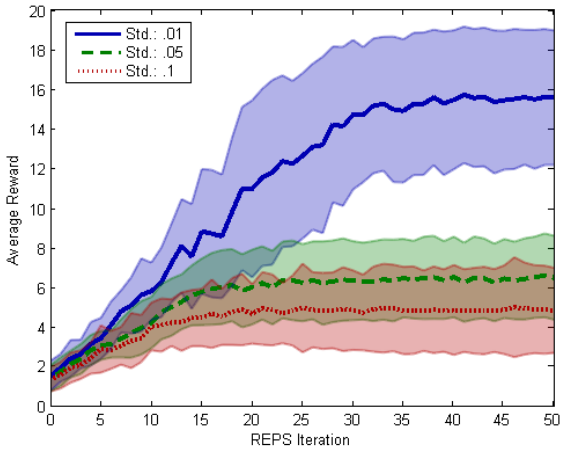

Figure 5: Bounce ball with elbow joint and joints for the pitch and roll of the hand. The initial ball state is varied in the x velocity using the three different standard deviations of .01, .05 and .1. REPS finds good results for the smallest deviation but degrades for the larger deviations. For all cases the variance in the rewards is still large in the end.

4.2 Point-wise ball initialization and three joints

The second configuration builds on top of the first scenario. However, instead of only utilizing the elbow two extra joints modeling the pitch and roll of the paddle are now also used. The RDMP for each joint individual is modeled with five weights each, so the total action base has now fifteen dimensions. We tested both random initial means for the actions as well as a sample from the final policy found in configuration 4.1. The initial action variance however is the same for all means. Figure 4 shows the resulting mean and standard deviation of the reward for two runs using an initially random policy and one run using the previously computed 1-D policy with increased variance. The performance for this larger action space is still good. However, the optimum of 18 is only reached when using the improved mean. The random initial means fail to get to the optimum solution and instead converge earlier. The different points of convergence suggest that the bounce ball task might be multimodal.

4.3 Varying ball velocity in x and three joints

For the third scenario the utilized joints and the number of weights for RDMP stay the same. However, the ball initialization is now no longer constant but has a small varying velocity along the x-dimension, which is perpendicular to the forearm of the robot, i.e. to the left and right of the paddle. We test three different standard deviations for the initial ball velocity to see how good REPS can deal with the perturbance. The largest deviation is chosen in such a way that the ball is almost always hitting the paddle. This ensures that the learning process is not hindered by these outliers. Figure 5 shows the trend of the mean reward for the three different standard deviations. For the smallest deviation of .01 the solution is still good. However, REPS fails to find the optimal solution of 18 and still has a large variance in the end. For larger deviations in the initial velocity the results degrade even more. The final solutions are only able to hit the ball a couple of times before it falls down. In addition the variance is also still very large. We hypothesize that this behavior is the result of REPS trying to average over multiple modes similar to the tetherball task in [1].

5 Conclusion

The bounce ball task provides an interesting challenge for robot learning, because of its complex motor controls and high dimensional action space. We implemented REPS for learning the parameters of RDMP which we used to model the movement of the paddle.

In order to evaluate the performance we used a simulated robotic arm. We defined a number of increasingly difficult configurations, which varied both the number of used joints as well as the initial ball state.

Experimental results show that for configurations

where the ball initialization is point-wise the simulation performed well and converged quickly, enabling the robot bounce the ball for the entire simulation period. However, for cases where the initial ball velocity is varied the solution fails to converge to an optimum and the variance remains large.

We hypothesize that the poor performance of REPS for the more difficult configurations is due to their multimodal nature. This is supported by the experiments conducted in [1]. They specifically address the issue of multimodal tasks with their extension HiREPS. Applying HiREPS to the bounce ball task would thus be a logical next step.

References

[1] C. Daniel, G. Neumann, and J. Peters. Hierarchical relative entropy policy search. In *International Conference on Artificial Intelligence and Statistics*, 2012.

[2] A. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. *Advances in neural information processing systems*, 15:1523–1530, 2002.

[3] J. Kober and J. Peters. Policy search for motor primitives in robotics. *Machine Learning*, 84(1):171–203, 2011.

[4] J. Peters, K. Mülling, and Y. Altun. Relative entropy policy search. In *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI), Physically Grounded AI Track*, 2010.

[5] S. Schaal. The sl simulation and real-time control software package. *University of Southern California*, 2009.

[6] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Control, planning, learning, and imitation with dynamic movement primitives. In *Proceedings of the Workshop on Bilateral Paradigms on Humans and Humanoids, IEEE 2003 International Conference on Intelligent RObots and Systems (IROS)*, pages 27–31, 2003.

[7] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Learning movement primitives. *Robotics Research*, pages 561–572, 2005.