2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

Learning Robot Trajectories subject to Kinematic Joint Constraints

Jonas C. Kiemel[1] and Torsten Kröger

2021 IEEE International Conference on Robotics and Automation (ICRA) | 978-1-7281-9077-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICRA48506.2021.9561159

***Abstract*— We present an approach to learn fast and dynamic robot motions without exceeding limits on the position θ, velocity $\dot{\theta}$, acceleration $\ddot{\theta}$ and jerk $\dddot{\theta}$ of each robot joint. Movements are generated by mapping the predictions of a neural network to safely executable joint accelerations. The neural network is invoked periodically and trained via reinforcement learning. Our main contribution is an analytical procedure for calculating safe joint accelerations, which considers the prediction frequency f_N of the neural network. As a result, the frequency f_N can be freely chosen and treated as a hyperparameter. We show that our approach is preferable to penalizing constraint violations as it provides explicit guarantees and does not distort the desired optimization target. In addition, the influence of the selected prediction frequency on the learning performance and on the computing effort is highlighted by various experiments.**

I. INTRODUCTION

Over the last few decades, robots have become increasingly prevalent in the manufacturing industry. While industrial robots are superior to humans in terms of speed and precision, they lack the capability to react to unforeseen events. For instance, human workers can easily deal with elastic or varying workpieces, whereas robots rely on precise models of their environment. Tracing back to remarkable achievements in gaming domains [1], [2], model-free reinforcement learning (RL) has caught the attention of researchers trying to address the problem of flexible industrial production [3]–[11]. Most commonly, the learning process involves a neural network that receives sensor data as input and outputs actions to parameterize motions. In simulated environments, various action parameterizations (e.g. target joint positions, velocities or torques) have proven to perform well [12]. However, when learning trajectories for real robots, the predicted motions must be executable without overloading the robot joints. In particular, jerk constraints are often ignored, although unbounded jerks are known to decrease the life span of the robot joints and to stimulate natural frequencies [13], [14].

Fig. 1 illustrates our approach to limit the position, velocity, acceleration and jerk based on an exemplary trajectory for a single joint. At each discrete decision step, the desired acceleration at the beginning of the following interval is determined. As a first step, we calculate the range of accelerations that can be safely executed. In Fig. 1, this range is visualized by a red line. Secondly, a neural network predicts an action $\in [-1, 1]$ that is linearly mapped along the range of feasible accelerations. In our example, every action is set to 1, meaning that the next acceleration setpoint

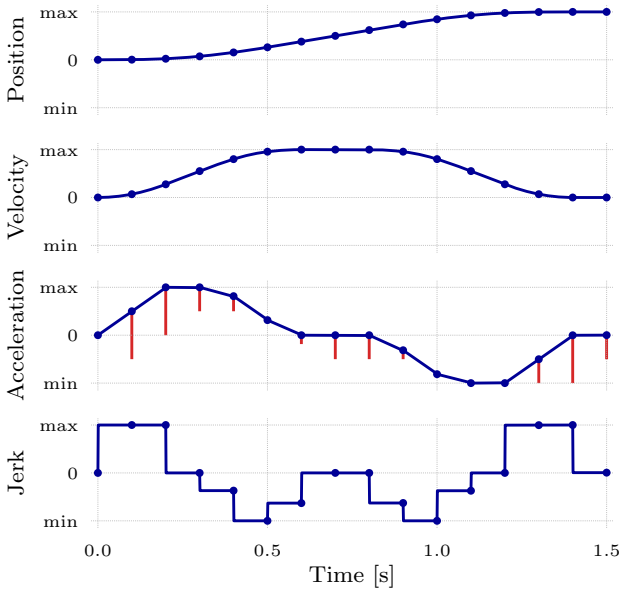
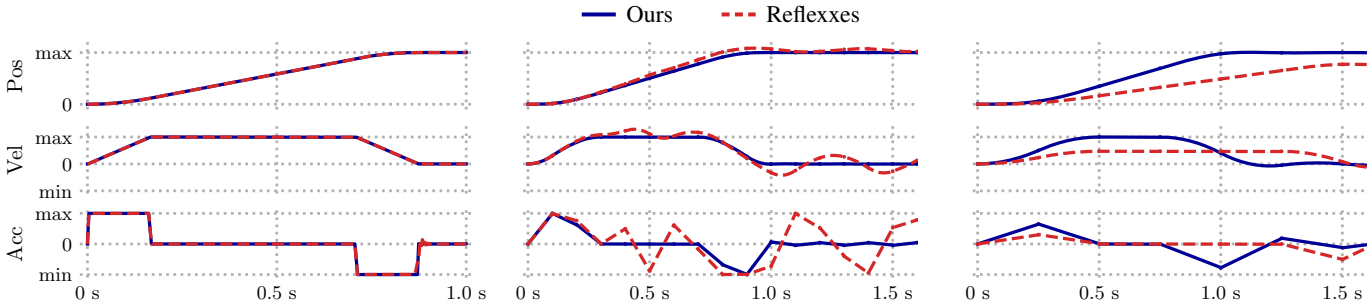


Fig. 1: Our method enables learning of robot trajectories that do not violate position, velocity, acceleration and jerk limits. At each decision step, a neural network predicts an action $\in [-1, 1]$ that is linearly mapped along the range of safe accelerations (red line). Prediction frequency f_N: 10 Hz

is mapped to the top end of the red line. In contrast, an action of -1 would be mapped to the bottom end of the corresponding red line. This way, each action $\in [-1, 1]$ leads to a feasible motion. Finally, a continuous trajectory is generated by linearly connecting the predicted acceleration setpoints (blue line). When calculating the range of safe accelerations, it is not sufficient to focus on violations within the following interval. For instance, the acceleration setpoint at $t = 0.9\,\mathrm{s}$ could be set to zero without causing immediate constraint violations. However, the maximum position would be exceeded at a later point in time. Our approach ensures that at least one valid acceleration setpoint exists at each future decision step. As a result, motions close to position or velocity limits can be learnt safely. Being able to exploit the full kinematic potential of the robot joints is important for industrial automation processes as faster motions enable shorter cycle times.

In the following sections, we explain how safe accelerations are computed and demonstrate the effectiveness of our approach by learning fast motions for two different tasks without exceeding joint limits. In addition, we show how the prediction frequency f_N influences the learning process and transfer a policy trained in simulation to a real robot. Our code to compute safe accelerations is publicly available.[2]

[1]Institute for Anthropomatics and Robotics – Intelligent Process Automation and Robotics (IAR-IPR), Karlsruhe Institute of Technology (KIT), jonas.kiemel@kit.edu, [2]www.github.com/translearn/limits



(a) f_N: 300 Hz (b) f_N: 10 Hz (c) f_N: 4 Hz

Fig. 2: Exemplary trajectories computed with Reflexxes and with our method by selecting the largest possible acceleration at each decision step. When using a high prediction frequency (a), both approaches produce feasible trajectories. If the frequency is reduced, the trajectory generated by Reflexxes exceeds the maximum position as in (b) or does not reach it as in (c).

II. Related work

A. Learning motions subject to safety constraints

In recent years, reinforcement learning has been adapted to industrial applications like bin picking [3]–[5], autonomous assembly [6], [7] and precise insertion [9]–[11]. While research institutions have presented well-functioning prototypes, the usage of neural networks in industrial environments still poses major challenges in terms of safety and reliability. According to [15], safety constraints can be incorporated into the learning process by either modifying the optimization criterion or the exploration process. Modifying the optimization criterion by adding penalties [16]–[18] is an easy-to-implement way to avoid undesired outcomes. However, compliance with safety constraints is not explicitly guaranteed, which is particularly problematic if the trained policy is subject to a domain transfer (e.g. training in simulation but deployment on a real robot).

The exploration process can be influenced by incorporating external knowledge. In some cases, the action space of a neural network can be designed in such a way that all actions are executable without violating safety constraints [17]. If the range of valid actions depends on the current state, task-specific heuristics can be applied to adapt unsafe action selections [19]. In [18], a method to learn action corrections from previous trajectories generated with random actions is presented. Achiam et al. [20] introduced a policy search algorithm for constrained Markov decision processes with near-constraint satisfaction at each iteration. In contrast, our approach provides explicit safety guarantees. In [21], a method to consider explicit inequality constraints in the context of model-free RL is presented. At each decision step, a quadratic program is defined. Its solution is the closest action to an initial network prediction that satisfies the chosen constraints. However, focusing on the current decision step is not sufficient when learning fast robot motions. As explained in the introduction, inevitable violations might occur at a later point in time. Contrary to the aforementioned work, our approach ensures that at least one valid action exists for each following decision step. A method to learn jerk-limited trajectories for point-to-point motions is presented in [22]. However, position constraints are not considered explicitly. Instead, the trajectory execution is aborted if the deviation to a reference trajectory is higher than a specified threshold. Contrary to that, the approach presented in this paper enables safe learning of arbitrary movements without requiring precalculated reference trajectories.

B. Relation to model-based trajectory optimization

Within the context of model-based trajectory optimization and model predictive control, joint limits can be considered by defining an optimization problem with explicit inequality constraints [23]. In contrast to model-free reinforcement learning, these techniques require a model of the dynamics and a differentiable loss function. While our approach considers the kinematics of robot joints, the interaction between the robot and its environment is learnt without a model.

C. Relation to online trajectory generation (OTG)

In the context of online trajectory generation, the Reflexxes motion library [24] can be used to compute time-optimal trajectories for industrial robots that comply with position, velocity, acceleration and jerk constraints. Contrary to our approach, Reflexxes assumes continuous control of the accelerations. Fig. 1 shows that the maximum position is finally reached at a velocity of zero if the highest valid acceleration is selected at each decision step. Reflexxes expects a target position and a target velocity as input and outputs the acceleration required to reach the target state in a time-optimal way. Given the maximum position and a velocity of zero as target, Reflexxes might be suitable to compute the highest valid acceleration. However, Fig. 2 illustrates that the library is not directly applicable to learning problems as the influence of discrete decision steps is ignored. At a prediction frequency of 300 Hz, both Reflexxes and our approach yield almost identical trajectories that do not violate the specified constraints. When selecting a frequency of 10 Hz, the trajectory generated by Reflexxes violates both position and velocity constraints. At a frequency of 4 Hz, neither the maximum position or the maximum velocity is reached when using Reflexxes. In conclusion, Reflexxes can be used for high prediction frequencies, while our approach ensures constraint satisfaction at arbitrary frequencies.

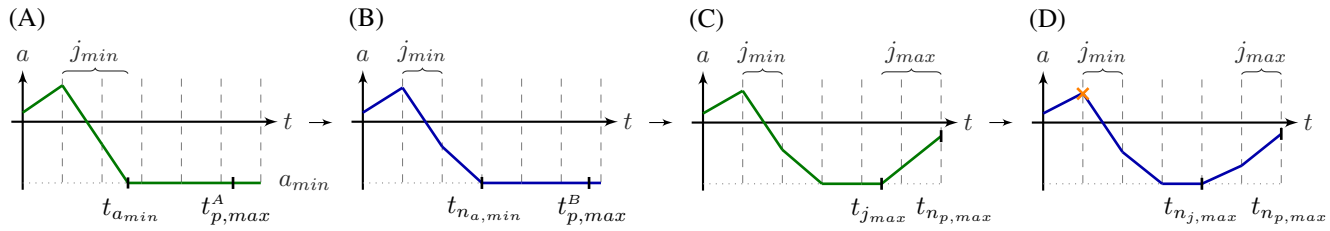


Fig. 3: Stepwise calculation of the highest acceleration that does not lead to a violation of the position limit p_{max} by successively applying the acceleration profiles (A) to (D).

III. Problem Statement

This work addresses the problem of learning online generation of robot motions without exceeding kinematic joint limits. In particular, the following constraints on the joint angle θ are defined for each robot joint and need to be satisfied at any time:

$$p_{min} \leq \theta \leq p_{max} \tag{1}$$
$$v_{min} \leq \dot{\theta} \leq v_{max} \tag{2}$$
$$a_{min} \leq \ddot{\theta} \leq a_{max} \tag{3}$$
$$j_{min} \leq \dddot{\theta} \leq j_{max}, \tag{4}$$

The limits are specified by the robot manufacturer but can also be set to lower values, e.g. to restrict the workspace of the robot or to enforce smoother movements.

IV. Learning Safe Motions

A. Formalization

We formalize the learning problem by defining a Markov decision process $(\mathcal{S}, \mathcal{A}, P_{\underline{a}}, R_{\underline{a}})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space and $R_{\underline{a}}$ is the immediate reward caused by action $\underline{a}$. The transition probabilities $P_{\underline{a}}$ are unknown. Using model-free RL, a neural network is trained to maximize the sum of immediate rewards. Decisions are made at a prediction frequency of f_N. The immediate reward is assigned according to task-specific optimization goals. The state $s_t \in \mathcal{S}$ that is given as input to the neural network consists of two parts: The kinematic state of each robot joint (position p_t, velocity v_t, acceleration a_t) and a task-specific component f_t, which might include external sensor signals. The action $\underline{a}_t \in \mathcal{A}$ is composed of a single scalar $m \in [-1, 1]$ per joint. This scalar is translated to a_{t+1}, the desired joint acceleration at the beginning of the next time interval. Let $a_{t+1_S} = [a_{t+1_{min}}, a_{t+1_{max}}]$ be the range of acceleration setpoints that does not lead to constraint violations. Given a_{t+1_S}, the network prediction m can be mapped to a safe acceleration a_{t+1}:

$$a_{t+1} = a_{t+1_{min}} + \frac{1+m}{2} \cdot (a_{t+1_{max}} - a_{t+1_{min}}) \tag{5}$$

The computation is performed for each joint and the resulting movement is executed by sending linearly interpolated acceleration setpoints to a joint trajectory controller.

B. Calculation of safe accelerations

This subsection describes, how the range of safe accelerations $a_{t+1_S} = [a_{t+1_{min}}, a_{t+1_{max}}]$ is computed. The following explanations focus on the maximum acceleration $a_{t+1_{max}}$ as the minimum acceleration $a_{t+1_{min}}$ can be calculated correspondingly. The basis idea is to break down the problem by calculating the maximum acceleration for each individual constraint (1) to (4). By selecting the smallest of these accelerations, all constraints are fulfilled simultaneously. In the following, the principle is explained based on position constraints. Further information on the other constraints and additional background knowledge can be found in [25].

Fig. 3 illustrates the steps to calculate the highest possible acceleration at the next decision step that does not lead to a violation of the position limit p_{max}. The desired acceleration is marked by an orange cross. A robot joint exceeds its position limit if the maximum position p_{max} is reached at a velocity greater than zero. On the contrary, the highest possible acceleration can be achieved if the maximum position is reached at a velocity of zero. Assuming $t_0 = 0$, this condition can be expressed as follows:

$$v_0 + \int_0^{t_{p,max}} a(t)\ dt = 0 \tag{6}$$

$$p_0 + v_0 \cdot t_{p,max} + \int_0^{t_{p,max}} \int_0^{t} a(t)\ dt\, dt = p_{max}, \tag{7}$$

with $t_{p,max}$ being the time at which the maximum position is reached. Equations (6) and (7) can be solved for the desired acceleration if the course of future accelerations is known. Given that the desired acceleration should be as high as possible, it has to be followed by a deceleration phase. The deceleration should be as strong as possible, but has to comply with the jerk and acceleration limits. As visualized in Fig. 3, the required acceleration profile can be determined in a step-wise manner:

- In step (A), the minimum jerk j_{min} is applied until the minimum acceleration a_{min} is reached. By solving equations (6) and (7), the continuous switching time $t_{a_{min}}$ can be calculated. Since the decisions of our neural network are made at discrete time steps, this profile is not directly applicable to our problem. However,

$t_{n_{a,min}}$, the time of the decision step that follows $t_{a_{min}}$, can be computed.

- Knowing $t_{n_{a,min}}$, the acceleration profile shown in step (B) can be applied. This profile is used to calculate $t^B_{p,max}$, the time at which the maximum position is reached. As demonstrated in the accompanying video[3], the use of this profile induces oscillations that can be avoided if the maximum position is reached at a discrete decision step. Given $t^B_{p,max}$, the next discrete time step $t_{n_{p,max}}$ can be computed.
- If $t_{n_{p,max}}$ is known, oscillations can be suppressed by applying acceleration profile (C). Similar to step (A), a continuous switching time $t_{j_{max}}$ is computed in the first place. Based on this value, the discrete switching time $t_{n_{j,max}}$ is calculated.
- Given $t_{n_{j,max}}$, the desired maximum acceleration, which is marked by an orange cross, can finally be calculated by applying acceleration profile (D).

C. Implementation

For given acceleration profiles, like those shown in Fig. 3, specific systems of equations can be derived from (6) and (7). For each resulting equation system, an analytical solution is obtained using SymPy [26]. To speed up calculations, the analytical expressions are translated into C code and compiled as a Python module. Our step-wise approach supports arbitrary prediction frequencies and does not rely on numeric solvers. In addition, our analytical expressions are differentiable. We note that the range of safe accelerations a_{t+1_S} can also be translated into a range of safe velocities v_{t+1_S} or positions p_{t+1_S}. Mapping the predicted actions linearly along a_{t+1_S}, v_{t+1_S} or p_{t+1_S} is equivalent.

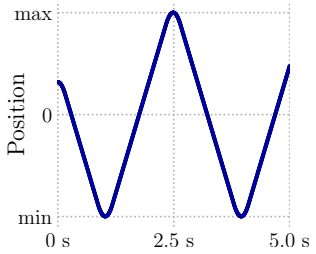
V. Evaluation

A. Description of the evaluation tasks

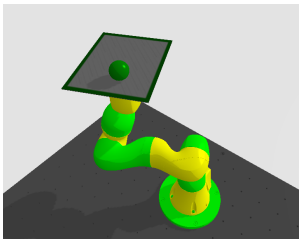
We evaluate our approach by applying it to two model-free learning problems performed by a KUKA iiwa robot with seven degrees of freedom. Both tasks are simulated with PyBullet [27] and trained using an RL algorithm called Proximal Policy Optimization [28]. Renderings of the resulting policies can be found in the accompanying video[3].

1) Velocity maximization task: The goal of this task is to maximize the absolute velocity of all joints over time without violating kinematic constraints. As shown in Fig. 4a, the optimal performance is achieved if each joint oscillates between its upper and its lower position limit. This task is selected for our evaluation, as the robot joints are required to work close to their kinematic limits in order to receive high rewards. In addition, the average absolute velocity achieved by the optimal policy helps to assess the final performance of the learning process. The duration of each generated trajectory is set to five seconds.

[3]https://youtu.be/JpkKCd9jyss



(a) Optimal trajectory for the velocity maximization task.



(b) Ball-on-plate task

Fig. 4: Evaluation tasks

2) Ball-on-plate task: In this task, the robot is trained to roughly follow a reference trajectory while balancing a ball on a plate. The following two varieties differ with respect to their reward function: The goal of the first version is to keep the ball as close as possible to its initial position ("in place"), while the second variant aims to prevent the ball from falling off the plate ("on plate"). Further information on the ball-on-plate task can be found in [22].

B. Constraint satisfaction

As a first step, we evaluate whether all trajectories generated by our method satisfy the specified constraints (1) to (4). Fig. 5 shows an exemplary trajectory for a single joint that is produced by choosing random actions $\in [-1, 1]$ at each decision step. The green dotted lines represent the range of safe accelerations. As expected, the kinematic limits are not violated. At $t = 2.2\,\mathrm{s}$, the range of safe accelerations is very small. However, at least one valid acceleration setpoint exists at any decision step. The border case of selecting the largest possible acceleration at each decision step is shown in Fig. 1 on the front page of this paper. It can be seen that

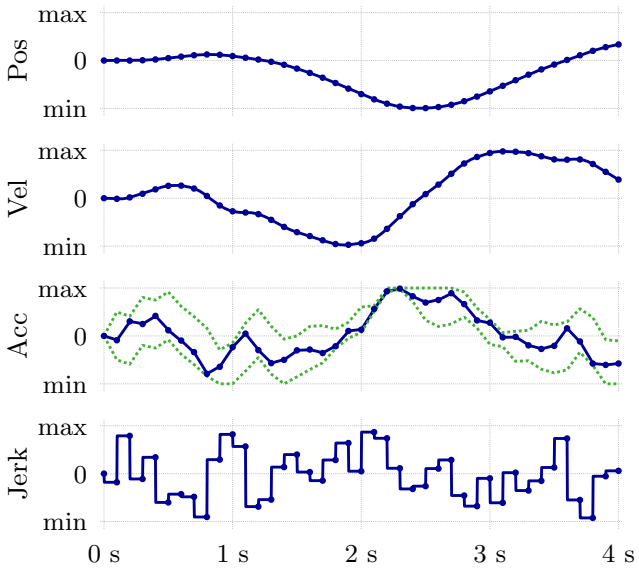


Fig. 5: An exemplary trajectory generated by choosing random actions. Our method ensures that joint constraints are never violated. Prediction frequency f_N: 10 Hz

TABLE I: Performance metrics for the velocity maximization task using different strategies to avoid constraint violations. Each settings is evaluated by generating 1000 episodes with neural networks that are trained until convergence.

	Method	Average velocity	Violation rate	Max. position	Max. velocity
240 Hz	Ours	0.91	0.0 %	1.00	1.00
	Soft penalties	0.51	8.0 %	1.25	0.90
	Hard penalties	0.71	52.7 %	1.43	1.56
20 Hz	Ours	0.88	0.0 %	1.00	1.00
	Soft penalties	0.44	1.2 %	0.93	1.14
	Hard penalties	0.58	41.5 %	1.14	1.69

the resulting trajectory is constrained by the maximum jerk, the maximum acceleration, the minimum jerk, the maximum velocity and finally by the maximum position. To assess the reliability of our approach on a larger scale, two neural networks are trained for the velocity maximization task. The prediction frequencies of the networks are set to 240 Hz and 20 Hz, respectively. We generate 1000 trajectories with each network and determine the highest occurring position and velocity value. These values are normalized with respect to their limits and listed in Table I. For both frequencies, the maximum position and the maximum velocity are equal to the specified limits.

C. Benchmarking with reward shaping

Avoiding constraint violations by adding penalties to the reward function is an easy-to-implement alternative to our approach. Table I compares our method with two different versions of penalty assignment based on the learning performance of the velocity maximization task. Green and red values indicate the best and the worst outcome for each metric. When using hard penalties, the reward is set to zero if a position or a velocity constraint is violated. In case of soft penalties, the reward is gradually reduced if either the current position or the current velocity is higher than 50 % of its maximum value. The violation rate indicates the fraction of trajectories with at least one constraint violation. Our results show that both penalty-based techniques fail to prevent constraint violations. While the violation rate with soft penalties is comparatively low, penalty-based methods do not provide explicit safety guarantees, which is unacceptable when working with real robots. In addition,

TABLE II: Performance metrics for the velocity maximization task at different prediction frequencies.

Frequency	Average velocity			Trajectories until convergence	
	Random	Trained	Optimal	Absolute	Relative
240 Hz	0.212	0.912	0.921	48 900	1.00
120 Hz	0.268	0.908	0.920	62 650	1.28
20 Hz	0.431	0.880	0.919	107 250	2.20
10 Hz	0.487	0.865	0.913	128 250	2.62

TABLE III: Performance metrics for the ball-on-plate task. An episode is considered as successful if the reference trajectory is tracked and if the ball is balanced correctly.

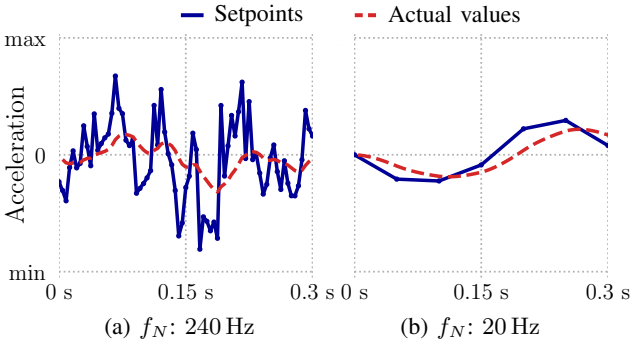
	Frequency	Success rate	Balancing error	Tracking error
In place	No adaptations	0.6 %	99.4 %	0.0 %
	240 Hz	0.5 %	11.0 %	88.5 %
	120 Hz	81.1 %	3.5 %	15.4 %
	20 Hz	99.4 %	0.1 %	0.5 %
	2.5 Hz	0.0 %	88.2 %	11.8 %
On plate	No adaptations	4.7 %	95.3 %	0.0 %
	240 Hz	33.5 %	64.2 %	2.3 %
	120 Hz	85.8 %	14.1 %	0.1 %
	20 Hz	98.5 %	1.4 %	0.1 %
	2.5 Hz	94.8 %	4.8 %	0.4 %

the average velocity, which should be maximized as part of the learning process, is significantly higher when using our method. The results are plausible as penalties distort the desired optimization target. In this particular case, the penalties encourage the robot to stay away from the joint limits, which is in contradiction to the goal of the velocity maximization task.

D. Influence of the prediction frequency f_N

Our approach supports arbitrary prediction frequencies as the impact of discrete decision steps is considered. In this subsection, the influence of the prediction frequency on the learning process is analyzed. Table II lists various performance metrics for the velocity maximization task. The results show that both the optimal and the trained policy achieve higher velocities if the prediction frequency is increased. This behaviour is plausible as higher prediction frequencies allow more granular control of the robot joints.

To compare the data efficiency, we assume convergence of a training process once the gain in reward has reached 98 % of its final value. Our results show that the number of trajectories required for convergence decreases if the prediction frequency is increased. Data efficiency is especially important if the training process is to be performed



(a) f_N: 240 Hz (b) f_N: 20 Hz

Fig. 6: Exemplary acceleration setpoints compared to actual values for a single joint.

by a real robot. Our observations are plausible as higher prediction frequencies lead to more training data per trajectory. Table III shows the performance of the ball-on-plate task at different prediction frequencies. The best balancing performance is achieved at a prediction frequency of 20 Hz. If the prediction frequency is set to 2.5 Hz, the "in place" version fails as the time between decision steps is too high to prevent the ball from moving. Contrary to the velocity maximization task, the ball-on-plate task is influenced by the tracking performance of the trajectory controller. Fig. 6 shows exemplary acceleration setpoints and the resulting actual values for a trajectory segment of the ball-on-plate task. Since the trajectory controller behaves like a low-pass, the acceleration setpoints are tracked more accurately, if the prediction frequency is reduced. Beyond that, the reaction of the system is delayed due to the inertia of the ball. At high prediction frequencies, the correlation between the predicted action and the reaction of the system is harder to assess. We conclude that the final learning performance does not necessarily profit from higher prediction frequencies if the controlled system behaves similarly to a low-pass.

E. Computing effort and real-time capability

In order to transfer a control policy from simulation to a real robot, all calculations need to be real-time capable. This subsection analyzes the calculating times to generate trajectories measured on a system equipped with a CPU (Intel i7-8700K) and a GPU (Nvidia GTX 1080 Ti). Computing the range of safe accelerations a_{t+1S} took 111 µs per joint and decision step, which is comparable to the calculating time required by Reflexxes (123 µs). Table IV shows how the computing effort is influenced by the selected prediction frequency. The computing effort increases at higher prediction frequencies as the calculations have to be performed more frequently. It is important to note that the calculating times need to be significantly shorter than the execution time of the resulting trajectory. Ideally, all calculations should be made at discrete points in time as illustrated by the arrows at t_0 and t_1 in Fig. 7. In practice, sensor data has to be collected prior to the desired decision time, which leads to a timing jitter of Δt_J. The relative jitter $J_R = \frac{\Delta t_J}{t_1 - t_0}$ increases at higher prediction frequencies and is further raised if additional safety checks for collision avoidance are to be performed. In summary, the technical effort required to achieve real-time capability is reduced when choosing a lower prediction frequency. As shown in the accompanying video, we successfully transferred a policy for the ball-on-plate task from simulation to a real robot using a prediction frequency of 20 Hz. Since our network predicts accelerations rather than torques, the domain transfer could be conducted without further measures to compensate model errors.

TABLE IV: Calculating times to generate a trajectory with a duration of 1000 seconds.

Frequency	Total	Action prediction		Safety checks		Other calculations	
		Abs.	Rel.	Abs.	Rel.	Abs.	Rel.
240 Hz	544 s	194 s	36 %	187 s	34 %	163 s	30 %
120 Hz	284 s	96 s	34 %	93 s	33 %	95 s	33 %
20 Hz	71 s	16 s	23 %	16 s	23 %	39 s	54 %
10 Hz	49 s	8 s	16 %	8 s	16 %	33 s	68 %

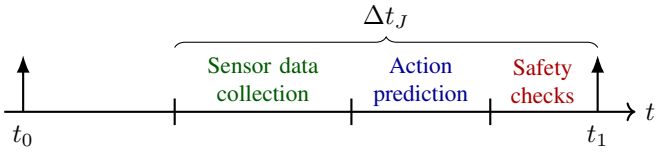


Fig. 7: Jitter Δt_J caused by the processing time of real systems.

VI. Conclusion and Future Work

This paper presented an approach to learn fast robot trajectories while satisfying kinematic joint constraints. The effectiveness of our method is demonstrated by successfully learning two different tasks without violating joint limits. In contrast to penalizing constraint violations, our approach provides explicit safety guarantees, which is crucial when working with real robots. The proposed method considers the impact of discrete decision steps, thereby enabling arbitrary prediction frequencies. Our experiments showed that the prediction frequency influences the final learning performance, the amount of training data required until convergence and the processing power needed for real-time execution.

For practical applications, it is also important to avoid collisions and violations of torque limits when generating robot movements. Therefore, the coupling between the robot joints has to be taken into account. In our follow-up work [29], we present a method for learning torque-limited robot trajectories while avoiding collisions with static obstacles and other robots. In future work, we are interested in extending our approach such that collisions with moving obstacles can also be avoided.

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 thank Tamim Asfour for his valuable feedback and advice.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[3] 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*, vol. 37, no. 4-5, pp. 421–436, 2018.

[4] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, *et al.*, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Conference on Robot Learning*, 2018, pp. 651–673.

[5] L. Berscheid, P. Meißner, and T. Kröger, "Self-supervised learning for precise pick-and-place without object model," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4828–4835, 2020.

[6] G. Thomas, M. Chien, A. Tamar, J. A. Ojea, and P. Abbeel, "Learning robotic assembly from cad," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–9.

[7] J. Luo, E. Solowjow, C. Wen, J. A. Ojea, A. M. Agogino, A. Tamar, and P. Abbeel, "Reinforcement learning on variable impedance controller for high-precision robotic assembly," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 3080–3087.

[8] 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.

[9] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana, "Deep reinforcement learning for high precision assembly tasks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 819–825.

[10] G. Schoettler, A. Nair, J. Luo, S. Bahl, J. A. Ojea, E. Solowjow, and S. Levine, "Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards," *arXiv preprint arXiv:1906.05841*, 2019.

[11] M. Kaspar, J. D. Muñoz Osorio, and J. Bock, "Sim2real transfer for reinforcement learning without dynamics randomization," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 4383–4388.

[12] X. B. Peng and M. van de Panne, "Learning locomotion skills using deeprl: Does the choice of action space matter?" in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2017, pp. 1–13.

[13] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *The International Journal of Robotics Research*, p. 027836492098785, Jan 2021. [Online]. Available: http://dx.doi.org/10.1177/0278364920987859

[14] T. Kröger, *On-Line Trajectory Generation in Robotic Systems: Basic Concepts for Instantaneous Reactions to Unforeseen (Sensor) Events*. Springer, 2010, vol. 58.

[15] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[16] O. 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," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[17] 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.

[18] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa, "Safe exploration in continuous action spaces," *arXiv preprint arXiv:1801.08757*, 2018.

[19] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[20] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *International Conference on Machine Learning*, 2017, pp. 22–31.

[21] T.-H. Pham, G. De Magistris, and R. Tachibana, "Optlayer-practical constrained optimization for deep reinforcement learning in the real world," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6236–6243.

[22] J. C. Kiemel, R. Weitemeyer, P. Meißner, and T. Kröger, "TrueÆdapt: Learning smooth online trajectory adaptation with bounded jerk, acceleration and velocity in joint space," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020.

[23] N. Mansard, O. Khatib, and A. Kheddar, "A unified approach to integrate unilateral constraints in the stack of tasks," *IEEE Transactions on Robotics*, vol. 25, no. 3, pp. 670–685, 2009.

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

[25] J. C. Kiemel, "Background knowledge for learning robot trajectories subject to kinematic joint constraints," 2021.

[26] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, *et al.*, "Sympy: symbolic computing in python," *PeerJ Computer Science*, vol. 3, p. e103, 2017.

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

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

[29] J. C. Kiemel and T. Kröger, "Learning collision-free and torque-limited robot trajectories based on alternative safe behaviors," *arXiv preprint arXiv:2103.03793*, 2021.