Reinforcement Learning vs Human Programming in Tetherball Robot Games

Simone Parisi[1], Hany Abdulsamad[1], Alexandros Paraschos[1], Christian Daniel[1], Jan Peters[1,2]

***Abstract*— Reinforcement learning of motor skills is an important challenge in order to endow robots with the ability to learn a wide range of skills and solve complex tasks. However, comparing reinforcement learning against human programming is not straightforward. In this paper, we create a motor learning framework consisting of state-of-the-art components in motor skill learning and compare it to a manually designed program on the task of robot tetherball. We use dynamical motor primitives for representing the robot's trajectories and relative entropy policy search to train the motor framework and improve its behavior by trial and error. These algorithmic components allow for high-quality skill learning while the experimental setup enables an accurate evaluation of our framework as robot players can compete against each other. In the complex game of robot tetherball, we show that our learning approach outperforms and wins a match against a high quality hand-crafted system.**

I. Introduction

Efficient acquisition of complex motor skills is crucial for robotics. In recent years, robot learning of dynamic motor tasks has been shown the ability to acquire a variety of skills, ranging from the game ball-in-a-cup [1], archery [2], the peg-in-hole task [3], the tetherball hitting game [4] to walking [5] and jumping [6]. One of the main breakthroughs that led to these results has been the introduction of motor primitives [7]. Motor primitives offer a compact representation of basic movements such as grasping and hitting, and are commonly represented by dynamical systems. However, although imitation learning allows the robot to learn simple skills [8], intelligent algorithms are necessary to adapt primitives to different tasks and to learn new optimized policies [4]. In the field of real robot learning, Reinforcement Learning (RL) and especially Policy Search (PS) methods have become increasingly important [9]. PS is a general approach to learn policies using sampled trajectories, also called rollouts. The search takes place in a subset of the policy space using the experienced reward from the rollouts as quality assessment for the policy. While the locality of the search might be viewed as undesirable in many simulated tasks, it is an important feature in real robot learning. Being able to search for solutions only around initial demonstrations does not only speed up learning but also reduces the risk that the resulting trajectories will be dangerous to the robot or its environment. Further advantages of PS methods are their ability to cope with continuous state-action spaces as well as their sample efficiency. Where learning in continuous state-action spaces is a fundamental requirement of real robot learning, sample efficiency on real systems is not only desirable to reduce learning time but is essential as many currently available platforms require intensive maintenance after many rollouts.

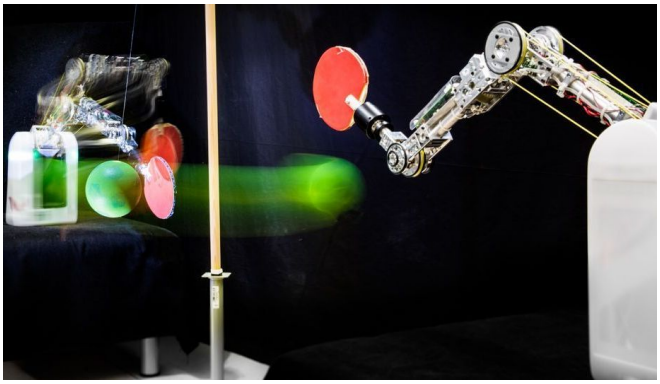


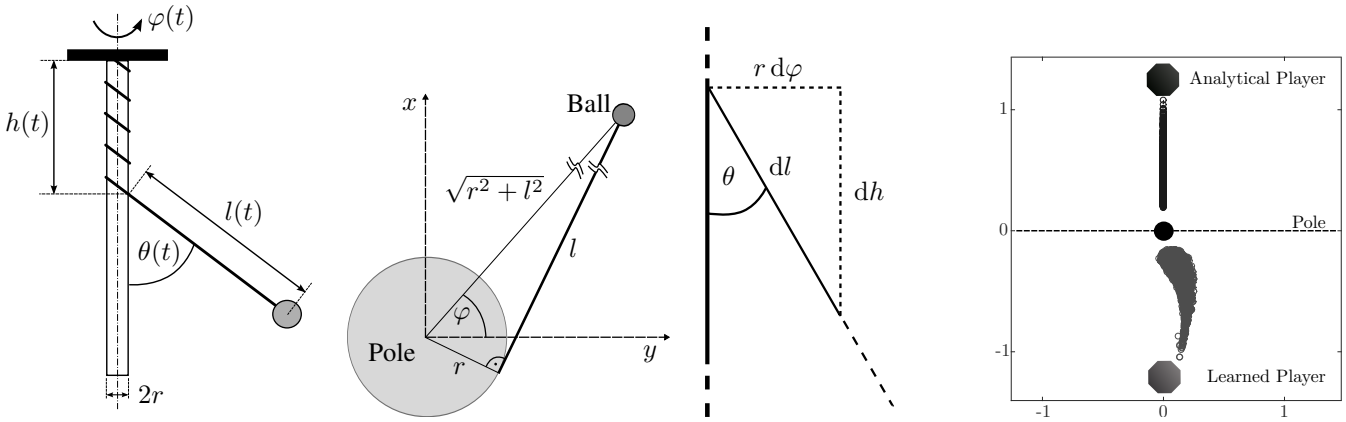
Fig. 1: In robot tetherball, a pole with a ball hanging from the top is placed between two robotic arms. The goal of the robots is to repeatedly hit the ball without giving the opponent the chance to unwind it.

While there has been a variety of publications on the achievements of real robot learning, no systematic comparison of learned skills against manually programmed solutions has been performed so far. Conducting such a systematic evaluation is a challenging proposition for several reasons. First, it is non-trivial to find a suitable measure to compare hand-crafted and learned policies. By design all learned policies try to optimize a given reward function. However, using this reward function as comparison measure would introduce a bias, as there is no guarantee that the manual program maximizes the same reward function. Secondly, manually designing a proficient program to successfully solve a robotic task requires deep insights into both the platform and the problem itself. In this paper, we propose a task setup that addresses these problems and allows us to present a systematic and thorough evaluation of state-of-the-art approaches in real robot learning.

The remainder of the paper is structured as follows. We start by describing the robot and the tetherball setup in Section II and continue with a technical analysis of the task required to design the manual program in Section III. In Section IV we introduce the components for learning to play tetherball by trial and error. Subsequently, in Section V we give insights into different issues in RL and compare the quality of the motor skills of different robot players. Finally, in Section VI we discuss the results of this paper and propose possible avenues of investigation for future work.

[1]Autonomous Systems Labs, Technical University of Darmstadt, 64289 Darmstadt, Germany {last_name}@ias.tu-darmstadt.de

[2]Max Planck Institute for Intelligent Systems, 72076 Tübingen, Germany



(a) Tetherball Model (b) Planar View (c) Differential View (d) Distribution of Hitting Points

Fig. 2: Schematics of the mechanical tetherball model. a) State representation in spherical space with four degrees of freedom. b) Simplification of the $\{x, y\}$ coordinates by ignoring the pole radius r. c) The infinitesimal change of the states for formulating the nonholonomic constraints. d) The distribution of the hitting points for both players. The analytical player hits the ball in the fixed plane defined by the pole and origin of the robot. The learned player does not have any constraint and can freely choose the interception point.

II. Robot Tetherball

To address the problem of finding a fair evaluation measure, we propose to use a task that is based on the game of tetherball. The advantage of using a game based task is that there is a pre-defined success measure, i.e., the game score. As we will see, this success measure cannot be used directly for either the learned player or the manually designed program, but is well-suited to evaluate their relative performance. A robot tetherball task has been presented previously by Daniel et. al [10]. However we propose the full two-player setup, where Daniel et. al only presented a strongly simplified single player version of the task. Apart from being more challenging, the two player setup offers important new possibilities of evaluating a learned player directly against the manually designed program.

In our setup, shown in Fig. 1, we use two BioRob robots with six degrees of freedom each [11]. The BioRobs are cable driven lightweight robots, that achieve highly dynamic behavior due to the integration of springs between the motors and the cables which drive the joints. This highly dynamic behavior, however, makes it hard to provide a inverse dynamics model for the robot.

In the two player tetherball task, the robots are set up facing each other and a pole is centered between them. A ball is attached to the top of the pole using a rope. The goal of the robots is to hit the ball such that the rope winds around the pole as often as possible in one direction without giving the opponent the chance to unwind it. To achieve this task, one robot tries to hit the ball clockwise, while the opponent tries to hit it counter-clockwise.

Formally, we denote the state of the environment by $\boldsymbol{x} = \{\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{b}, \dot{\boldsymbol{b}}, h\}$, where $\boldsymbol{q}, \dot{\boldsymbol{q}} \in \mathbb{R}^6$ are the joint positions and velocities, $\boldsymbol{b}, \dot{\boldsymbol{b}} \in \mathbb{R}^3$ are the ball position and velocity in Cartesian coordinates, h is the height of the pivot point along the pole. The control actions $\boldsymbol{u} \in \mathbb{R}^6$ are torques generated by a low-level controller $\boldsymbol{u} = f_{\text{PD}}(\boldsymbol{x}, \boldsymbol{x}_{\text{des}})$. The low-level controller chooses actions $\boldsymbol{u}$ such that the robot follows a desired trajectory $\boldsymbol{\tau}_{\text{des}} = \{\boldsymbol{x}_{\text{des},i}\}_{i=1:H}$, where H is the length of the desired trajectory. While the joint velocities $\dot{\boldsymbol{q}}$ can be controlled directly, the ball positions $\boldsymbol{b}$ and the pivot point h are only controlled indirectly. Given this setup, we formulate both an analytical player and a learned player capable to play robot tetherball against each other.

III. The Analytical Player

In this section, we dissect the tetherball task and discuss how the problem can be described mathematically in order to derive an analytical player. This hand-crafted player uses a mechanical model of tetherball to predict the ball trajectory and the resulting interception point. Subsequently, a stroke movement represented by a spline function is generated and followed by the low-level controller $f_{\text{PD}}(\boldsymbol{x}, \boldsymbol{x}_{\text{des}})$.

A. Mathematical Trajectory Prediction

Mathematically describing tetherball requires a complex mechanical model [12]. Even after disregarding effects like friction, air drag and changes in string tension, the underlying equations are still highly nonlinear. As shown in Fig. 2a, we can describe the state of the model with four degrees of freedom when represented in spherical space. Here, $\{\theta, \varphi, l\}$ are the spherical coordinates while h is the vertical translation of the pivot point along the pole. These four coordinates fully determine the Cartesian position of the ball according to the equations

$$x \simeq l \sin\theta \cos\varphi, \quad y \simeq l \sin\theta \sin\varphi, \quad z = l \cos\theta + h. \tag{1}$$

As shown in Fig. 2b, for $\{x, y\}$ coordinates we ignore the pole radius r when compared to the length of the string l. In addition to these algebraic equations, the system is constrained by two nonholonomic, i.e., not integrable, velocity conditions for $\{\dot{l}, \dot{h}\}$, as shown in Fig. 2c. These

constraints reflect the behavior of the string while it (un-)winds around the pole and are given by

$$\dot{l} = \frac{-r\dot{\varphi}}{\sin\theta}, \quad \dot{h} = \frac{r\dot{\varphi}}{\tan\theta}. \tag{2}$$

For the purpose of estimating the ball trajectory we need the full equations of motion, which we derive using the principle of Lagrange-D'Alembert [13] for systems with nonholonomic velocity constraints

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\rho}_i} - \frac{\partial L}{\partial \rho_i} = \sum_{j=1}^{n} \lambda_j a_i^j, \tag{3}$$

where ρ_i are the degrees of freedom and $L = T - U$ is the Lagrangian consisting of the kinetic energy $T = mv^2/2$ and potential energy $U = -mgz$. The right side of the equation incorporates the velocity constraints a_i, as shown in Eq. (2), and their respective Lagrangian multipliers λ_i

$$\boldsymbol{a}^\mathsf{T}\dot{\boldsymbol{q}} = \begin{bmatrix} 0 & r\left[\frac{1}{\tan\theta} - \frac{1}{\sin\theta}\right] & 1 & 1 \end{bmatrix} \left[\dot{\theta}, \dot{\varphi}, \dot{l}, \dot{h}\right]^\mathsf{T}. \tag{4}$$

By expanding the Lagrange-D'Alembert equation we obtain

$$\begin{aligned}
\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} &= 0, \\
\frac{d}{dt}\frac{\partial L}{\partial \dot{\varphi}} - \frac{\partial L}{\partial \varphi} &= \lambda_1 \frac{r}{\sin\theta} - \lambda_2 \frac{-r}{\tan\theta}, \\
\frac{d}{dt}\frac{\partial L}{\partial \dot{l}} - \frac{\partial L}{\partial l} &= \lambda_1, \\
\frac{d}{dt}\frac{\partial L}{\partial \dot{h}} - \frac{\partial L}{\partial h} &= \lambda_2.
\end{aligned} \tag{5}$$

The resulting differential equations and constraints are solved for the accelerations $\{\ddot{\theta}, \ddot{\phi}, \ddot{l}, \ddot{h}\}$ and the two Lagrangian multipliers $\{\lambda_1, \lambda_2\}$. Numerically integrating the initial state $\{\theta, \dot{\theta}, \phi, \dot{\phi}, l, h\}$ according to these equations allows us to predict the motion of the ball.

The analytical player uses this model to estimate and continuously update the ball trajectory until interception. This trajectory is transformed into the Cartesian frame of the player and used to choose a hitting point that lies in the plane defined by the pole and origin of the robot (see Fig. 2d). Once such a point is found on a trajectory with a rotation direction corresponding to the hitting direction of the player, the time and position information are passed on to the lower controller to plan and execute the appropriate joint movement.

B. Generating Hitting Trajectories

Given the desired interception point $\boldsymbol{x}_{\text{des}}$, the low-level controller starts by solving a constrained inverse kinematics problem

$$\boldsymbol{q}_{\text{des}} = f_{\text{kin}}^{-1}(\boldsymbol{x}_{\text{des}}) \quad \text{s.t.} \quad \boldsymbol{q}_{\min} < \boldsymbol{q}_{\text{des}} < \boldsymbol{q}_{\max}. \tag{6}$$

Thereupon, it generates a minimum jerk trajectory for a smooth transition to the hitting posture while satisfying the end time and speed constraints. A minimum jerk trajectory is represented by a fifth order polynomial $\boldsymbol{q}_t = \sum_{i=0}^{5} \boldsymbol{c}_i t^i$ whose parameters $\boldsymbol{c}_i$ reflect the start and end conditions of a trajectory. In the end, the joint trajectory is transformed into a series of motor torques $\boldsymbol{u}_i$ by an underlying PD-controller with gravity compensation.

IV. Learning to Play Tetherball

After introducing the analytical player, based on a mechanical model of tetherball, we show how to formulate the problem of tetherball as a contextual episodic Markov Decision Process (MDP) [9] such that we can employ reinforcement learning (RL) algorithms to learn how to play it. We formulate the problem with contexts $\boldsymbol{s} \in \mathcal{S}$, actions $\boldsymbol{\theta} \in \Theta$ and returns $R_{\boldsymbol{\theta},\boldsymbol{s}} \in \mathcal{R}$. The goal of the robot is to learn a context-dependent policy $\pi(\boldsymbol{\theta}|\boldsymbol{s})$ which selects actions according to the context and can, thus, generalize to different scenarios. We choose this distribution to be a Gaussian with linear mean $\pi(\boldsymbol{\theta}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{a} + \boldsymbol{A}\boldsymbol{s}, \boldsymbol{\Sigma})$. We use Dynamic Motor Primitives (DMPs) proposed by Ijspeert et al. [7] to represent trajectories. DMPs allow us to parametrize trajectories such that one vector of actions $\boldsymbol{\theta}$ defines a desired trajectory to be followed by the robot. In this setting, the context $\boldsymbol{s}$ defines the initial Cartesian position and velocity of the ball as well as the pivot point of the rope at the start of the episode, i.e., $\boldsymbol{s} = \{\boldsymbol{b}_0, \dot{\boldsymbol{b}}_0, h_0\}$. We define each player turn as an episode. A turn starts when a player hits or misses the ball.

Alternatively, the tetherball task could be formulated as infinite horizon problem where a policy $\pi(\boldsymbol{u}|\boldsymbol{x})$ directly selects control actions at every time step. However, formulating real robot learning problems based on time indexed states and actions is difficult for several important reasons. Firstly, a high control and sensory sampling frequency can lead to effects that render the problem Markovian only in a higher order. This means that effects of control actions will not be registered immediately by the sensors, and a history over several time steps has to be kept to make the learning problem feasible. Reducing the frequency too much can lead to jerky behaviors or controllers that do not resolve the desired behavior finely enough to solve the problem at hand. Even worse, it is possible that there is no 'sweet spot' between the two extremes at all. Non-Markovian behavior in the proposed platform is additionally introduced due to the fact that the state of the springs and the extension of the cables cannot be directly observed. Furthermore, solving the optimization problem with a time-dependent policy requires exploration noise for all states. Requiring such noise on all states should be avoided for two reasons. First, it generates non-smooth trajectories which may be not only undesirable but usually harmful to the robot. Second, many step-based RL methods require the ability to sample from the state space *during* the trajectory, i.e., to be able to set the robot and the environment to an arbitrary state, which obviously is physically impossible.

However, while trajectory based learning methods can be easy to use, increase performance and reduce the learning time in many scenarios, they are not suitable for arbitrary problems. If, for example, feedback during the trajectory is crucial, step-based RL methods can be more appropriate.

A. Compact Skill Representation

As stated above, DMPs offer a compact representation of basic movements. Formally, DMPs are defined as a second order dynamic system that acts like a spring-damper system which is driven by a non-linear forcing function $\boldsymbol{f}(z_t, \boldsymbol{\theta})$

$$\begin{aligned} \ddot{\boldsymbol{q}}_t &= \tau^2 \left(\alpha \left(\beta \left(\boldsymbol{g} - \boldsymbol{q} \right) - \frac{\dot{\boldsymbol{q}}}{\tau} \right) + \boldsymbol{f}(z_t, \boldsymbol{\theta}) \right), \\ \dot{z}_t &= -\tau \alpha_z z_t, \end{aligned} \tag{7}$$

where $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$ are the joint positions, velocities and accelerations respectively. The unique goal attractor position and velocity are defined by $\boldsymbol{g}, \dot{\boldsymbol{g}}$. Variable z_t denotes the phase and τ is a temporal scaling factor. Finally, α, β, α_z are fixed parameters. Commonly, a separate DMP is used for each joint of the robot and the phase z_t is shared between all joints in order to synchronize them. The forcing function is linear in its weights $\boldsymbol{\theta}$ but non-linear in the phase z_t, i.e., $\boldsymbol{f}(z_t, \boldsymbol{\theta}) = \boldsymbol{\psi}(z_t)^\mathsf{T} \boldsymbol{\theta}$. Usually, for stroke-based movements, normalized Gaussian basis functions are used. A crucial aspect that affects the accuracy of the resulting trajectory is the number of bases used, as they influence learning speed and expressiveness of the final policy. We evaluate the effects of this choice in the experimental section.

The resulting desired trajectory represented by DMPs is then followed by the same PD-controller with gravity compensation that is used by the analytical player. However, even if the parameters of the PD-controller are not precisely tuned to allow perfect tracking of the input trajectory, the RL agent can learn to produce input trajectories that compensate for the tracking error of the PD-controller, while the analytical player depends on accurate tracking of the desired trajectory $\boldsymbol{\tau}_{\text{des}}$.

B. Imitation Learning of DMPs

When using DMPs to generate desired trajectories, we can easily bootstrap the learning process by demonstrating an initial trajectory, i.e., by imitation learning. In many cases, bootstrapping by imitation learning is essential to make the learning process feasible, as it not only provides a good initial policy but also helps avoiding sampling trajectories that are dangerous to the robot and its environment.

More formally, given a desired joint trajectory $\boldsymbol{\tau}_{\text{des}} = [\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i, \ddot{\boldsymbol{q}}_i]_{i=1:T}$, a unique goal attractor $\boldsymbol{g}, \dot{\boldsymbol{g}}$, fixed parameters α, β, α_z and a temporal scaling factor τ, we can compute the forcing function $\boldsymbol{f}(z_t, \boldsymbol{\theta})$ for each time step and obtain the parameters $\boldsymbol{\theta}$ by linear regression

$$\begin{aligned} \boldsymbol{f}(z_t, \boldsymbol{\theta}) &= \frac{\ddot{\boldsymbol{q}}_t}{\tau^2} - \alpha \left(\beta(\boldsymbol{g} - \boldsymbol{q}_t) + \frac{\dot{\boldsymbol{g}} - \dot{\boldsymbol{q}}_t}{\tau} \right) \\ \boldsymbol{\theta} &= \left(\boldsymbol{\Psi}^\mathsf{T} \boldsymbol{\Psi} + \lambda \boldsymbol{I} \right)^{-1} \boldsymbol{\Psi}^\mathsf{T} \boldsymbol{f}(z_t, \boldsymbol{\theta}) \end{aligned} \tag{8}$$

where $\boldsymbol{\Psi} = [\boldsymbol{\psi}(z_1), \ldots, \boldsymbol{\psi}(z_T)]$.

In the tetherball games, the movement we want to imitate using DMPs consists of several phases, i.e., the robot starts from its initial resting position, hits the ball and finally goes back to the resting position. While it is already challenging to learn such a movement for fixed initial conditions, we require the robot to generalize to different contexts, i.e., ball positions and velocities to win a match. Thus, the robot needs to adapt shape and execution speed of the DMPs according to new contexts without re-learning the whole task. This generalization is represented by the linear factor in the Gaussian policy. To initialize the policy $\pi(\boldsymbol{\theta}|\boldsymbol{s})$, we use a set of M initial demonstrations paired with different contexts. Given a dataset $\mathcal{D} = \{\boldsymbol{y}_i, \boldsymbol{s}_i\}_{i=1:M}$, we obtain M parameterizations $\boldsymbol{\theta}_i$ using imitation learning and subsequently initialize $\boldsymbol{a}, \boldsymbol{A}, \boldsymbol{\Sigma}$ with linear regression

$$\begin{aligned} \boldsymbol{\Theta} &= \left[\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_N \right]^\mathsf{T}, \\ \boldsymbol{\Phi} &= \left[\left[\boldsymbol{s}_1, \cdots, \boldsymbol{s}_N \right]^\mathsf{T}, \boldsymbol{1} \right], \\ \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{a} \end{bmatrix} &= (\boldsymbol{\Phi}^\mathsf{T} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi} \boldsymbol{\Theta}, \\ \boldsymbol{\Sigma} &= \mathrm{Cov}(\boldsymbol{\Theta}). \end{aligned} \tag{9}$$

When using DMPs we can choose whether to only provide a single demonstration to determine the mean of the initial policy or to provide multiple demonstrations to also determine the covariance of the initial policy. We evaluate the effects of this decision in the experimental section.

C. Contextual Episodic RL

While imitation learning is a powerful approach to initialize the policy, it is often not sufficient to solve the learning problem. Thus, the robot relies on RL methods to further refine the policy. The goal for the robot, then, is to learn a policy $\pi(\boldsymbol{\theta}|\boldsymbol{s})$ that selects trajectories which hit the ball and wind it around the pole for arbitrary initial ball positions and velocities. More formally, the robot aims to find the policy π that maximizes the reward function

$$\begin{aligned} R^\pi &= \int_{\boldsymbol{s}} \mu(\boldsymbol{s}) \int_{\boldsymbol{\theta}} \pi(\boldsymbol{\theta}|\boldsymbol{s}) R_{\boldsymbol{\theta},\boldsymbol{s}} \, \mathrm{d}\boldsymbol{\theta} \, \mathrm{d}\boldsymbol{s}, \\ \pi^*(\boldsymbol{\theta}|\boldsymbol{s}) &= \arg\max_\pi R^\pi, \end{aligned} \tag{10}$$

where $\mu(\boldsymbol{s})$ is the distribution over contexts and $R_{\boldsymbol{\theta},\boldsymbol{s}}$ is the expected reward over all possible trajectories executed with actions $\boldsymbol{\theta}$ in context $\boldsymbol{s}$, i.e., $R_{\boldsymbol{\theta},\boldsymbol{s}} = \int_{\boldsymbol{\tau}} p(\boldsymbol{\tau}|\boldsymbol{\theta}, \boldsymbol{s}) R(\boldsymbol{\tau}, \boldsymbol{s}) \, \mathrm{d}\boldsymbol{\tau}$. The reward function $R(\boldsymbol{\tau}, \boldsymbol{s})$ is used to determine the quality of trajectory $\boldsymbol{\tau}$ executed during an episode with context $\boldsymbol{s}$. For the robot tetherball game, we formulate the reward in terms of four components p_i, i.e., $R(\boldsymbol{\tau}, \boldsymbol{s}) = p_1 + p_2 + p_3 + p_4$. The first term $p_1 = k_1(1 - \exp(d^2))$ reflects the minimum distance d between the paddle and the ball during the episode. Component $p_2 = k_2 J$ penalizes the total jerk of all joints during the whole trajectory. The third term p_3 is a penalty occurring when the robot misses the ball or hits it in the wrong direction. Finally, p_4 is an additional penalty given when the trajectory would cause a collision with the base the robot is standing on. The scale factors k_i are to transform costs into rewards and to scale the objectives magnitude.

To solve the problem expressed in Eq. (10), we use contextual relative entropy policy search (REPS) [14]. The update step for the policy $\pi(\boldsymbol{\theta}|\boldsymbol{s})$ is defined as the solution of the constrained optimization problem that determines the

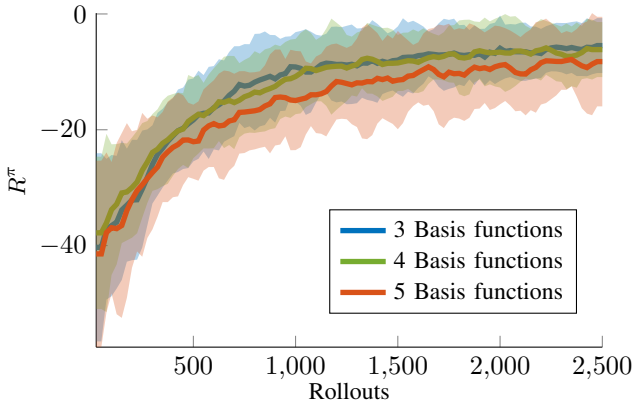


Fig. 3: Results for different number of Gaussian basis functions per DoF used for the DMP. Four bases achieve the best results. Using only three leads to less smooth trajectories, while using five bases increases the complexity of the learning problem without improving the quality of the final policy.

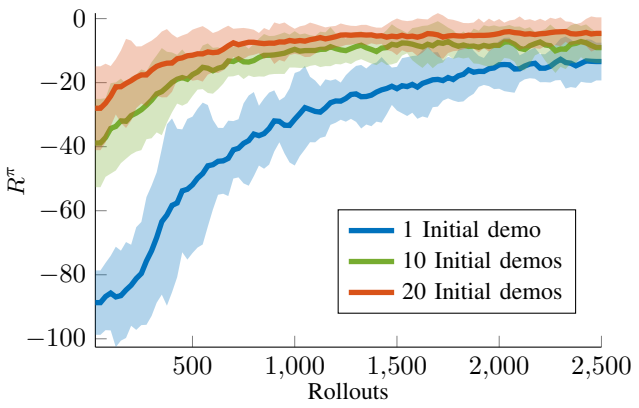


Fig. 4: Comparison over different number of demonstrations to initialize the policy π. Using only a single demonstration, the initial policy performance is considerably worse, as it is too explorative. Providing more demonstrations bootstraps the learning process and leads to better final policies.

distribution $p(\boldsymbol{s}, \boldsymbol{\theta}) = \mu(\boldsymbol{s})\pi(\boldsymbol{\theta}, \boldsymbol{s})$ that maximizes the average return R^π. At the same time, REPS bounds the relative entropy (also called Kullback-Leibler divergence) between the new distribution p and the current one q to stay close to the observed data to balance exploration and exploitation, i.e., $\int_{\boldsymbol{s}} p(\boldsymbol{s}, \boldsymbol{\theta}) \log \left(p(\boldsymbol{s}, \boldsymbol{\theta}) / q(\boldsymbol{s}, \boldsymbol{\theta}) \right) \mathrm{d}\boldsymbol{s} \leq \epsilon$. However, the context distribution $p(\boldsymbol{s}) = \int_{\boldsymbol{\theta}} p(\boldsymbol{s}, \boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}$ cannot be chosen freely, as it is specified by $\mu(\boldsymbol{s})$, i.e., we need to satisfy the constraint $\forall \boldsymbol{s} : p(\boldsymbol{s}) = \mu(\boldsymbol{s})$. For continuous context vectors, we implement this constraint by matching feature averages instead of single probability values, i.e., $\int p(\boldsymbol{s}) \boldsymbol{\varphi}(\boldsymbol{s}) \mathrm{d}\boldsymbol{s} = \hat{\boldsymbol{\varphi}}$, where $\boldsymbol{\varphi}(\boldsymbol{s})$ is a feature vector describing the context and $\hat{\boldsymbol{\varphi}}$ is the mean observed feature vector. The resulting constrained optimization problem is given by

$$\begin{aligned} \max_p & \iint_{\boldsymbol{s},\boldsymbol{\theta}} p(\boldsymbol{s}, \boldsymbol{\theta}) R_{\boldsymbol{\theta},\boldsymbol{s}} \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{\theta}, \\ \text{s.t.} \quad & \iint_{\boldsymbol{s},\boldsymbol{\theta}} p(\boldsymbol{s}, \boldsymbol{\theta}) \log \frac{p(\boldsymbol{s}, \boldsymbol{\theta})}{q(\boldsymbol{s}, \boldsymbol{\theta})} \, \mathrm{d}\boldsymbol{s} \leq \epsilon, \\ & \iint_{\boldsymbol{s},\boldsymbol{\theta}} p(\boldsymbol{s}, \boldsymbol{\theta}) \boldsymbol{\varphi}(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{\theta} = \hat{\boldsymbol{\varphi}}, \\ & \iint_{\boldsymbol{s},\boldsymbol{\theta}} p(\boldsymbol{s}, \boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{\theta} = 1. \end{aligned} \tag{11}$$

Using the method of Lagrangian multipliers we derive the closed form for the new distribution update step

$$p(\boldsymbol{s}, \boldsymbol{\theta}) \propto \exp\left(\frac{R_{\boldsymbol{s},\boldsymbol{\theta}} - V(\boldsymbol{s})}{\eta} \right), \tag{12}$$

where the value function $V(\boldsymbol{s}) = \boldsymbol{\gamma}^\mathsf{T} \boldsymbol{\varphi}(\boldsymbol{s})$ is a context-dependent baseline, while η and $\boldsymbol{\gamma}$ are the Lagrangian parameters, found by optimizing the dual function.

As the relationship between the context-policy parameters pair $\{\boldsymbol{s}, \boldsymbol{\theta}\}$ and the corresponding expected reward $R_{\boldsymbol{\theta},\boldsymbol{s}}$ is not known, sample rollouts are used to approximated the integrals in Eq. (11). To execute the i-th rollout, we first observe the context $\boldsymbol{s}^{[i]} \sim \mu(\boldsymbol{s})$. Subsequently, we sample the DMPs parameters $\boldsymbol{\theta}^{[i]} \sim \pi(\theta | \boldsymbol{s}^{[i]})$. Finally, we execute the DMPs with parametrization $\boldsymbol{\theta}^{[i]}$ in context $\boldsymbol{s}^{[i]}$ to obtain reward $R^{[i]}_{\boldsymbol{\theta},\boldsymbol{s}}$. Repeating this process for N rollouts, we obtain the average return R^π and the weights $p^{[i]}$ for a maximum likelihood update of the distribution $p(\boldsymbol{s}, \boldsymbol{\theta})$

$$\begin{aligned} p^{[i]} &\propto \exp\left(\frac{R^{[i]}_{\boldsymbol{\theta},\boldsymbol{s}} - V(\boldsymbol{s}^{[i]})}{\eta} \right), \\ R^\pi &= \langle R_{\boldsymbol{\theta},\boldsymbol{s}} \rangle. \end{aligned} \tag{13}$$

The quality of the learned policy and the convergence speed depend on the number of episodes N and by the basis functions $\boldsymbol{\varphi}$ used. Using many rollouts helps reducing the variance of an update step, but may also increase the number of episodes to converge. In the same way, using many features to approximate $V(\boldsymbol{s})$ can slow down the solution of the constrained optimization problem. We evaluate the effects of these decisions in the experimental section.

V. Evaluation

After detailing both the analytical player as well as the RL-based player, we evaluate both and show the results in this section. We start by investigating the effects of the different parameters on the learned player and, subsequently, present the results of comparing the analytical player to the learned player. The evaluations done in simulation are averaged over ten trials, while the evaluations done on the real robot are averaged over three trials. As REPS does not rely on using samples from only the last policy, we base the policy update on samples from the last 20 policies to stabilize the policy update. This stabilization is important to keep the shape of the explorative variance of the policy in high dimensional action spaces. The KL bound for all experiments in Eq. (11) is set to $\epsilon = 0.9$. The parameters, α_z, α, β and τ of the DMPs are set to $\alpha_z = 0.8, \alpha = 5, \beta = 1, \tau = 1$. These parameters are chosen to reflect a qualitative trajectory shape that broadly fits the stroke profiles in our task.

For the real robot experiments, a Kinect sensor mounted on the ceiling delivers color and depth images at a rate of 30Hz.

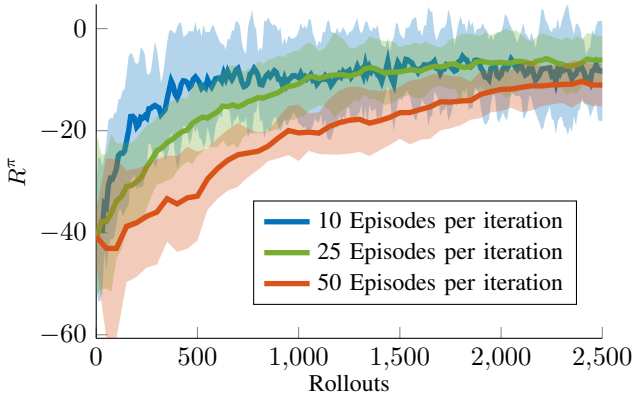


Fig. 5: Evaluation of different number of episodes per iteration. With ten samples the algorithm already performs well. Providing 25 samples slightly improves asymptotic quality but requires more overall samples. With 50 samples the policy does not converge in the limit of 2,500 total evaluations.

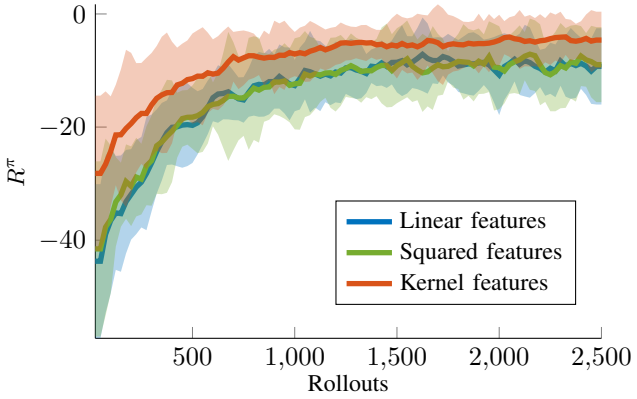


Fig. 6: Comparison over different types of basis functions to approximate the value function. While kernel based features are computationally more intensive and generally require more samples to work well, they also increase performance of the final policy.

As the pivot point of the string cannot be detected by the vision system, an unscented Kalman filter [15] approximates the translation of the pivot point h along the pole.

A. Evaluation of DMP Basis Functions

An important parameter of DMPs is the number of basis function $\psi(z_t)$ used to parametrize the forcing function $\boldsymbol{f}(z_t, \boldsymbol{\theta})$. A higher number of basis functions allows for more complicated trajectory shapes but also increases the dimensionality of the action space Θ and, thus, makes the learning problem harder. Fig. 3 shows the comparison of learning with different numbers of Gaussian basis functions $\psi(z_t)$. The plot shows that using three or four basis functions leads to comparable results, while using five basis functions per joint is increasing the complexity offsetting the potential benefit of learning more expressive trajectories. While the expected reward achieved with three and four basis functions is similar, the trajectories generated using four basis functions are overall smoother. Therefore we use four basis function for all following experiments.

B. Selection of Initial Demonstration

We also investigate the number of demonstration recorded to initialize the policy $\pi(\boldsymbol{\theta}, \boldsymbol{s})$. Fig. 4 shows that providing only a single demonstration leads to poor results. The reason is that with only one demonstration the initial covariance $\boldsymbol{\Sigma}$ needs to be set manually and uniformly in all dimensions (in our trials, $\boldsymbol{\Sigma} = 100\boldsymbol{I}$). This decision is not straightforward and can produce policies that are too explorative, significantly increasing the convergence time. Indeed, it can be noticed that after 2,500 episodes, the policy is still improving, but its quality is far below those policies whose initial exploration bounds were determined through additional demonstrations. When using multiple demonstrations, we choose a set of trajectory-context pairs that span the largest context space possible. We also allow low quality demonstrations in which the robot misses the ball. As expected, the more demonstrations are provided, the better the quality of the initial policy is. However, in order to restrict the number of demonstrations to a reasonable value, we decide not to use more than 20 demonstrations.

C. Evaluation of Value Function Features

Contextual REPS depends on a feature representation $\boldsymbol{\varphi}(\boldsymbol{s})$ of the state $\boldsymbol{s}$ to predict the expected quality $V(\boldsymbol{s})$ of that state. While simpler features like linear or squared features can be more robust and sample efficient, kernel based features promise better performance for complex problems. Fig. 6 shows that for the problem of robot tetherball kernel based features indeed outperform the simpler alternatives. While linear and squared features can be used to achieve acceptable performance, the greater flexibility of kernel based features offers an edge in due to the non-linearity of the task.

D. Selection of Number of Rollouts

An important trade-off for any policy search algorithm is the number of evaluations used per policy update. While more samples yield more stable updates, they also slow down the learning process. Fig. 5 shows that when using only 10 episodes the reward quickly increase but the asymptotic performance is slightly below what can be achieved when using 25 rollouts per iteration. Using 50 episodes slows down the learning process considerably and does not lead to a converged policy after the limit of 2,500 total episodes. The plot for using ten samples per iteration is more noisy as it is averaged over fewer rollouts per data point.

E. Analytical vs. Learned Player

According to the results reported, we set the learned player to use 20 initial demonstrations, four Gaussian basis functions for the DMP, kernel based features with three centers for approximating value function, and 25 episodes per iteration for the policy update. To get an initial estimate of their relative quality, we compare the learned player to the analytical one in a single player experiment using the hit rate as measure of evaluation. The comparison is performed

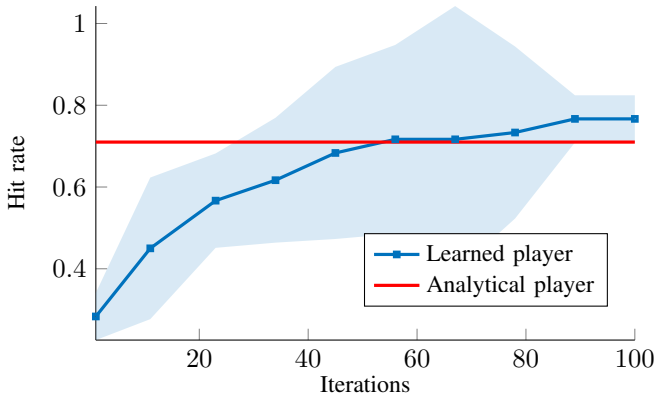


Fig. 7: Comparison of the real robots in the single player setup, averaged over three trials and ten rollouts per iteration. The learned player reaches performance levels similar to the analytical one after about 50 iterations and outperforms it at the end of learning.

on the real system over three trials, every ten iterations and over ten episodes, summing up to 300 rollouts.

As shown in Fig. 7, the learned player outperforms the hand-crafted system. The advantage of the learned system is due to its ability to learn to compensate the highly non-linear forward dynamics of the robot, as caused by springs in the joints. On the other hand, lacking a full inverse dynamics model, the analytical player fully depends on accurate tracking of the planned trajectory. Therefore, improving the analytical player would require a time-consuming process to explicitly approximate the dynamics of the robots.

F. Two Player Full Tetherball

As final evaluation, we let the real robots play full matches against each other and compare them according to the score. A match is started by the referee throwing the ball randomly to one of the robots and ends when none of the robots is able to hit the ball anymore. The score of a match is determined by how often the ball winds around the pole for each player, i.e., by the number of misses of the opponent. The two robots played a total of 25 games. The analytical player won six of the 25 games, while the learned player won the remaining 19 games. Throughout the 25 games, the analytical player scored eight times, while the learned player scored 38 times.

VI. Conclusion

While learning policies to solve complex tasks is an appealing concept, no thorough comparison of a learned skill versus a hand-crafted solution has been presented so far. In this paper we showed the steps necessary to build both a high quality analytical player as well as a learned player that uses state-of-the-art modules in reinforcement learning for the game of robot tetherball. Although the learning modules do require the tuning of open parameters, doing so is far less time intensive than building a mathematical model for a given task. Moreover, the learning approach achieved better results, as the learned player outperforms the hand-crafted opponent. We also evaluate the open parameters of the RL components used in this paper and give insights into how these parameters can be chosen for arbitrary episodic problems. Our results show that the number of rollouts per iteration and number of initial demonstrations are crucial for sample efficiency in high-dimensional context space. As tasks requiring human participation can pose a limit on sample complexity, we want to investigate the use of importance sampling techniques and model-based approaches. Finally, we also plan to incorporate online feedback by extending our framework to step-based RL approaches.

TABLE I: Match results for the real robots. The learned player defeats the analytical one and achieves better overall hit rate, averaged over 25 matches.

Player	Hit rate	Matches won	Total score
Analytical	71%	6/25	8
Learned	**85%**	**19/25**	**38**

Acknowledgement

The authors want to thank for the support of the German Research Foundation project # PE 2315/2-1 (ScARL) and of the European Union project # FP7-ICT-2011-9 (CoDyCo).

References

[1] J. Kober and J. Peters, "Policy search for motor primitives in robotics," in *Advances in Neural Information Processing Systems (NIPS)*, 2009.
[2] P. Kormushev, S. Calinon, and D. G. Caldwell, "Reinforcement learning in robotics: Applications and real-world challenges," *Robotics*, vol. 2, no. 3, 2013.
[3] V. Gullapalli, J. A. Franklin, and H. Benbrahim, "Acquiring robot skills via reinforcement learning," *Control Systems*, vol. 14, no. 1, 1994.
[4] C. Daniel, G. Neumann, and J. Peters, "Learning concurrent motor skills in versatile solution spaces," in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2012.
[5] T. Matsubara, J. Morimoto, J. Nakanishi, M. Sato, and K. Doya, "Learning sensory feedback to cpg with policy gradient for biped locomotion," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2005.
[6] E. Theodorou, J. Buchli, and S. Schaal, "Learning policy improvements with path integrals," in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.
[7] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *Advances in Neural Information Processing Systems (NIPS)*, 2002.
[8] S. Schaal, A. Ijspeert, and A. Billard, "Computational approaches to motor learning by imitation," *Philosophical Transactions of the Royal Society. Series B: Biological Sciences*, vol. 358, no. 1431, 2003.
[9] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics." *Foundations and Trends in Robotics*, vol. 2, no. 1-2, 2013.
[10] C. Daniel, G. Neumann, and J. Peters, "Hierarchical relative entropy policy search," in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.
[11] T. Lens and O. von Stryk, "Design and dynamics model of a lightweight series elastic tendon-driven robot arm," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2013.
[12] H. Abdulsamad, T. Buchholz, T. Croon, and M. El Khoury, "Playing tetherball with compliant robots," TU Darmstadt, Tech. Rep., 2014.
[13] A. M. Bloch, *Nonholonomic mechanics and control*. Springer, 2003, vol. 24.
[14] J. Peters, K. Muelling, and Y. Altun, "Relative entropy policy search," in *Proceedings of the Conference on Artificial Intelligence (AAAI)*, 2010.
[15] D. Simon, *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.