2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

TrueÆdapt: Learning Smooth Online Trajectory Adaptation with Bounded Jerk, Acceleration and Velocity in Joint Space

Jonas C. Kiemel[1]*, Robin Weitemeyer[1]*, Pascal Meißner[1] and Torsten Kröger

***Abstract*— We present TrueÆdapt, a model-free method to learn online adaptations of robot trajectories based on their effects on the environment. Given sensory feedback and future waypoints of the original trajectory, a neural network is trained to predict joint accelerations at regular intervals. The adapted trajectory is generated by linear interpolation of the predicted accelerations, leading to continuously differentiable joint velocities and positions. Bounded jerks, accelerations and velocities are guaranteed by calculating the range of valid accelerations at each decision step and clipping the network's output accordingly. A deviation penalty during the training process causes the adapted trajectory to follow the original one. Smooth movements are encouraged by penalizing high accelerations and jerks. We evaluate our approach by training a simulated KUKA iiwa robot to balance a ball on a plate while moving and demonstrate that the balancing policy can be directly transferred to a real robot.**

I. INTRODUCTION

Robots frequently interact with their environment while executing movements. Industrial applications include spray painting, welding, bonding or grinding. In service robotics, an illustrative use-case is a waiter robot trying to transport glasses on a tray without spilling water.

If the behaviour of the environment is precisely known in advance, motion planning can be performed offline. However, imperfect environment models or unforeseen external disturbances may cause the initial motion plan to fail. For instance, welding distortion might be hard to predict, elastic components might cause problems during bonding and the grinding behaviour might alter over time due to wear of the abrasives. Reacting to unpredictable disturbances typically implies online adaptation of the initially planned trajectory. Designing a model-based control system for smooth trajectory adaptation in task space is challenging, especially if the robot is required to work near to kinematic singularities or close to the velocity limits of its joints to meet time requirements.

With TrueÆdapt, we replace the need for a plant model by learning how to adapt trajectories from simulated experiences using model-free reinforcement learning. Kinematic singularities do not cause problems as the algorithm works in joint space. Bounded and continuously differentiable joint velocities are guaranteed and smooth adaptations are favoured since jerky movements are punished during training. We demonstrate successful sim-to-real transfer for a dynamic balancing task, which is motivated by the aforementioned job of a waiter robot. Like in the industrial applications mentioned above, the environment is directly influenced by the movements of the robot. However, the balancing task does not alter the environment permanently, which facilitates quantitative evaluation of the real-world performance. In addition, state-of-the-art physic engines allow fast simulation, making the task attractive for research on sim-to-real transfer.

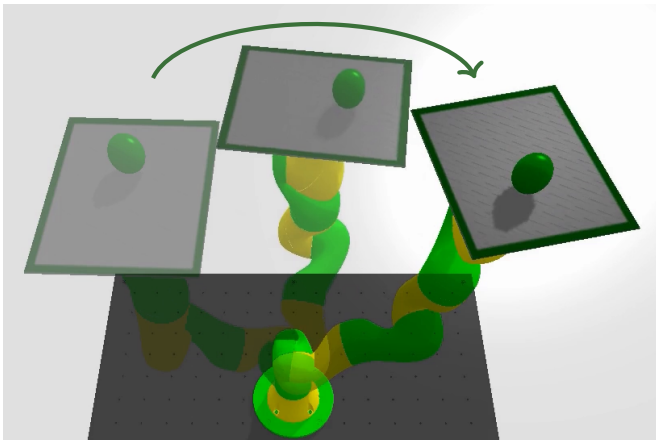


Fig. 1: TrueÆdapt applied to a balancing task: The robot has learned to keep a ball at the same spot on a plate while moving along a reference trajectory.

II. RELATED WORK

A. Trajectory Generation

With sampling-based motion planners [1], finding a suitable robot trajectory is typically split in two distinct phases [2]. Firstly, a collision-free geometric path is generated. Secondly, timestamps are added to the waypoints of the path, leading to a time-parameterized trajectory. The approach assumes that an appropriate path can be found without taking the timing of the movement into account. Although this assumption is not fulfilled for dynamic tasks like balancing, sampling-based motion planner can be used to generate reference trajectories for TrueÆdapt. In [3], a method for time-optimal online trajectory generation with bounded jerk and acceleration is presented. For offline scenarios, time-optimal trajectory parameterization can be performed considering both kinematic [2] and dynamic joint constraints [4].

B. Reinforcement Learning in Robotics

In recent years, reinforcement learning (RL) has been applied to a variety of robotic applications like locomotion [5], [6], grasping [7], [8] or dexterous manipulation [9], [10].

[1]Institute for Anthropomatics and Robotics – Intelligent Process Automation and Robotics (IAR-IPR), Karlsruhe Institute of Technology (KIT), jonas.kiemel@kit.edu
* These authors contributed equally.

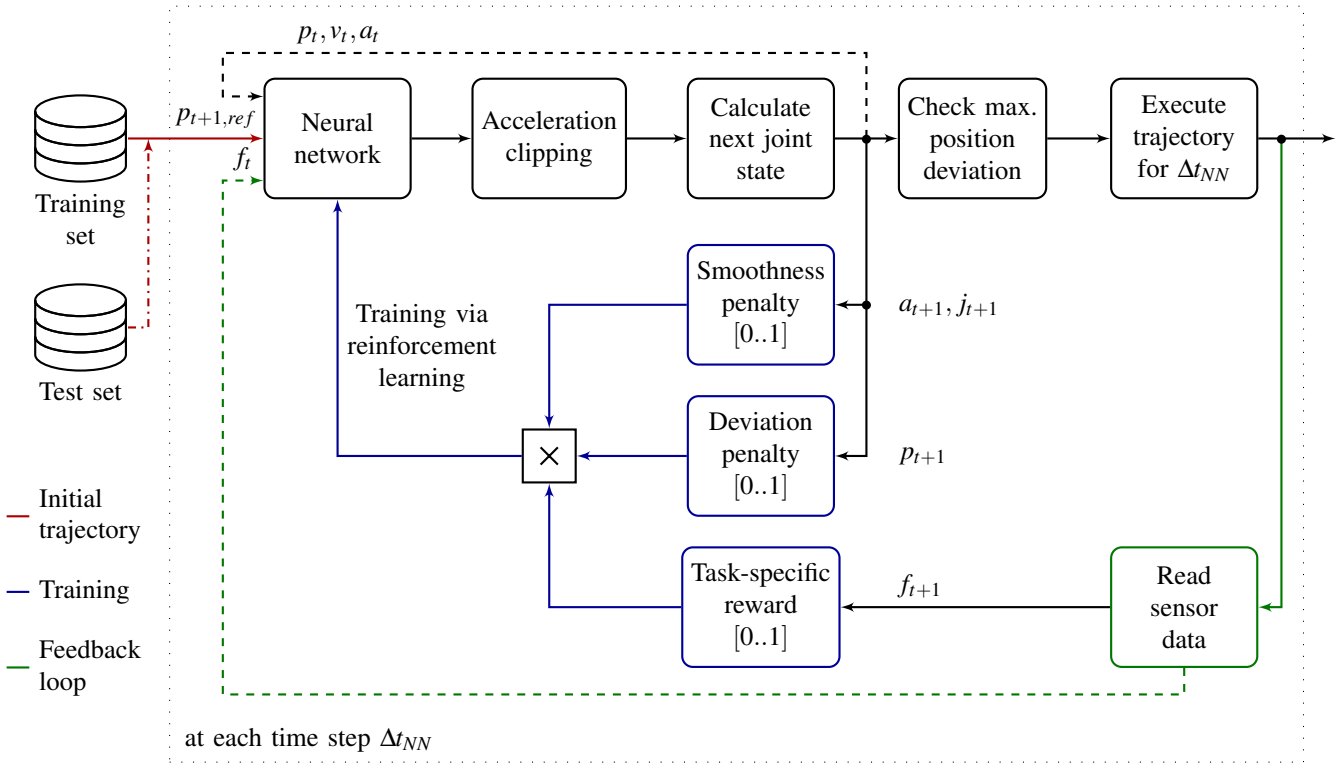


Fig. 2: System components to learn online adaptations with TrueÆdapt. Δt_{NN} is the time span between network predictions.

An RL-based method to smoothly track a jerky reference path with an industrial robot in the presence of unknown dynamical constraints is presented in [11]. The authors train a neural network to predict joint velocities and penalize the chosen action based on the distance to the reference path. In contrast, we predict joint accelerations to ensure continuously differentiable joint velocities and use a time-parameterized trajectory as reference. In [12], movements are learnt with a real robot by mapping a camera image directly to motor torques. However, when training in simulation, a very accurate dynamic model is required to generate meaningful torque commands for a real robot.

C. Sim-to-real Transfer

Generating sufficient training data for model-free RL-algorithms with real robots is costly and time-consuming. Conducting training in simulation is an appealing and widely used alternative. However, transfer from simulation to the real world typically leads to a drop in performance. One approach to bridge the so-called reality gap is randomization of the simulation to learn a robust policy. Domain randomization can be applied to simulated images [13] as well as to physical parameters like friction or damping [14]. Making the simulation more realistic is another way to improve sim-to-real transfer. In [15], generative adversarial networks are trained to make synthetic renderings look like real images, whereas [6] incorporates an accurate actuator model and sensor latency to improve the simulation fidelity.

III. System Overview

The most important system components of TrueÆdapt are shown in Fig. 2. A neural network predicts joint accelerations based on sensory feedback, the current state of the joints and the following positions of a reference trajectory. The predicted accelerations are clipped to ensure that jerk, acceleration and velocity limits are not violated. In addition, the adapted trajectory is not executed if the adapted point deviates too much from the reference trajectory, thereby avoiding self-collision and violation of position limits. During training, a smoothness penalty penalizes jerky movements, while a deviation penalty ensures that the adapted trajectory follows the original one. A task-specific reward makes the system learn the intended task like balancing a ball. Details on each step will be explained in the following sections.

IV. Generation of Reference Trajectories

Suitable reference trajectories for TrueÆdapt should follow the desired path of the movement, whereas dynamic interactions with the environment do not have to be considered. Our procedure to generate reference trajectories is illustrated in Fig. 3. As a first step, Cartesian waypoints are sampled randomly within predefined areas. Spline interpolation is used to produce a smooth Cartesian path. After converting the path to joint space via inverse kinematics, time-optimal trajectory parameterization is performed with a method described in [2]. As a final step, the trajectory is uniformly sampled using the time span between network predictions Δt_{NN}, which we choose to be 50 ms for our experiments.

Each trajectory is assigned either to the training set or to the test set.

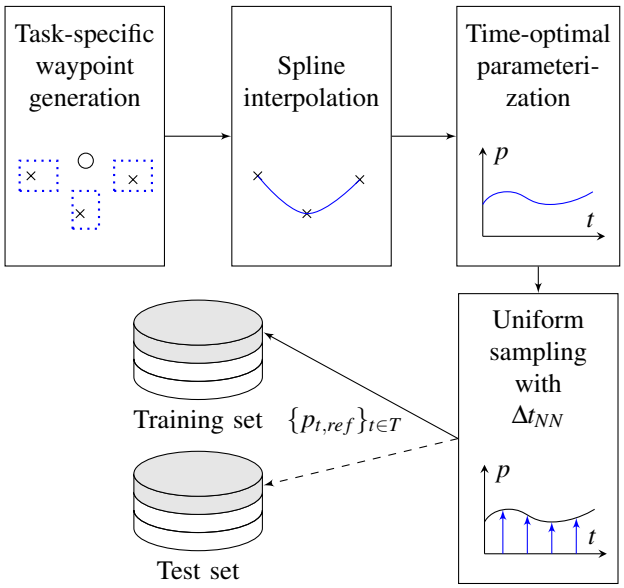


Fig. 3: Generation of reference trajectories.

We note, that an offline method like [16] can be used to generate appropriate trajectories without the need to define task-specific sampling areas.

V. Learning Online Trajectory Adaptations

A. Objectives

We define the following objectives for our online trajectory adaptation approach:

- The primary goal is to accomplish the specified task (e.g. balancing a ball).
- The adapted trajectory should stay close to the original one.
- Jerk, acceleration, velocity and position limits of the joints should not be violated.
- Self-collision should be avoided.
- The adapted trajectory should be smooth.

B. Formalization

The learning problem is formalized as a Markov Decision Process $(\mathcal{S}, \mathcal{A}, R_{\underline{a}})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space and $R_{\underline{a}}$ is the immediate reward due to action $\underline{a}$. We use model-free RL for training a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ to maximize the expected sum of future rewards. Each element of $s \in \mathcal{S}$ and $\underline{a} \in \mathcal{A}$ is normalized to be in the range of [-1, 1]. Decisions are made in real-time during motion with a cycle time of Δt_{NN}.

1) State Definition: The state s_t consists of the current joint position p_t, velocity v_t and acceleration a_t as well as sensory feedback f_t and N future positions of the reference trajectory $p_{t+\{1..N\},ref}$. Instead of using measured values for p_t, v_t and a_t, we use the setpoints from the previous calculation step, thereby avoiding sensor noise and latency. The results of ablation studies to identify the influence of each part of the state can be found in TABLE I.

2) Action Definition: The action $\underline{a_t}$ determines a_{t+1}, the normalized angular acceleration for each robot joint at the beginning of the next time step. Jerk, acceleration and velocity limits are respected by clipping the predicted acceleration a_{t+1} accordingly. Linear interpolation between a_t and a_{t+1} is performed to produce continuous accelerations within the current time step. Intermediate setpoints for a position controller are generated by integrating the accelerations twice. We note that the movements of an untrained agent are not influenced by the selected reference trajectory. Instead, the network learns to stay close to the reference trajectory because of a deviation penalty. With our approach, the execution times of the adapted trajectory and the reference trajectory are identical.

3) Reward Definition: The reward per decision step $R_{\underline{a}} \in [0,1]$ is calculated by multiplying a task-specific reward $R_T \in [0,1]$ with a smoothness penalty $P_S \in [0,1]$ and a deviation penalty $P_D \in [0,1]$.

$$R_{\underline{a}} = R_T \cdot (1 - P_S) \cdot (1 - P_D) \tag{1}$$

The smoothness penalty P_S is composed of an acceleration penalty $P_A \in [0,1]$ and a jerk penalty $P_J \in [0,1]$.

$$P_S = \frac{P_A + P_J}{2} \tag{2}$$

P_A penalizes accelerations that are higher than a user-defined threshold a_{th}. In the following equation, $a_{abs} \in [0,1]$ is the highest absolute value in a_{t+1}.

$$P_A = \begin{cases} 0 & a_{abs} \in [0,\ a_{th}) \\ \left(1 - \dfrac{1 - a_{abs}}{1 - a_{th}}\right)^2 & a_{abs} \in [a_{th},\ 1] \end{cases} \tag{3}$$

The following definition of P_J is inspired by [17]. N_J corresponds to the number of joint. $j_{abs,\,t+1,\,i}$ is the unnormalized absolute jerk of joint i, while $j_{abs,\,max,\,i}$ is the unnormalized jerk limit. c is a user-defined weighting factor.

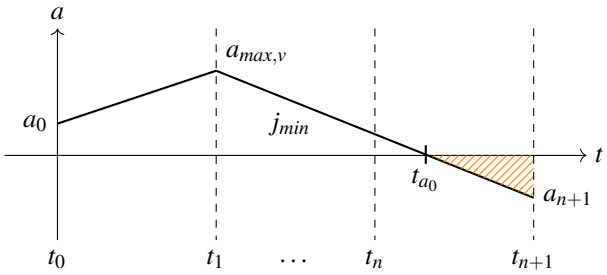
$$j_p = \sum_{i=1}^{N_J} \left(j_{abs,\,t+1,\,i}\right)^2 \tag{4}$$

$$j_{sat} = \frac{1}{c} \cdot \sum_{i=1}^{N_J} \left(j_{abs,\,max,\,i}\right)^2 \tag{5}$$

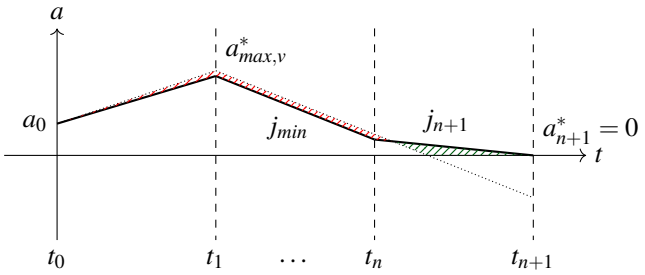
$$P_J = \begin{cases} \left(\dfrac{j_p}{j_{sat}}\right)^2 & j_p \in [0,\ j_{sat}] \\ 1 & j_p > j_{sat} \end{cases} \tag{6}$$

The deviation penalty P_D ensures that the adapted trajectory stays close to the reference. Δp_{max} is the greatest absolute joint position deviation between p_{t+1} and $p_{t+1,ref}$, while Δp_l and Δp_h are thresholds that lead to a punishment of 0 and 1, respectively.

$$P_D = \begin{cases} 0 & \Delta p_{max} \in [0,\ \Delta p_l) \\ \left(\dfrac{\Delta p_{max} - \Delta p_l}{\Delta p_h - \Delta p_l}\right)^2 & \Delta p_{max} \in [\Delta p_l,\ \Delta p_h] \\ 1 & \Delta p_{max} > \Delta p_h \end{cases} \tag{7}$$



(a) The maximum acceleration $a_{max,v}$ is followed by a deceleration with j_{min}. The maximum velocity v_{max} is reached at t_{a_0}.



(b) If the acceleration a^*_{n+1} is limited, $a^*_{max,v}$ is shifted in such a way that the areas hatched in red and green are of equal size.

Fig. 4: Consideration of velocity limitations.

4) Termination: A training episode terminates if the angular deviation between p_{t+1} and $p_{t+1,\,ref}$ exceeds a fixed threshold for at least one joint. The termination serves a dual purpose: Firstly, the system learns to stay close to the reference as termination leads to a smaller sum of rewards. Secondly, violation of position limits as well as self-collision are avoided, provided that the reference trajectory maintains a certain safety distance.

C. Implementation

We use a fully-connected neural network with SELU activations [18] and two hidden layers of size [256, 128] to map states to actions. The training process is performed in parallel using the Ray framework [19] and reference implementations provided by RLlib [20]. Because of its stability and reliability, the on-policy algorithm PPO [21] is chosen for training. The batch size is set to 2^{14}.

VI. Consideration of Joint Limitations

When executing online adaptations with a real robot, joint limitations have to be considered to avoid permanent damage to the robot joints. The basic idea of our approach is to calculate for each joint i and at each decision step the acceleration range $[a_{min,i}\ ,\ a_{max,i}]$ that does not lead to a violation of joint limits. As analytical expressions can be derived, the calculation can be done in real-time. Computing the range of valid accelerations for seven joints took at most 0.9 ms with an Intel i9-9900K CPU. Once the valid range is known, adapting the network prediction $\underline{a_t}$ is straightforward:

$$a_{t+1,\,i} = \begin{cases} a_{min,\,i} & \underline{a_{t,\,i}} < a_{min,\,i} \\ \underline{a_{t,\,i}} & \underline{a_{t,\,i}} \in [a_{min,\,i},\ a_{max,\,i}] \\ a_{max,\,i} & \underline{a_{t,\,i}} > a_{max,\,i} \end{cases} \tag{8}$$

$a_{max,\,i}$ is the normalized minimum value of $a_{max,j}$, $a_{max,a}$ and $a_{max,v}$, which are defined in the following. Equations for $a_{min,\,i}$ can be derived correspondingly.

A. Jerk Limitation

Given that the jerk is constant within each control cycle, the maximum valid acceleration can be computed as follows:

$$a_{max,j} = a_0 + j_{max} \cdot \Delta t_{NN} \tag{9}$$

We note that the linear interpolation of accelerations naturally limits jerk to:

$$j_{max,\,interpolation} = \frac{a_{max} - a_{min}}{\Delta t_{NN}} \tag{10}$$

B. Acceleration Limitation

Restricting accelerations is trivial as the range of valid accelerations corresponds to the specified acceleration limits.

$$a_{max,a} = a_{max} \tag{11}$$

C. Velocity Limitation

To guarantee bounded velocities, it is no longer sufficient to consider the next time step only. When working close to the velocity limit at a high acceleration, there might be no way to stay within the permitted velocity range without violating jerk limitations. Our approach prevents the robot from getting in such a situation. Fig. 4 illustrates the main idea. The maximum acceleration at the next time step $a_{max,v}$ must be followed by a deceleration with j_{min}. In addition, $a_{max,v}$ has to be chosen in such a way that the maximum velocity is reached at zero acceleration. For $v_0 + \frac{a_0 \cdot \Delta t_{NN}}{2} < v_{max}$, the following formula can be derived

$$a_{max,v} = \frac{j_{min} \cdot \Delta t_{NN}}{2} \cdot \left(1 - \sqrt{1 + \frac{8 \cdot (v_0 - v_{max}) + 4 \cdot a_0 \cdot \Delta t_{NN}}{j_{min} \cdot \Delta t_{NN}^2}}\right), \tag{12}$$

whereas

$$a_{max,v} = a_0 \cdot \left(1 - \frac{1}{2} \cdot \frac{a_0 \cdot \Delta t_{NN}}{v_{max} - v_0}\right) \tag{13}$$

applies for $v_0 + \frac{a_0 \cdot \Delta t_{NN}}{2} \geq v_{max}$.

The approach described above can cause oscillations, as the velocity does not necessarily reach its maximum value at a discrete decision step. In Fig. 4a, the area hatched in orange indicates the difference between v_{max} and the velocity at the next discrete decision step v_{n+1}. The problem can be mitigated by shifting $a_{max,v}$, as shown in Fig. 4b.

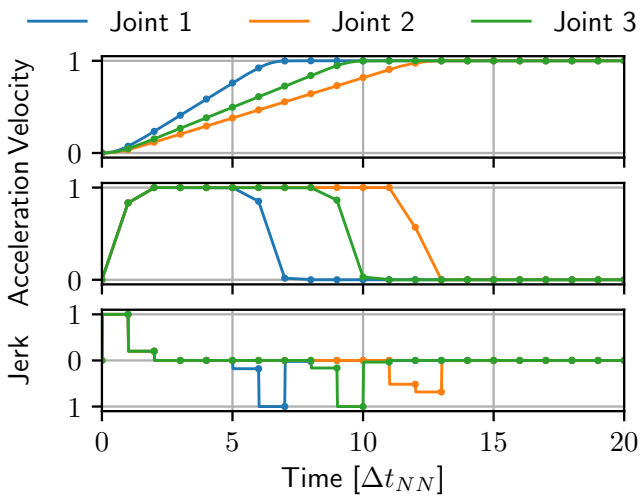


Fig. 5: Jerk, acceleration and velocity limitation when choosing the maximum valid acceleration at each decision step.

D. Validation

We validated our approach by running tests with over 100 000 simulated trajectories without exceeding the maximum velocities, accelerations and jerks. Fig. 5 illustrates the system behavior if the maximum acceleration is chosen at each decision step. As expected, the acceleration is first restricted due to jerk constraints, followed by acceleration and velocity limitations. In Fig. 6 random accelerations are sampled from the calculated range of valid accelerations. The figure shows that smooth velocities are generated and that the joint limits are not violated.

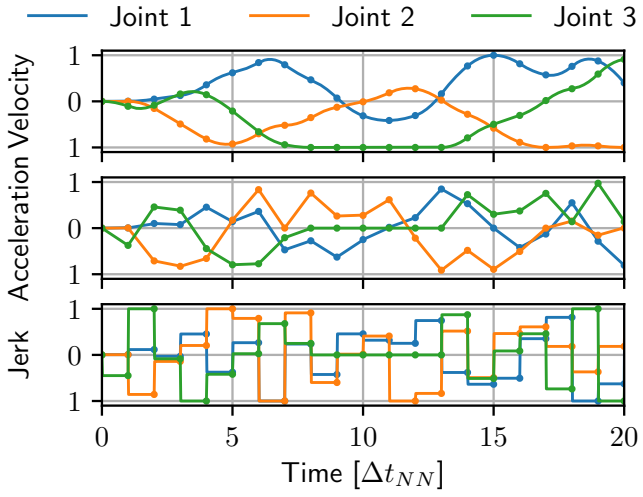


Fig. 6: Exemplary trajectory when choosing random accelerations within the range of valid accelerations.

VII. Experimental Setup

We evaluated our approach with two versions of a dynamic ball-on-plate task performed by a KUKA iiwa robot with seven degrees of freedom. While the basic task is to balance a ball on a plate during motion, the first version allows the ball to move within a large area of the plate ("on plate"). In contrast, the second version tries to keep the ball as close as possible to its initial position ("in place"). Fig. 9 shows how the task-specific reward is defined for both cases. The second version is related to traditional control tasks as there is one fixed setpoint for the ball position.

A. Reference Trajectories

The training dataset consists of 150 000 reference trajectories at different heights with sampling areas like those shown in Fig. 7. For reasons of symmetry, each trajectory can be mirrored along two planes, leading to a total of 600 000 trajectories.

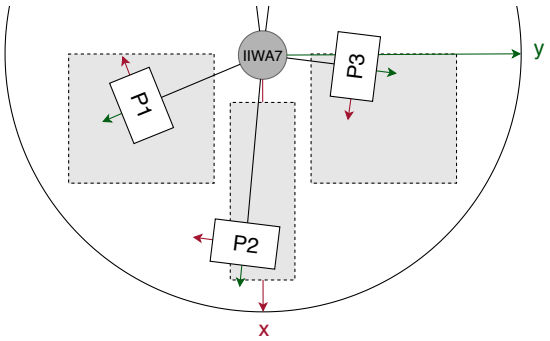


Fig. 7: Top view on the sampling areas to generate waypoints for an exemplary balancing task.

B. Sensory Feedback

Feedback on the task execution is given by adding the current and the last ball position to the state. For the "in place" task, we additionally include the two-dimensional distance to the initial ball position, which serves as a measure of the control error.

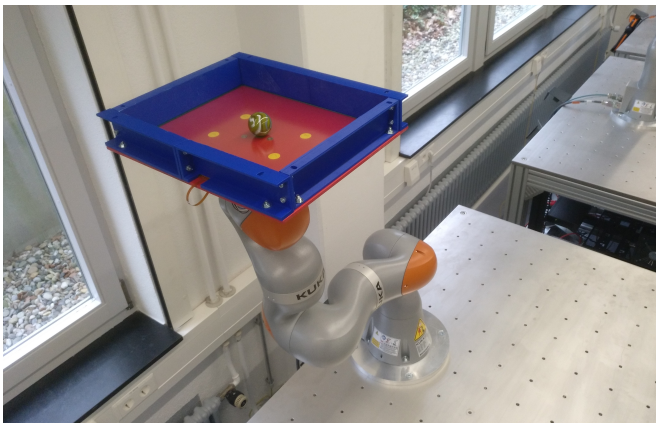
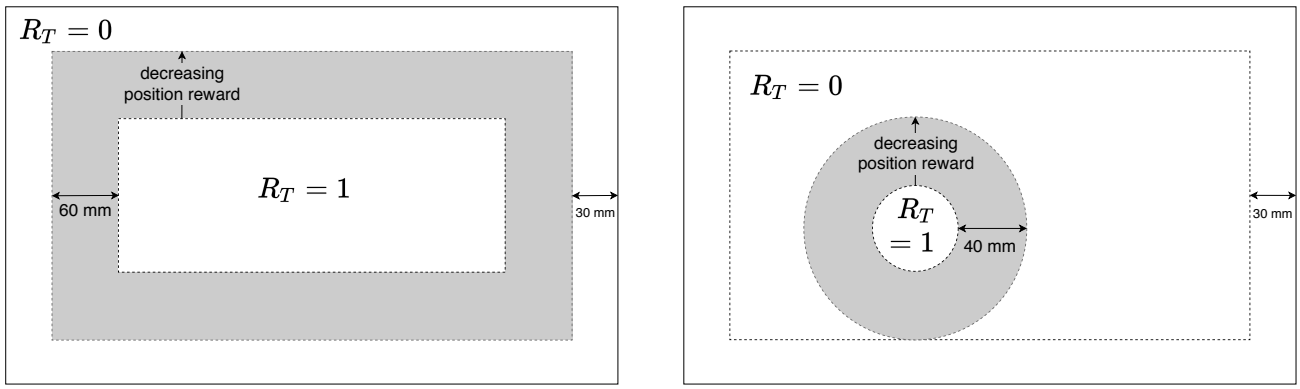


Fig. 8: Real-world setup for sim-to-real transfer.

C. Physics Simulation

The physics engine PyBullet [22] is used to generate training data in simulation. With the aim to learn a robust policy, we randomize the ball characteristics (mass, friction, radius) and model the measuring error of the ball position by adding noise to the corresponding signal.



(a) On plate (b) In place

Fig. 9: Definition of the task-specific reward R_T for two versions of a ball-on-plate task.

	Setting	Success rate	Trajectory fraction	Error distance	Acceleration	Jerk
On plate	Reference trajectories (no adaptations)	4.2 %	40.4 %	-	1.7 %	0.5 %
	TrueÆdapt: test set	89.6 %	97.6 %	-	7.0 %	3.7 %
	TrueÆdapt: training set	90.7 %	97.9 %	-	7.0 %	3.7 %
	Open loop: evaluation only	8.9 %	53.4 %	-	7.0 %	3.6 %
	Open loop: training and evaluation	45.6 %	84.2 %	-	7.1 %	4.1 %
	State: no current position	0.1 %	30.8 %	-	7.4 %	4.5 %
	State: no current velocity	13.7 %	68.5 %	-	5.4 %	3.2 %
	State: no current acceleration	91.4 %	97.9 %	-	20.8 %	16.0 %
	State: no following positions	0.4 %	31.3 %	-	8.8 %	5.4 %
	State: ten following positions	92.9 %	98.3 %	-	20.2 %	15.5 %
	Punishment: no jerk penalty	87.5 %	96.7 %	-	23.0 %	18.1 %
	Punishment: no acceleration penalty	50.6 %	79.3 %	-	62.4 %	50.4 %
In place	Reference trajectories (no adaptations)	0.3 %	22.1 %	-	2.0 %	0.3 %
	TrueÆdapt: test set	98.6 %	99.7 %	1.2 cm	6.4 %	3.4 %
	TrueÆdapt: real robot	82.0 %	96.1 %	1.7 cm	6.9 %	3.6 %
	Open loop: evaluation only	34.3 %	73.0 %	2.4 cm	6.6 %	3.3 %
	Open loop: training and evaluation	61.7 %	91.7 %	1.6 cm	6.7 %	4.1 %

TABLE I: Average values of the success rate, the successful trajectory fraction and the distance to the initial ball position for different configurations. The mean of the normalized absolute accelerations and jerks is averaged over all joints.

D. Real Setup

A picture of the real setup is shown in Fig. 8. For real-world experiments, the current ball position is detected by a resistive touch panel with a size of 34 cm × 27 cm. The robot is controlled via position commands at a rate of 200 Hz.

VIII. Evaluation

We define two metrics to measure the performance of the task execution, namely the success rate and the successfully executed trajectory fraction. For the "on plate" task, a trajectory is considered as successful if the ball does not touch the border of the plate, while the "in place" task allows a deviation of at most 6 cm from the initial ball position. The performance of the "on plate" task is evaluated after 32 million training steps, whereas 220 million training steps were conducted for the "in place" task. To generate the performance metrics in simulation, 10 000 trajectories from the test set were executed. Real world performance was evaluated with 50 trajectories and five different initial ball positions as indicated by the yellow spots in Fig. 8. For the "on plate" task, a trajectory fraction of 97.6 % and a success rate of 89.6 % was achieved. The "in place" task accomplished a trajectory fraction of 99.7 % and a success rate of 98.6 %. Transferring the policy to a real robot led to a a trajectory fraction of 96.1 % and a success rate of 82.0 %.

A. Ablation Studies

Ablation studies were performed to analyze the influence of individual system components. The results are listed in TABLE I. As expected, the network was not able to learn the task when omitting the current position or the next position of the reference trajectory from the state. Poor performance was achieved when omitting the current velocity. Our experiments show that the acceleration penalty is crucial for successful task execution. Without the penalty, jerky movements are produced, making it potentially harder to control the ball. The jerk penalty further improves the smoothness of the generated trajectories. Adding more than one future reference position to the state had a marginal impact on the performance. However, having access to more points might be crucial for tasks with a longer planning horizon.

B. Importance of Sensory Feedback

To assess the importance of closed loop feedback, we analyzed the performance of a network trained with sensory

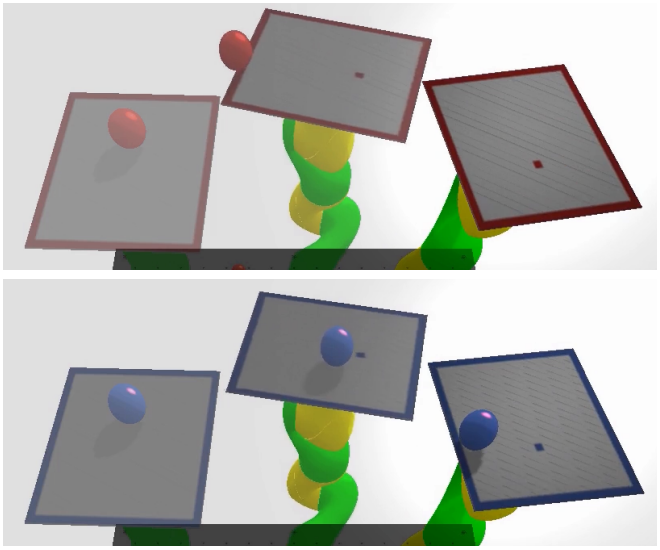


Fig. 10: Exemplary trajectory execution. Top: The reference trajectory fails to keep the ball on the plate. Bottom: When adapting the trajectory without updating the sensor signals, the ball stays on the plate but not at the desired spot.

feedback without updating the sensor signals during evaluation (open loop). For the "in place" task the success rate dropped from 98.6 % to 34.3 %, showing that the feedback is essential for the network. An exemplary rollout is illustrated in Fig. 10. Training a policy from scratch without sensory feedback led to a success rate of 61.7 %. We conclude that the network has, to a certain extent, learned to anticipate future movements of the ball.

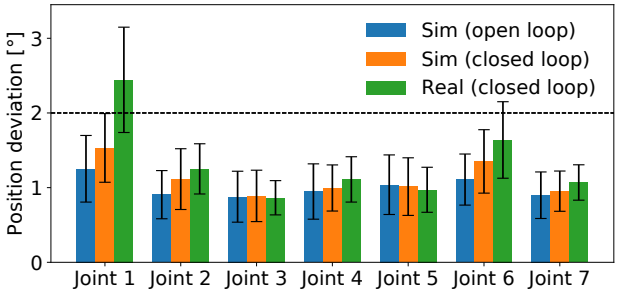


Fig. 11: Mean position deviation for the "in place" task. The lower threshold of the deviation penalty Δp_l was set to 2°.

C. Deviation to the Reference Trajectory

Fig. 11 shows the mean position deviation from the reference trajectory for the "in place" task. On average, all joints stay close to their reference. During real-world execution, stronger adaptations are predicted, especially for joint 1 and joint 6. This appears reasonable as the movement of the ball is harder to predict if the target domain differs from the training domain.

D. Sim-to-Real Transfer

Assuming that the actual joint positions closely follow their setpoints, we use setpoints instead of actual values for the robot state. Fig. 12 visualizes the tracking accuracy of the trajectory controller in simulation and in the real world. During fast movements a small delay can be noticed. However, as the delay appears in both simulated and real data, the policy can learn to cope with it.

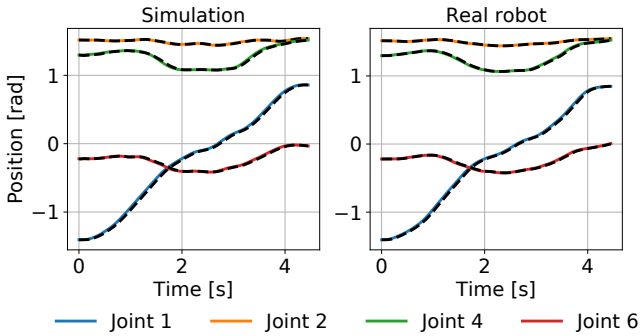


Fig. 12: Tracking performance of the trajectory controller in simulation and in the real world. Setpoints are shown as solid lines. Actual values are represented by dashed black lines.

Fig. 13 shows a successful rollout of the "in place" task for both simulation and real execution.

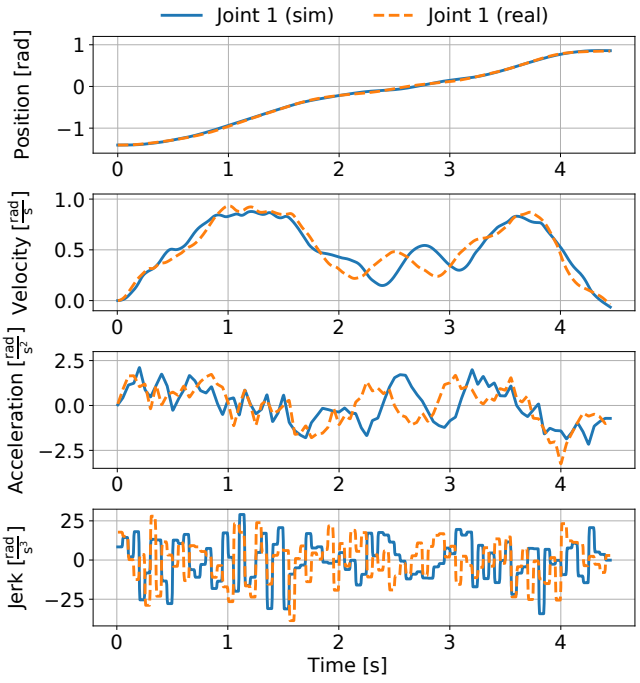


Fig. 13: Comparison of simulated and real setpoints for an exemplary trajectory execution of the "in place" task.

As shown in the accompanying video, the sim-to-real transfer could be successfully conducted with various balls, differing in mass, size and material.

IX. CONCLUSIONS

We presented a real-time capable approach for learning online adaptations based on sensory feedback and a method to ensure that the jerks, accelerations and velocities of the adapted trajectories are bounded. The effectiveness of our approach was demonstrated by learning to balance a ball on a plate while moving. The policy was trained in simulation and successfully transferred to a real robot. The evaluation showed that the adapted trajectories stay close to their reference and that sensory feedback is crucial for successful task execution. In future work, we intend to analyze the performance of our approach for tasks that require further deviation from the reference trajectory. In addition, we aim to develop a more sophisticated method for avoiding position limits and self-collision during motion.

ACKNOWLEDGMENT

This research was supported by the German Federal Ministry of Education and Research (BMBF) and the Indo-German Science & Technology Centre (IGSTC) as part of the project TransLearn (01DQ19007A). We would like to thank Tamim Asfour for his valuable input and helpful advice. All real-world experiments were performed at the KUKA Robot Learning Lab at KIT [23]. Special thanks to Wolfgang Wiedmeyer for his tremendous effort in building up the lab.

REFERENCES

[1] R. Geraerts and M. H. Overmars, "A comparative study of probabilistic roadmap planners," in *Algorithmic Foundations of Robotics V*. Springer, 2004, pp. 43–57.

[2] T. Kunz and M. Stilman, "Time-optimal trajectory generation for path following with bounded acceleration and velocity," *Robotics: Science and Systems VIII*, pp. 1–8, 2012.

[3] T. Kröger, "Opening the door to new sensor-based robot applications—the reflexxes motion libraries," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 1–4.

[4] H. Pham and Q. C. Pham, "A new approach to time-optimal path parameterization based on reachability analysis," *IEEE Transactions on Robotics*, vol. 34, pp. 645 – 659, 06 2018.

[5] T. Haarnoja, A. Zhou, S. Ha, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," *CoRR*, vol. abs/1812.11103, 2018. [Online]. Available: http://arxiv.org/abs/1812.11103

[6] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.

[7] L. Berscheid, T. Rühr, and T. Kröger, "Improving data efficiency of self-supervised learning for robotic grasping," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 2125–2131.

[8] 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.

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

[10] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. W. Pachocki, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *CoRR*, vol. abs/1808.00177, 2018. [Online]. Available: http://arxiv.org/abs/1808.00177

[11] K. Ota, D. K. Jha, T. Oiki, M. Miura, T. Nammoto, D. Nikovski, and T. Mariyama, "Trajectory optimization for unknown constrained systems using reinforcement learning," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov 2019, pp. 3487–3494.

[12] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[13] 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 *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2017, pp. 23–30.

[14] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 1–8.

[15] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 4243–4250.

[16] J. C. Kiemel, P. Meißner, and T. Kröger, "TrueRMA: Learning fast and smooth robot trajectories with recursive midpoint adaptations in cartesian space," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2020.

[17] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 4569–4574.

[18] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," in *Advances in neural information processing systems*, 2017, pp. 971–980.

[19] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, W. Paul, M. I. Jordan, and I. Stoica, "Ray: A distributed framework for emerging AI applications," *CoRR*, vol. abs/1712.05889, 2017. [Online]. Available: http://arxiv.org/abs/1712.05889

[20] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica, "Rllib: Abstractions for distributed reinforcement learning," *arXiv preprint arXiv:1712.09381*, 2017.

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

[22] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," *GitHub repository*, 2016.

[23] W. Wiedmeyer, M. Mende, D. Hartmann, R. Bischoff, C. Ledermann, and T. Kroger, "Robotics education and research at scale: A remotely accessible robotics development platform," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 3679–3685.