High Acceleration Reinforcement Learning for Real-World Juggling with Binary Rewards

Kai Ploeger*, **Michael Lutter***, **Jan Peters**
Computer Science Department, Technical University of Darmstadt
{kai, michael, jan}@robot-learning.de

Abstract: Robots that can learn in the physical world will be important to enable robots to escape their stiff and pre-programmed movements. For dynamic high-acceleration tasks, such as juggling, learning in the real-world is particularly challenging as one must push the limits of the robot and its actuation without harming the system, amplifying the necessity of sample efficiency and safety for robot learning algorithms. In contrast to prior work which mainly focuses on the learning algorithm, we propose a learning system, that directly incorporates these requirements in the design of the policy representation, initialization, and optimization. We demonstrate that this system enables the high-speed Barrett WAM manipulator to learn juggling two balls from 56 minutes of experience with a binary reward signal. The final policy juggles continuously for up to 33 minutes or about 4500 repeated catches. The videos documenting the learning process and the evaluation can be found at **https://sites.google.com/view/jugglingbot**

Keywords: Reinforcement Learning, Dynamic Manipulation, Juggling

1 Introduction

Robot learning is one promising approach to overcome the stiff and pre-programmed movements of current robots. When learning a task, the robot autonomously explores different movements and improves its behavior using scalar rewards. In recent years, research has focused a lot on improving task-agnostic deep reinforcement learning (DRL) algorithms by changing either the optimization [1, 2], the simulation to use perturbed physics parameters [3, 4], or the task to gradually increase complexity [5]. While these approaches have propelled learning robots to very complex domains in simulation, ranging from full-body control of humanoids [6] to control of dexterous hands [7, 8], most of these approaches are not applicable to learn on physical systems as they neglect the intricate complexities of the real world. On the physical system, the learning is constrained to real-time and a single instance. Hence, the learning must not damage the robot with jerky actions and must be sample efficient.

Consider the high-acceleration task of juggling two balls next to each other with a single anthropomorphic manipulator. The manipulator is required to throw a ball upwards, move to the right, catch and throw the second ball and return to the left in time to catch the first ball. To sustain this cyclic juggling pattern, the robot must always throw the ball sufficiently vertical and maintain precise timing. Therefore, this task pushes the limits of the robot to achieve the required high accelerations (of up to 8g), while maintaining precise control of the end-effector and the safety of the physical system. The task is not only inherently difficult to master[1] but also requires learning on the physical system. Real-world experience is required as simulation models, while good at simulating contact-free rigid-bodies, cannot represent the non-linear effects close to the torque limits of the actuators and the highly dynamic contacts between end-effector and balls. For the tendon driven Barrett WAM, rigid-body-simulators also cannot model the dominating cable dynamics at high accelerations. Hence, simulation-based solutions cannot be transferred for very dynamic tasks given our current simulators. For this specific task even robot to robot transfer between robots of the same

*Equal contribution

[1]For reference, the untrained human jugglers of the lab achieve 2 repeated catches and improve to about 20 repeated catches after a few hours of training.

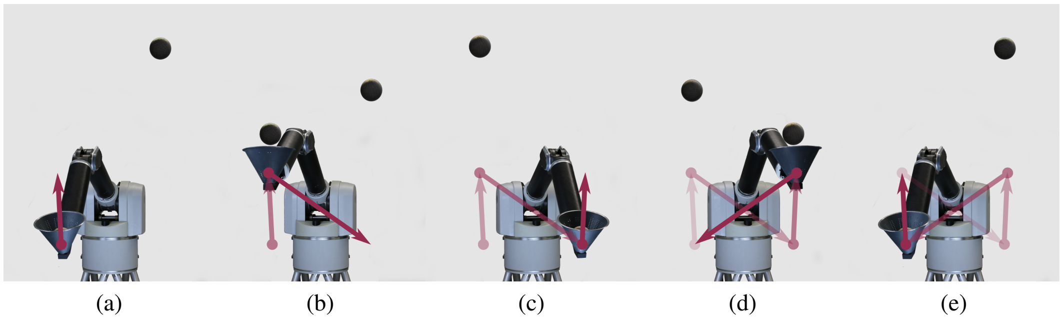


Figure 1: The juggling movement consisting of four separate movements, which are repeated to achieve juggling of two balls with a single anthropomorphic manipulator: (a) throw ball 1, (b) catch ball 2, (c) throw ball 2, (d) catch ball one, (e) repeat.

model is not possible. The learned policy fails immediately when transferred to a different Barrett WAM. Therefore, the optimal policy depends on the exact robot instance and must be learned on the individual robot. The high accelerations amplify the safety and sample efficiency requirements for learning on the physical system as collisions at high velocities severely damage the system. Furthermore, the high acceleration movements induce a lot wear and tear and cannot be executed for days. Therefore, high acceleration tasks are an ideal problem to test the limitations of current learning algorithms on the physical system.

To emphasize important considerations for building a real-world learning system for high-acceleration tasks, we describe our robot learning setup to learn juggling and the taken system design considerations. We limit ourselves to *existing* methods to focus solely on the requirements for the real-world application of current learning approaches. This is in contrast to many prior works, which mainly propose new policy representations or policy optimizations and demonstrate that the new algorithm can be applied to the physical system. Instead, we focus on a single challenging task and evaluate which existing representations and algorithms can be used to build a safe and sample efficient robot learning system. We also highlight limitations of learning approaches, which - given the current knowledge - we do not dare to apply to the physical system for robot juggling. Afterwards, we describe one approach to solve robot toss juggling with two balls and an anthropomorphic manipulator and validate the chosen approach in the real-world. The used approach - in our opinion - optimally combines engineering expertise and learning to obtain a reliable, safe, and sample efficient solution for this task.

Our contribution is (1) the application of robot learning to the challenging task of single-arm toss juggling with two balls and (2) highlighting the challenges of applying a learning system in the physical world for a high-acceleration task. In the following, we first cover the prior work on robot juggling and learning dynamical tasks on the physical robot in Section 2. Afterwards, we compare different approaches to learn real-world toss juggling in Section 3 and describe the implemented learning system in Section 4. The experimental setup and results are presented in Section 5.

2 Related Work

2.1 Robot Juggling

For decades robot juggling has been used to showcase the ingenuity of mechanical system design and control engineering as listed in table 1. Starting with Claude Shannon in the 1970s, many different juggling machines were built.[2] For example, the Shannon juggler used cups to bounce up to five balls off a surface [9], the devil-sticking machine stabilized a stick in the air [10, 11], paddle jugglers, built from designated hardware [12, 9] or by attaching tennis rackets to manipulators [13, 14, 15, 16, 17, 18, 19], juggled multiple balls using paddling. Toss jugglers were built using manipulators to juggle one [20, 21] or two balls [22] and even humanoids were used to juggle up to three balls using two arms [23, 24]. Most of these approaches proposed new engineering solutions for movements and controllers showing that these systems achieve juggling when the control parameters are manually

[2]A historic overview of robot juggling is available at https://www.youtube.com/watch?v=2ZfaADDlH4w.

fine-tuned. Only a few approaches used supervised learning for model learning [14, 11], behavioral cloning [18], or evolutionary strategies with dense fitness functions [25] to achieve paddle juggling and devil sticking. We build upon the vast experience on end-effector and controller design for robot juggling but in contrast to the prior work, we demonstrate, to the best of our knowledge, the first robot learning system that learns toss juggling with two balls and a single anthropomorphic manipulator in the real world using only binary rewards. Using this approach, we achieve juggling of up to 33 minutes and high repeatability between trials.

Table 1: Prior work on different types of robot juggling

Juggling Type	**Approach**	**Papers**
Devil Sticking	Engineered	[9]
Devil Sticking	Model Learning	[11]
Paddle Juggling	Engineered	[15, 16, 9, 17, 12, 19]
Paddle Juggling	Imitation	[18]
Paddle Juggling	Model Learning	[14]
Paddle Juggling	Evolutionary Strategies	[25]
Toss Juggling	Engineered	[22, 24, 23, 21, 20]
Toss Juggling	**Reinforcement Learning**	[**Ours**]

2.2 Learning Dynamical Tasks on the Physical Robot

Despite the recent surge of deep reinforcement learning algorithms for controlling robots, most of these approaches are constrained to learn in simulation due to sample complexity and the high risk of catastrophic policies. Only relatively few DRL approaches have been applied to physical robots, e.g., robot manipulators [26, 27, 8, 28, 29, 30] or mobile robots [31, 32]. Most of the work for manipulators focuses on non-dynamic tasks. Only Schwab et al. [30] and Büchler et al. [33] applied DRL to learn the dynamic tasks of Ball-in-a-Cup or robot table tennis. In [30], the authors built a fully automated environment to achieve large-scale data collection and engineered classical safety mechanisms to avoid damaging the physical system. Using the safe and automated environment, SAC-X was able to learn Ball-in-a-Cup from raw pixels within three days [30]. Most other approaches for learning dynamical tasks on the physical system use more classical robot learning techniques. These algorithms combine engineering- and task knowledge with learning to achieve sample efficient and safe learning that does not require completely safe and fully automated environments. For example, combining model learning with trajectory optimization/model predictive control [34, 8, 11] or model-free reinforcement learning with engineered policy representation, expert policy initialization, and dense rewards [35, 36, 18, 37, 38]. In our work, we extend the classical robot learning approach to a robot learning system that learns the high acceleration task of juggling with designed feature representations, expert policy initialization, and binary rewards instead of dense rewards. We also discuss the necessary design decisions that incorporate engineering and task expertise to achieve safety and sample efficiency. This focus is in contrast to most prior work as these mostly highlight the details of the learning algorithms but not the many engineering details that enable learning on the physical system.

3 System Design for High-Acceleration Tasks

Designing a robot learning system for robot juggling can be approached from different learning paradigms with different benefits. In the following we briefly discuss these trade-offs to motivate our approach presented in section 4.

3.1 Model-based vs. Model-free Learning

To minimize the wear and tear during the high acceleration movements and minimize the manual resets of picking up the balls, one desires to learn with minimal trials. Commonly model-based reinforcement learning (MBRL) methods are much more sample efficient compared to model-free reinforcement learning (MFRL) [39, 40]. However, MBRL requires to learn an accurate model describing the system such that the optimal policy transfers to the real system. For the considered task of robot juggling, the model would need to accurately describe the rigid-body dynamics of the manipulator, the stiction of the cable-drives, the contacts of the end-effectors with the balls, and the ball-movement. The model would also need to be robust to out-of-distribution model exploitation to avoid optimizing a spurious and potentially harmful solution. Out-of-distribution exploitation is especially challenging for deep networks as the networks do not generalize well to previously unexplored states and this may result in unpredictable behaviors potentially damaging the system.

We are not aware of a model-learning approach that can capture the different phenomena ranging from multi-point contacts to low-level stiction with sufficiently high fidelity and are robust to out of distribution generalization. Therefore, we resort to a MFRL approach. Besides the theoretical aspects, the practical implementation of observing the necessary quantities to construct such accurate models is challenging by itself, e.g., the collisions of the ball and end-effector are obscured by the end-effector and hence not observable.

3.2 Open-Loop Policy vs. Closed-Loop Policy

Closed-loop policies are favorable for robot juggling as the interactions between the end-effector and ball are not consistent. A closed-loop policy could recover from these irregularities. Learning a closed-loop policy for juggling is non-trivial as one closes the loop on noisy ball observation. The noisy observations or potential outliers might cause the system to become unstable. The unstable behavior might cause the robot to hit its joint limits with high momentum and severely damage the system. One would need to guarantee that the closed-loop policy is stable for all possible observations including many edge-cases. For the pre-dominant closed-loop deep network controller of deep reinforcement learning, the stable behavior cannot be guaranteed. Especially as the out of distribution prediction for networks is undefined. Currently, to the best of our knowledge, no network verification process for complex closed-loop systems exists. Therefore, we currently do not dare to apply a real-time closed-loop network policy to the physical system where the network is expected to execute accelerations of up to 8g.

Instead we are using an open-loop policy consisting of a desired position trajectory and a tracking controller. This representation is sufficient for the task as well trained jugglers can juggle basic patterns blindfolded and prior literature [9, 11, 17] has shown the stability of open-loop juggling robots. A *naive* but sufficient safety verification of this policy can be achieved by the stability of the tracking controller and by enforcing tight box constraints in joint space for the desired trajectory. For the juggling setup the box-constraints prevent self-collisions and hitting the joint limits. Hybrid approaches that adapt the desired trajectory in closed-loop w.r.t. to ball observations exist. We also tried a *naive* local adaption of the desired trajectory using a task-space controller but this adaptation even reduced system performance. The system would adapt to the balls during catching but could not throw straight afterwards. As the open-loop policy already achieved high repeatability and long juggling duration, we did not investigate more complex hybrid policies further.

4 System Implementation

4.1 Policy Representation

The probabilistic policy is defined by a normal distribution $\mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ over via-points in joint space. Each parameter $\boldsymbol{\theta} = [\mathbf{q}_0, \dots \mathbf{q}_N, \dot{\mathbf{q}}_0, \dots, \dot{\mathbf{q}}_N, t_0, \dots, t_N]$ corresponds to a possible juggling movement consisting of a via-point sequence. Each via point is defined by the position $\mathbf{q}_i$, velocity $\dot{\mathbf{q}}_i$ and duration t_i. To execute the movement, the via-points are interpolated using cubic splines and tracked by a PD controller with gravity compensation. Therefore, the motor torques at each time-step are computed by,

$$\boldsymbol{\tau} = \mathbf{K}_{\mathrm{P}}(\mathbf{q}_{\mathrm{ref}} - \mathbf{q}) + \mathbf{K}_{\mathrm{D}}(\dot{\mathbf{q}}_{\mathrm{ref}} - \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q})$$

$$\text{with} \quad \mathbf{q}_{\mathrm{ref}}(t) = \sum_{j=0}^{3} \mathbf{a}_{i,j}\ (t - t_{i,0})^j, \quad \dot{\mathbf{q}}_{\mathrm{ref}}(t) = \sum_{j=1}^{3} j\ \mathbf{a}_{i,j}\ (t - t_{i,0})^{j-1}, \quad t_{i,0} = \sum_{k=0}^{i-1} t_k$$

the control gains $\mathbf{K}_{\mathrm{P}}$ and $\mathbf{K}_{\mathrm{D}}$ and the jth parameter of the ith spline $\mathbf{a}_{ij}$. The gains are set to be low compared to industrial manipulators to achieve smooth movements. The spline parameters are computed using the via points by

$$\mathbf{a}_{i,0} = \boldsymbol{q}_i, \qquad \mathbf{a}_{i,1} = \dot{\mathbf{q}}_i, \qquad \mathbf{a}_{i,2} = 3\ (\boldsymbol{q}_{i+1} - \boldsymbol{q}_i)\, t_i^2 - (\dot{\boldsymbol{q}}_{i+1} + 2\dot{\mathbf{q}}_i)\, t_i,$$
$$\mathbf{a}_{i,3} = 2\ (\boldsymbol{q}_i - \boldsymbol{q}_{i+1})\, t_i^3 + (\dot{\boldsymbol{q}}_{i+1} + \dot{\mathbf{q}}_i)\, t_i^2.$$

We initialize the parameters with expert demonstrations to reduce the sample complexity. Especially in the case of binary rewards, such initialization is required as the reward signal is sparse. Most random initialization would not make any contact with the balls. Hence, the rl algorithm could not infer any information about the desired task. Instead of using kinesthetic demonstrations [36, 35, 18, 37, 38], we directly initialize the interpretable via points manually. This initialization is preferable for robot juggling because the human demonstrator cannot achieve the necessary accelerations using kinesthetic teaching.

The desired juggling movement for two balls consists of four repeated movements, i.e., (a) throwing the first ball, (b) catching the second ball (c) throwing the second ball, and (d) catching the first ball (Fig. 1). We define the switching points between these movements as the via-points of the policy and to achieve a limit-cycle, we keep repeating these via-points. The cyclic pattern is prepended with an initial stroke movement that quickly enters the limit cycle without dropping a ball. Applying the limit cycles PD-references from the start would result in a bad first throw. Furthermore, we enforce symmetry of cyclic pattern and zero velocity at the via point. Thus reducing the effective dimensionality from 48 open parameters to only 21.

4.2 Policy Optimization

The policy optimization is framed as an episodic reinforcement learning problem, sampling a single policy parameter $\boldsymbol{\theta}_i \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ per roll-out and evaluating the episodic reward. This framing is identical to a bandit setting with high-dimensional and continuous actions. For the physical system, the episodic exploration is favorable over step-based exploration because this exploration yields smooth and stable action sequences given the previous policy representation. To optimize the policy parameters, we use a variant of the information-theoretic policy search approach episodic Relative Entropy Policy Search (eREPS) [41, 42]. Our eREPS variation not only limits the Kullback-Leibler (KL) divergence when computing the sample weights but also enforces the reverse KL divergence when updating the policy parameters. This optimization can be solved in closed form for Gaussian distributions as described in Abdolmaleki et al. [43]. We use eREPS instead of EM-based [36, 35] or gradient-based [44, 45] policy search as the KL constraint prevents premature convergence and large, possibly unsafe, jumps of the policy mean.[3]

Let the optimization problem of eREPS be defined as

$$\pi_{k+1} = \arg\max_{\pi} \int \pi(\boldsymbol{\theta}) R(\boldsymbol{\theta}) d\boldsymbol{\theta}, \qquad \text{s.t.} \qquad d_{\mathrm{KL}}(\pi_{k+1} || \pi_k) \leq \epsilon$$

with the updated policy π_{k+1}, the episodic reward R, and the KL divergence d_{KL}. With the additional constraint of π being a probability distribution, this optimization problem can be solved by first optimizing the dual to obtain the optimal Lagrangian multiplier η^* and fitting the new policy using weighted maximum likelihood. The sample-based optimization of the dual is described by

$$\eta^* = \arg\min_{\eta} \eta\epsilon + \eta \log \sum_{i=0}^{N} \pi_k(\boldsymbol{\theta}_i) \exp\left(R(\boldsymbol{\theta}_i)/\eta\right).$$

The optimization of the weighted likelihood $\mathcal{L}$ to obtain the updated policy is described by

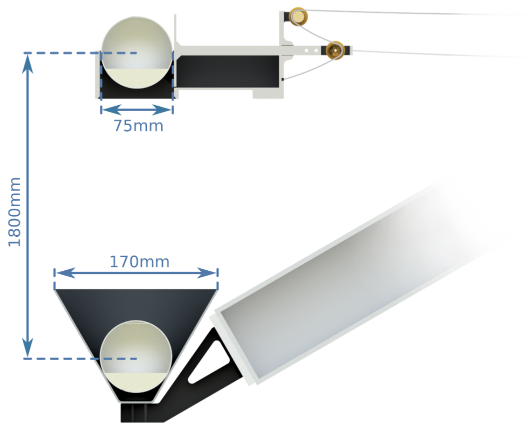
$$\boldsymbol{\mu}_{k+1},\ \boldsymbol{\Sigma}_{k+1} = \arg\max_{(\boldsymbol{\mu},\,\boldsymbol{\Sigma})} \sum_{i=0}^{N} w_i \log(\mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \boldsymbol{\theta}_i)) \qquad \text{s.t.} \qquad d_{\mathrm{KL}}(\pi_k || \pi_{k+1}) \leq \epsilon.$$

with the weights $w_i = \exp\left(R(\boldsymbol{\theta}_i)/\eta^*\right) / \sum_i \exp\left(R(\boldsymbol{\theta}_i)/\eta^*\right)$. We incorporate the reverse KL constraint to the optimization to guarantee that the policy parameters adhere to the KL constraint. In the original eREPS formulation only the KL divergence between the evaluated samples is constrained but not the KL divergence between the parametrized policies. Especially for high-dimensional problems and few sample evaluations, the KL divergence between policies is larger than ϵ without this explicit constraint. The reverse KL divergence is used as this formulation enables solving the policy update in closed form for Gaussian distributions. For the multivariate Gaussian policy distribution this update is described by

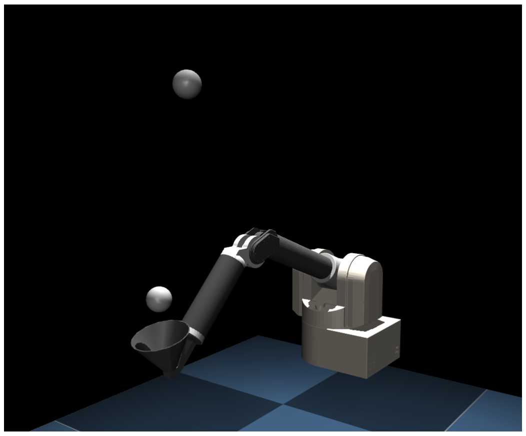
$$\boldsymbol{\mu}_{k+1} = \frac{\xi^* \boldsymbol{\mu}_k + \boldsymbol{\mu}_s}{1 + \xi^*}, \qquad \boldsymbol{\Sigma}_{k+1} = \frac{\boldsymbol{\Sigma}_s + \xi^* \boldsymbol{\Sigma}_k + \xi^* \left(\boldsymbol{\mu}_{k+1} - \boldsymbol{\mu}_k\right) \left(\boldsymbol{\mu}_{k+1} - \boldsymbol{\mu}_k\right)^T}{1 + \eta^*}$$

$$\boldsymbol{\mu}_s = \sum_{i=0}^{N} w_i\, \boldsymbol{\theta}_i \qquad \boldsymbol{\Sigma}_s = \sum_{i=0}^{N} w_i\, \left(\boldsymbol{\theta}_i - \boldsymbol{\mu}_{k+1}\right) \left(\boldsymbol{\theta}_i - \boldsymbol{\mu}_{k+1}\right)^T$$

[3]Further information about reinforcement learning for robotics can be found in the surveys Kober et al. [46] and Deisenroth et al. [41].



(a)



(b)

Figure 2: (a) Mechanical design of the juggling end-effector, the Russian style juggling balls, and the ball launcher. The end-effector consists of a funnel to achieve precise control of the throwing movement. The Russian juggle balls are partially filled with sand to avoid bouncing. (b) The MuJoCo juggling environment used for simulating the rigid-body-dynamics.

with the optimal Lagrangian multiplier ξ^*. This optimal multiplier can be obtained by solving

$$\xi^* = \arg\max_{\xi} \left[-\log(|\boldsymbol{\Sigma}_{k+1}|) - \sum_{i=0}^{N} w_i \left(\boldsymbol{\theta}_i - \boldsymbol{\mu}_{k+1}\right)^T \boldsymbol{\Sigma}_{k+1}^{-1} \left(\boldsymbol{\theta}_i - \boldsymbol{\mu}_{k+1}\right) \right]$$
$$+\xi \left[2\epsilon - \text{tr}\left(\boldsymbol{\Sigma}_{k+1}^{-1} \boldsymbol{\Sigma}_k\right) - n + \log\left(\frac{\boldsymbol{\Sigma}_{k+1}}{\boldsymbol{\Sigma}_k}\right) + \left(\boldsymbol{\mu}_{k+1} - \boldsymbol{\mu}_k\right)^T \boldsymbol{\Sigma}_{k+1}^{-1} \left(\boldsymbol{\mu}_{k+1} - \boldsymbol{\mu}_k\right) \right].$$

The complete derivation can be found in the appendix and the source code of eREPS is available at https://github.com/hanyas/rl.

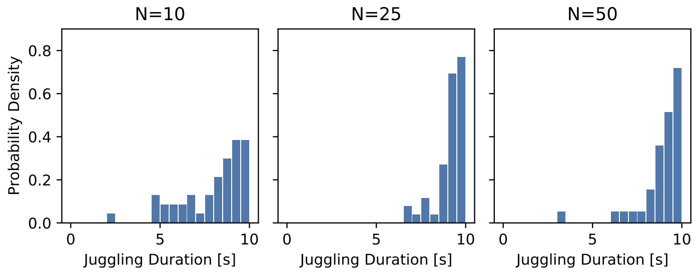
4.3 Reward Function

The reward function assigns a positive binary reward as long as the robot is juggling. Juggling is defined as keeping both balls at least 60cm above the floor, which is measured using the external marker tracker OptiTrack. Therefore, the step-based reward is described by

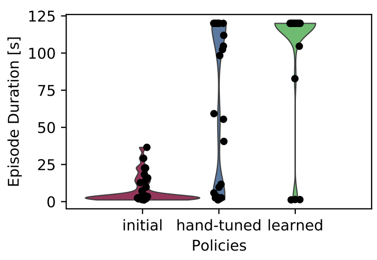
$$r(t) = \begin{cases} 1 & \text{if } \min_i(b_{i,z}) \geq 0.6 \\ 0 & \text{otherwise} \end{cases}$$

with the ith ball height $b_{i,z}$. This reward signal maximizes the juggling duration and is not engineered to incorporate any knowledge about the desired ball- or manipulator trajectory. This choice of reward function is intuitive but also uninformative as a bad action only causes a delayed negative reward. For example, a bad action within the throwing will cause a zero reward - a drop - seconds after the action. This delay between action and reward, i.e., 'credit assignment', is a challenge in many RL problems. The choice of the binary reward functions is in stark contrast to prior work as most of the previously proposed approaches use more informative dense rewards [30, 8, 36, 25].

The binary rewards are favorable as these rewards do not require tuning a dense reward function. To specify the dense reward function one would need to predict the optimal ball trajectory and compute the distance to the optimal trajectory. Especially as the optimal ball trajectory depends on the robot capabilities and end-effector this prediction is challenging. Furthermore, one would need to align the initialization of the juggling movement with the optimal ball trajectory. One could possibly initialize a good juggling movement but that might not fit with the specified dense reward, e.g., the dense reward prefers higher throws than the initialization. With the binary reward one only needs to provide a good initialization and does not need to tune the reward parameters. Besides the reward tuning aspect, evaluating the dense reward is also more challenging compared to evaluating the binary reward in the real world. For evaluating the dense reward one would require precise tracking



(a)



(b)

Figure 3: (a) Mean juggling duration of final policies learned with varying batch sizes N. The maximal juggling duration is 10s. (b) Comparison of the learned and hand-tuned policy on each 30 episodes with a maximum duration of 120s on the real system. The learned policy achieves an average juggling duration of 106.82s while the hand-tuned policy achieves 66.52s.

which is non-trivial to frequent occlusions by the end-effector. Even though we used OptiTrack to track the balls, we had a hard time achieving good tracking performance during the experiments due to the wear and tear on the reflective tape and the frequent occlusions. Every time a ball is in the end-effector, the ball is not observable.

5 Experiments

5.1 Experimental Setup

The experiments are performed with a Barrett WAM to achieve the high accelerations required for robot juggling. The 4 degrees of freedom (DoF) variant is used rather than the 7 DoF version to allow for more precise control at high velocities and to save weight. To achieve high repeatability, we use 75mm Russian style juggling balls that dissipate the kinetic energy of the balls and prevent them from bouncing out of the end-effector. These balls consist of a hollow plastic shell partially filled with 37.5g of sand (Figure 2a). The hard ball shells also result in more accurate throws compared to traditional beanbag juggling balls [21, 20], as they do not deform. As successfully described in prior toss juggling approaches [24, 23, 21, 20], the funnel-shaped end-effector passively compensates for minor differences in ball trajectories by sliding the balls from the edge of the 170mm opening to the center. The second ball is released via a launching mechanism 3m above the floor, shown in Figure 2a, to achieve consistent height, velocity, and timing. This mechanism releases the ball by pushing the ball to an opening using a piston attached to a pulley system. The release of the ball is detected using Optitrack to start the episode.

5.2 Simulation Studies

The initial validation of the proposed learning system is performed in MuJoCo to evaluate the convergence of different numbers of roll-outs per episode over seeds. Figure 3a shows the juggling duration distribution of the final policy averaged over 60 different seeds at 10, 25 and 50 roll-outs per episode. For 10 roll-outs per episode, the learning system frequently converges to a sub-optimal final policy, which does not achieve consistent juggling of 10 seconds. The probability of converging to a good policy is the highest for 25 roll-outs per episode and hence, we use 25 roll-outs per episode on the physical system. It is important to point out, that learning juggling in the simulation is actually more challenging compared to the real world as the stabilizing passive dynamics of the system could not be modeled accurately.

5.3 Learning on the real Barrett WAM

For the learning on the physical Barrett WAM 20 episodes were performed. During each episode 25 randomly sampled parameters were executed and the episodic reward evaluated. If the robot successfully juggles for 10s, the roll-out is stopped. Roll-outs that were corrupted due to obvious environment errors were repeated using the same parameters. Minor variations caused by the environment initialization were not repeated. After collecting the samples, the policy was updated using eREPS with a KL constraint of 2. The learning progress is shown in Figure 4. Initially, the robot on average achieves between 1 to 3 repeated catches. These initial catches are important as the robot

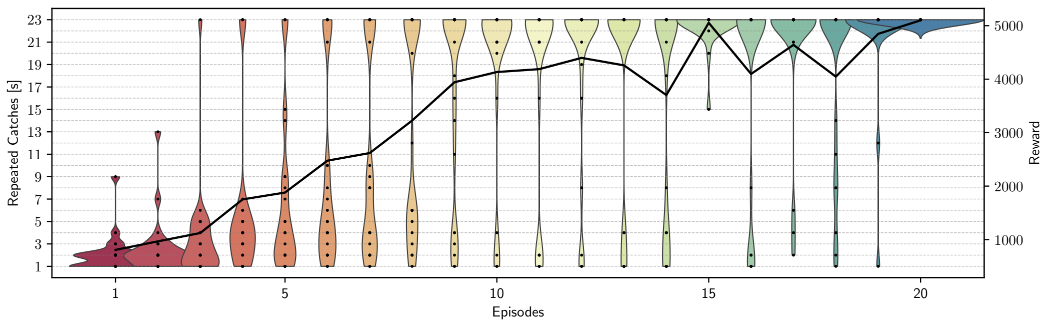


Figure 4: Reward distributions while learning to juggle on the physical system. Each point corresponds to a single roll-out on the system. Starting from the expert initialization, which only achieves juggling for a few repeated catches, the robot continuously improves using only binary rewards. After 56 minutes of training, the final policy achieves consistent juggling for more than 10s.

would otherwise not receive any meaningful information. Therefore, the rudimentary expert initialization must achieve some repeated catches to ensure fast learning. After the initial episodes, the robot rapidly improves the juggling duration. Starting from episode 10, the robot achieves consistent juggling of 10 seconds and only very few balls are dropped during the start. During the next 10 episodes, the average reward oscillates as the number of dropped balls varies but the robot achieves successful completion for the other trials. At episode 20 the robot achieves perfect juggling of all 25 randomly sampled parameters. Videos documenting the learning process and the evaluation can be found at **https://sites.google.com/view/jugglingbot**.

To test the repeatability and stability of the learned policy, the deterministic policy mean of episode 20 is executed for 30 repeated roll-outs with a maximum duration of 120 seconds. The achieved performance is compared to a hand-tuned policy in Figure 3b. Averaging at 106.82s, the learned policy performs significantly better compared to the hand-tuned policy with 66.51s. The weaker performance of the hand-tuned policy within the stroke-based initiation movement matches our expectations as tuning the stroke-based movement is the hardest part of the manual parameter tuning. Both policies do not achieve perfect repeatability due to the residual stochasticity of the environment.

To test the stability of the learned policy, the juggling was repeatedly executed and the maximum juggling duration recorded. The learned policy achieved juggling for 33 minutes, which corresponds to more than 4500 repeated catches on the second try, after 15 minutes on the first one. The high number of repeated catches, highlights the precision of the Barrett WAM, the end-effector design and both policies. Once the juggling is initiated successfully, the policies can recover from minor variations due to the passive stability induced by the end-effector design.

6 Conclusion

We described a robot learning system capable of learning toss juggling of two balls with a single anthropomorphic manipulator using only binary rewards. We demonstrated that our system can learn this high acceleration task within 56 minutes of experience, utilizing sufficient engineering and task knowledge designing the robot learning system. Starting from a rudimentary expert initialization, the system consistently improves until achieving repeated juggling of up to 33 minutes, which corresponds to more than 4500 repeated catches. Furthermore, the learned policy outperforms a hand-tuned policy in terms of repeatability and achieves significantly higher rewards average across 30 trials. In addition, we highlighted and discussed the incorporated engineering and task expertise to make learning on the physical system viable. This discussion should help future scientists and practitioners to address the needs of a physical system when building future robot learning systems. Nevertheless, this approach also pointed out the shortcomings of state-of-the-art robot learning approaches for learning dynamic tasks on the physical system. Despite the incorporated engineering and task knowledge the learning still takes up to 5 hours and hence, reiterates the necessity for more sample efficient representations and learning approaches for sparse and binary rewards.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No #640554 (SKILLS4ROBOTS). Furthermore, this research was also supported by grants from ABB and NVIDIA.

References

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[2] S. Fujimoto, H. Van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[3] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

[4] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[5] P. Klink, H. Abdulsamad, B. Belousov, and J. Peters. Self-paced contextual reinforcement learning. *Conference on Robot Learning (CoRL)*, 2019.

[6] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami, M. Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[7] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[8] A. Nagabandi, K. Konoglie, S. Levine, and V. Kumar. Deep dynamics models for learning dexterous manipulation. *arXiv preprint arXiv:1909.11652*, 2019.

[9] S. Schaal and C. G. Atkeson. Open loop stable control strategies for robot juggling. In *International Conference on Robotics and Automation (ICRA)*, 1993.

[10] S. Schaal and C. G. Atkeson. Robot juggling: An implementation of memory-based learning. 1994.

[11] S. Schaal and C. G. Atkeson. Robot juggling: implementation of memory-based learning. *IEEE Control Systems*, 1994.

[12] P. Reist and R. D'Andrea. Bouncing an unconstrained ball in three dimensions with a blind juggling robot. In *International conference on Robotics and Automation (ICRA)*, 2009.

[13] E. W. Aboaf, C. G. Atkeson, and D. J. Reinkensmeyer. Task-level robot learning. In *International Conference on Robotics and Automation (ICRA)*, 1988.

[14] E. W. Aboaf, S. M. Drucker, and C. G. Atkeson. Task-level robot learning: Juggling a tennis ball more accurately. In *International Conference on Robotics and Automation (ICRA)*, 1989.

[15] A. A. Rizzi, L. L. Whitcomb, and D. E. Koditschek. Distributed real-time control of a spatial robot juggler. *Computer*, 1992.

[16] A. A. Rizzi and D. E. Koditschek. Further progress in robot juggling: The spatial two-juggle. *Departmental Papers (ESE)*, 1993.

[17] S. Schaal, D. Sternad, and C. G. Atkeson. One-handed juggling: A dynamical approach to a rhythmic movement task. 1996.

[18] J. Kober, K. Mülling, O. Krömer, C. H. Lampert, B. Schölkopf, and J. Peters. Movement templates for learning of hitting and batting. In *International Conference on Robotics and Automation (ICRA)*, 2010.

[19] M. Müller, S. Lupashin, and R. D'Andrea. Quadrocopter ball juggling. In *International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[20] T. Sakaguchi, Y. Masutani, and F. Miyazaki. A study on juggling tasks. In *International Workshop on Intelligent Robots and Systems (IROS)*, 1991.

[21] T. Sakaguchi, M. Fujita, H. Watanabe, and F. Miyazaki. Motion planning and control for a robot performer. In *International Conference on Robotics and Automation (ICRA)*, 1993.

[22] T. Kizaki and A. Namiki. Two ball juggling with high-speed hand-arm and high-speed vision system. In *International Conference on Robotics and Automation (ICRA)*, 2012.

[23] M. Riley and C. G. Atkeson. Robot catching: Towards engaging human-humanoid interaction. *Autonomous Robots (AURO)*, 2002.

[24] J. Kober, M. Glisson, and M. Mistry. Playing catch and juggling with a humanoid robot. In *International Conference on Humanoid Robots (Humanoids)*, 2012.

[25] S. Schaal and D. Sternad. *Learning passive motor control strategies with genetic algorithms*, pages 913–918. Addison-Wesley, Redwood City, CA, 1993.

[26] L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE, 2016.

[27] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research (IJRR)*, 2018.

[28] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine. Residual reinforcement learning for robot control. In *International Conference on Robotics and Automation (ICRA)*, pages 6023–6029. IEEE, 2019.

[29] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[30] D. Schwab, T. Springenberg, M. F. Martins, T. Lampe, M. Neunert, A. Abdolmaleki, T. Herkweck, R. Hafner, F. Nori, and M. Riedmiller. Simultaneously learning vision and feature-based control policies for real-world ball-in-a-cup. *arXiv preprint arXiv:1902.04706*, 2019.

[31] G. Kahn, P. Abbeel, and S. Levine. Badgr: An autonomous self-supervised learning-based navigation system. *arXiv preprint arXiv:2002.05700*, 2020.

[32] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[33] D. Büchler, S. Guist, R. Calandra, V. Berenz, B. Schölkopf, and J. Peters. Learning to play table tennis from scratch using muscular robots. *arXiv preprint arXiv:2006.05935*, 2020.

[34] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems (NeuRIPS)*. 2007.

[35] P. Kormushev, S. Calinon, and D. G. Caldwell. Robot motor skill coordination with em-based reinforcement learning. In *nternational Conference on Intelligent Robots and Systems (IROS)*, 2010.

[36] J. Kober and J. R. Peters. Policy search for motor primitives in robotics. In *Advances in Neural Information Processing Systems (NeuRIPS)*, 2009.

[37] J. Kober, E. Öztop, and J. Peters. Reinforcement learning to adjust robot movements to new situations. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

[38] K. Mülling, J. Kober, O. Kroemer, and J. Peters. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research (IJRR)*, 2013.

[39] M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *International Conference on machine learning (ICML)*, 2011.

[40] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[41] M. P. Deisenroth, G. Neumann, J. Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2013.

[42] J. Peters, K. Mulling, and Y. Altun. Relative entropy policy search. In *Conference on Artificial Intelligence (AAAI)*, 2010.

[43] A. Abdolmaleki, B. Price, N. Lau, L. P. Reis, and G. Neumann. Deriving and improving cma-es with information geometric trust regions. In *Genetic and Evolutionary Computation Conference (GECCO)*, pages 657–664, 2017.

[44] S. M. Kakade. A natural policy gradient. In *Advances in neural information processing systems*, pages 1531–1538, 2002.

[45] J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

[46] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research (IJRR)*, 2013.

A KL Constrained Maximum Likelihood Optimization for eREPS

The standard eREPS formulation [41] solves the optimization problem described by

$$\pi_{k+1} = \arg\max_{\pi} \int \pi(\boldsymbol{\theta}) R(\boldsymbol{\theta}) d\boldsymbol{\theta}, \qquad \text{s.t.} \qquad d_{\mathrm{KL}}(\pi_{k+1} || \pi_k) \leq \epsilon \tag{1}$$

via computing the importance weights of each sample and fitting the policy to the weighted samples. The sample weights are described by

$$w_i = \frac{\exp(R(\boldsymbol{\theta}_i)/\eta^*)}{\sum_i \exp(R(\boldsymbol{\theta}_i)/\eta^*)} \qquad \text{with} \qquad \eta^* = \arg\min_{\eta} \eta\epsilon + \eta \log \sum_{i=0}^{N} \pi_k(\boldsymbol{\theta}_i) \exp\left(R(\boldsymbol{\theta}_i)/\eta\right).$$

The fitting of the policy to the weighted samples is described by

$$\boldsymbol{\mu}_{k+1},\ \boldsymbol{\Sigma}_{k+1} = \arg\max_{(\boldsymbol{\mu},\, \boldsymbol{\Sigma})} \sum_{i=0}^{N} w_i \log(\mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \boldsymbol{\theta}_i)).$$

In the case of Gaussian policies this optimization can be solved in closed form and the updated parameters are described by

$$\boldsymbol{\mu}_{k+1} = \sum_{i=0}^{N} w_i\, \boldsymbol{\theta}_i \qquad \boldsymbol{\Sigma}_{k+1} = \sum_{i=0}^{N} w_i\, \left(\boldsymbol{\theta}_i - \boldsymbol{\mu}_{k+1}\right)\left(\boldsymbol{\theta}_i - \boldsymbol{\mu}_{k+1}\right)^T.$$

For an in-depth derivation of this approach please refer to [41]. This approach to updating the policy works well if $N \gg n$ with the parameter dimensionality n. In this case the KL divergence between two consecutive parametrized policies is smaller than ϵ. If $N \approx n$ the KL divergence between two consecutive parametrized policies must not necessarily be smaller than ϵ. In this case the KL divergence between the weighted and unweighted samples is bounded by ϵ but the KL divergence between the parametrized policies is not. To ensure that the KL divergence between parametrized policies is bounded after the maximum likelihood optimization, we change the optimization problem to include a KL constraint. The constrained objective is described by

$$\boldsymbol{\mu}_{k+1},\ \boldsymbol{\Sigma}_{k+1} = \arg\max_{(\boldsymbol{\mu},\, \boldsymbol{\Sigma})} \sum_{i=0}^{N} w_i \log(\mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \boldsymbol{\theta}_i)) \qquad \text{s.t.} \qquad d_{\mathrm{KL}}(\pi_k || \pi_{k+1}) \leq \epsilon.$$

Please note that the order of the KL divergence is switched compared to Equation 1 and that the KL divergence is not symmetric. We switch from the I-projection to the M-projection because otherwise this optimization has no closed form solution for Gaussian policies. For bounding the distance between two consecutive policies the reverse KL divergence can be used as for small KL divergences both projections are comparable. The constrained optimization problem can be solved using Lagrangian multipliers as initially derived by [43]. The Lagrangian L for a Gaussian policy is described by

$$\begin{aligned}
L =& \sum_{i=0}^{N} w_i \log(\mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \boldsymbol{\theta}_i)) + \xi(\epsilon - d_{\mathrm{KL}}(\mathcal{N}(\boldsymbol{\mu}_k,\ \boldsymbol{\Sigma}_k)\ ||\ \mathcal{N}(\boldsymbol{\mu}_{k+1},\ \boldsymbol{\Sigma}_{k+1}))) \\
=& \frac{1}{2}\Bigg[-n\log(2\pi) - \log(|\boldsymbol{\Sigma}_{k+1}|) - \sum_{i=0}^{N} w_i \left(\boldsymbol{\theta}_i - \boldsymbol{\mu}_{k+1}\right)^T \boldsymbol{\Sigma}_{k+1}^{-1} \left(\boldsymbol{\theta}_i - \boldsymbol{\mu}_{k+1}\right) \Bigg] \\
& + \frac{\xi}{2}\Bigg[2\epsilon - \mathrm{tr}\left(\boldsymbol{\Sigma}_{k+1}^{-1}\boldsymbol{\Sigma}_k\right) - n + \log\left(\frac{\boldsymbol{\Sigma}_{k+1}}{\boldsymbol{\Sigma}_k}\right) + \left(\boldsymbol{\mu}_{k+1} - \boldsymbol{\mu}_k\right)^T \boldsymbol{\Sigma}_{k+1}^{-1} \left(\boldsymbol{\mu}_{k+1} - \boldsymbol{\mu}_k\right) \Bigg].
\end{aligned}$$

The updates for the mean and covariance can be computed in closed form by setting $\nabla_{\mu_{k+1}} L \coloneqq 0$ and $\nabla_{\Sigma_{k+1}} L \coloneqq 0$, i.e.,

$$\nabla_{\mu_{t+1}} L = \boldsymbol{\Sigma}_{k+1}^{-1} \left[\sum_{i=1}^{N} w_i\, \boldsymbol{\theta} + \xi\, \boldsymbol{\mu}_k - (\xi + 1)\, \boldsymbol{\mu}_{k+1} \right] \coloneqq 0$$

$$\nabla_{\Sigma_{t+1}} L = \boldsymbol{\Sigma}_{k+1}^{-1} \left(\boldsymbol{\Sigma}_s + \xi \boldsymbol{\Sigma}_k + \xi \left(\boldsymbol{\mu}_{k+1} - \boldsymbol{\mu}_k\right)\left(\boldsymbol{\mu}_{k+1} - \boldsymbol{\mu}_k\right)^T - (\xi + 1)\ \boldsymbol{\Sigma}_{t+1} \right) \boldsymbol{\Sigma}_{k+1}^{-1} \coloneqq 0$$

The resulting update rule for the mean and covariance is described by

$$\boldsymbol{\mu}_{k+1} = \frac{\xi^* \boldsymbol{\mu}_k + \boldsymbol{\mu}_s}{1 + \xi^*}, \qquad \boldsymbol{\Sigma}_{k+1} = \frac{\boldsymbol{\Sigma}_s + \xi^* \boldsymbol{\Sigma}_k + \xi^* \left(\boldsymbol{\mu}_{k+1} - \boldsymbol{\mu}_k\right)\left(\boldsymbol{\mu}_{k+1} - \boldsymbol{\mu}_k\right)^T}{1 + \eta^*}$$

$$\boldsymbol{\mu}_s = \sum_{i=0}^{N} w_i \, \boldsymbol{\theta}_i \qquad \boldsymbol{\Sigma}_s = \sum_{i=0}^{N} w_i \, \left(\boldsymbol{\theta}_i - \boldsymbol{\mu}_{k+1}\right)\left(\boldsymbol{\theta}_i - \boldsymbol{\mu}_{k+1}\right)^T$$

with the optimal multiplier ξ^*. The optimal multiplier cannot be obtained in closed form as $\boldsymbol{\mu}_{k+1}$ and $\boldsymbol{\Sigma}_{k+1}$ depend on ξ. Hence, ξ^* must be obtained by solving

$$\begin{aligned} \xi^* = \arg\max_{\xi} & \left[- \log(|\boldsymbol{\Sigma}_{t+1}|) - \sum_{i=0}^{N} w_i \left(\boldsymbol{\theta}_i - \boldsymbol{\mu}_{k+1}\right)^T \boldsymbol{\Sigma}_{k+1}^{-1} \left(\boldsymbol{\theta}_i - \boldsymbol{\mu}_{k+1}\right) \right] \\ & + \xi \left[2\epsilon - \mathrm{tr}\left(\boldsymbol{\Sigma}_{k+1}^{-1} \boldsymbol{\Sigma}_k\right) - n + \log\left(\frac{\boldsymbol{\Sigma}_{k+1}}{\boldsymbol{\Sigma}_k}\right) + \left(\boldsymbol{\mu}_{k+1} - \boldsymbol{\mu}_k\right)^T \boldsymbol{\Sigma}_{k+1}^{-1} \left(\boldsymbol{\mu}_{k+1} - \boldsymbol{\mu}_k\right) \right] \end{aligned}$$

with gradient based optimization.