Safe Reinforcement Learning of Robot Trajectories in the Presence of Moving Obstacles

Jonas Kiemel[1], Ludovic Righetti[2], Torsten Kröger and Tamim Asfour[1]

***Abstract*—In this paper, we present an approach for learning collision-free robot trajectories in the presence of moving obstacles. As a first step, we train a backup policy to generate evasive movements from arbitrary initial robot states using model-free reinforcement learning. When learning policies for other tasks, the backup policy can be used to estimate the potential risk of a collision and to offer an alternative action if the estimated risk is considered too high. No matter which action is selected, our action space ensures that the kinematic limits of the robot joints are not violated. We analyze and evaluate two different methods for estimating the risk of a collision. A physics simulation performed in the background is computationally expensive but provides the best results in deterministic environments. If a data-based risk estimator is used instead, the computational effort is significantly reduced, but an additional source of error is introduced. For evaluation, we successfully learn a reaching task and a basketball task while keeping the risk of collisions low. The results demonstrate the effectiveness of our approach for deterministic and stochastic environments, including a human-robot scenario and a ball environment, where no state can be considered permanently safe. By conducting experiments with a real robot, we show that our approach can generate safe trajectories in real time.**

***Index Terms*—Motion Control, Reinforcement Learning, Robot Safety, Collision Avoidance**

I. INTRODUCTION

In recent years, model-free reinforcement learning (RL) has become increasingly popular for generating robot trajectories in real time. This trend is mainly driven by the fact that model-free RL can be easily applied to a wide range of robotic applications. Instead of using a differentiable model of the system dynamics, well-performing actions are identified during a training phase based on trial and error. When performing movements with a real robot, however, it is important to ensure that the selected actions do not cause damage. One approach to avoid safety violations is to execute an action only if the robot will remain in a safe state after executing the action. However, when a robot is operating in an environment with moving obstacles, determining whether the subsequent state is safe becomes non-trivial: The exact motion of the obstacles might not always be known in advance and the kinematic constraints of the robot joints may limit its ability to execute arbitrary evasive movements. In modern model-free RL, actions for a desired learning task are typically generated by a task policy, which is parameterized by a neural network. To prevent collisions during and after the training phase of the task policy, we propose to additionally use a backup policy trained on avoiding obstacles. As shown in Fig. 1, the backup policy serves two purposes. First, an action from the backup policy is executed if the task policy would lead to an unsafe follow-up state. Second, a rollout of the backup policy is used to assess whether the follow-up state is safe. While the rollout can be carried out in a physics simulator, we additionally evaluate the performance of neural networks trained to estimate collision risks based on data from previous rollouts. This becomes important when dealing with stochastic environments and reduces the computational effort required for real-time execution. The main contributions of the work are:

- We propose an approach for learning safe goal-directed motions in environments with moving obstacles while considering the kinematic constraints of robot joints.

Manuscript received: June, 10, 2024; Revised September, 15, 2024; Accepted October, 5, 2024.

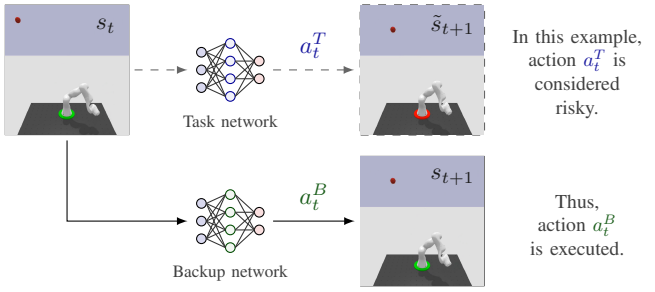
This paper was recommended for publication by Editor Jaydev P. Desai upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by a research travel grant of the DAAD-Stiftung.

[1]Institute for Anthropomatics and Robotics (IAR), Karlsruhe Institute of Technology (KIT), Germany, jonas.kiemel@kit.edu

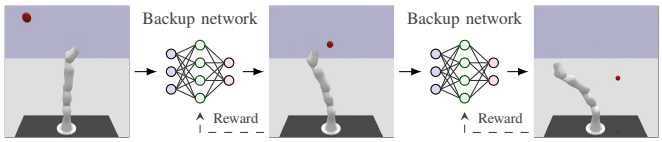
[2]Tandon School of Engineering, New York University (NYU), USA

Digital Object Identifier (DOI): see top of this page.

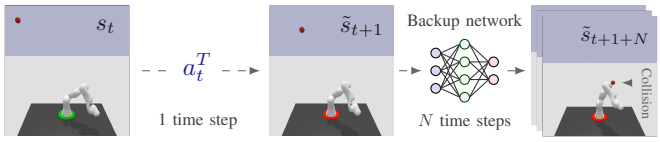
To avoid collisions, an action a_t^B from a backup policy is used if an action a_t^T from a task policy is considered risky:



The backup policy is trained in advance on avoiding collisions:



To check if a_t^T is risky, a background simulation is performed:



Alternatively, data-based risk estimators can be used (see Fig. 5).

Fig. 1: Our approach shown for the *Ball* environment, where balls are thrown towards the robot from random directions.

TABLE I: Overview of related approaches that address instantaneous safety constraints by adjusting actions of an RL agent.

	Action adjustment using	Application scenario	Main difference to our approach
• Pham et al. [1]	Faverjon and Tournassoud's method [2]	Reaching task with an industrial robot	Problem of conflicting constraints
• Fisac et al. [3]	Reachability analysis	Quadrotor control	Model-based safety controller
• Wabersich et al. [4]	Model-predictive control	Pendulum swing-up, quadrotor control	Model-based safety filter
• Kiemel et al. [5]	Braking trajectories	Reaching task with an industrial robot	No consideration of moving obstacles
• Yang et al. [6]	Backup policy trained via model-free RL	Locomotion with a quadruped robot	Model-based risk criteria
• Thananjeyan et al. [7]	Backup policy trained via model-free RL	Navigation and manipulation tasks	Backup policy depends on task policy

- We investigate the impact of stochastically moving obstacles, compare different methods to estimate collision risks, analyze potential reasons for incorrect estimates, and discuss measures to prevent them.
- We conduct a systematic quantitative analysis of the presented approach by learning a reaching task and a basketball task in three different environments. Experiments with a real robot demonstrate that safe trajectories can be generated in real time.

Our code is available at github.com/translearn/safeMotionsRisk.

II. Related work

The recent success of model-free RL in simulation environments has sparked a growing interest in research addressing the challenges of safe reinforcement learning with real robots. In model-free RL, safety constraints are typically formalized using a constrained Markov decision process (CMDP) [8]. In this context, a distinction is made between cumulative constraints [9], [10], which are defined with respect to a penalty received over time, and instantaneous constraints, which must be satisfied at each decision step. A survey of various approaches to increase safety during and after the training phase can be found in [11]. This work focuses on addressing instantaneous constraints by adjusting risky actions of an RL agent. An overview of existing approaches in this field and their differences to the approach presented in this work is provided in TABLE I.

Pham et al. [1] utilize Faverjon and Tournassoud's method [2] to avoid collisions with moving obstacles by solving a quadratic program (QP). This approach, however, does not ensure recursive feasibility, meaning that the QP may not have a solution due to conflicting constraints. For the specific case of collision avoidance in a two-dimensional plane, Zhao et al. [12] addressed the problem of recursive feasibility using barrier certificates. More generally, techniques from model-based control can be used to increase safety when using RL [13]–[15]. For example, Fisac et al. [3] utilize Hamilton–Jacobi reachability analysis to avoid ground contact with a quadrotor. Wang et al. [16] use barrier functions to avoid collisions with static obstacles in the context of real-time navigation. Wabersich et al. [4] utilize a safety filter based on model-predictive control to swing-up an inverted pendulum and to control a quadrotor in a simulation environment. The safety filter modifies actions if no safely executable backup trajectory with a specified time horizon is found otherwise. While our approach is also based on the idea of safe backup trajectories, we utilize a backup policy trained via model-free RL to adjust risky actions. The use of a model-free backup policy offers the advantage that no differentiable model of the system dynamics needs to be specified [17]. Moreover, generating an action with a backup policy represented by a neural network requires little computational effort. In one of our previous works [5], we utilized time-optimal braking trajectories as backup trajectories [18], [19]. The approach reliably prevents self-collisions and collisions with static obstacles, but does not account for moving obstacles. To overcome this limitation, we utilize model-free RL to train a backup policy that actively avoids collisions with moving obstacles which may behave stochastically. The use of model-free RL is related to Yang et al. [6], where a backup policy is trained to stabilize a quadruped robot, and to Thananjeyan et al. [7], where backup policies are used for navigation and manipulation tasks. In contrast to Yang et al. [6], our approach does not make use of model-based risk criteria. Instead, the risk of an action is determined by performing a background simulation or by utilizing a data-based risk estimator [20]. Compared to Thananjeyan et al. [7], our backup policy does not depend on the current task policy. As a result, the backup policy can be used to learn different tasks without needing to be updated. By utilizing a special action space for both the task policy and the backup policy [21], we additionally ensure that the resulting trajectories are jerk-limited no matter which policy is selected. In our evaluation section, we provide comparative results with two alternative approaches [1], [5] that have been used in the context of industrial robotics.

III. Problem statement

We assume that model-free RL is used to learn motions for a robotic manipulator. During and after the training phase, self-collisions and collisions with static or moving obstacles must be avoided. In addition, the following kinematic constraints must be satisfied by each robot joint at all times:

$$\mathrm{p}_{\min} \leq \ \theta \leq \mathrm{p}_{\max} \tag{1}$$

$$\mathrm{v}_{\min} \leq \ \dot{\theta} \leq \mathrm{v}_{\max} \tag{2}$$

$$\mathrm{a}_{\min} \leq \ \ddot{\theta} \leq \mathrm{a}_{\max} \tag{3}$$

$$\mathrm{j}_{\min} \leq \dddot{\theta} \leq \mathrm{j}_{\max}, \tag{4}$$

where θ is the joint position.

As common for industrial robots, the base of the manipulator is assumed to be fixed. In the most general case, a motion determined by an action of an RL agent can be safely executed if an infinite follow-up trajectory is known, such that the composed movement satisfies the desired safety constraints [18]. If no such trajectory exists, the action of the RL agent needs to be adjusted. While reasoning over infinite trajectories is impractical, the condition can be relaxed if safe goal states exist. In this context, a safe goal state is a state in which a safe

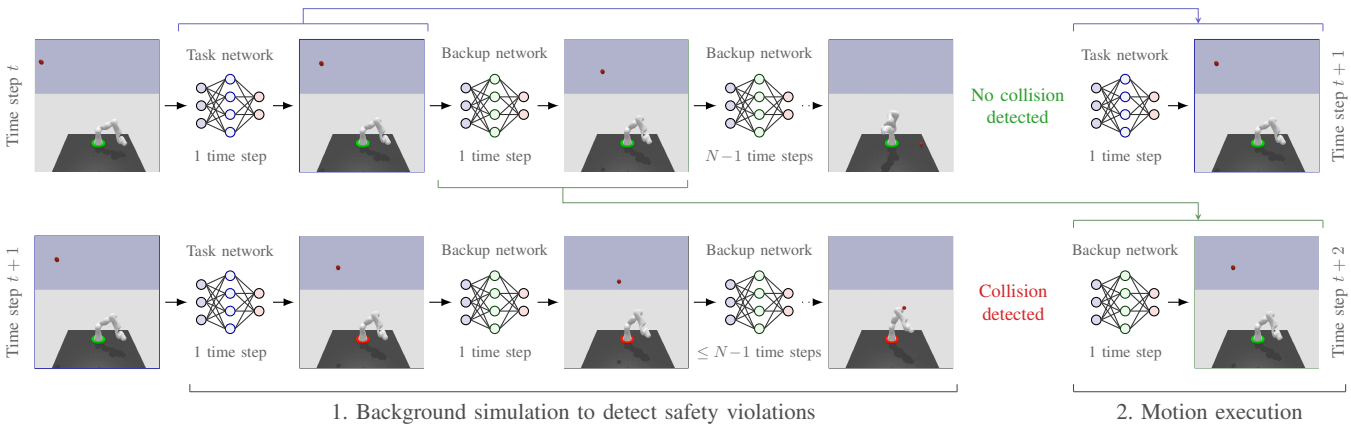


Fig. 2: Collision avoidance by ensuring the existence of a safe backup trajectory. See section IV-A for details.

follow-up trajectory is known. The most trivial case is a resting state, provided that the robot is stopped in a region of the environment where no moving obstacles are present. The better a method performs in finding safe follow-up trajectories, the less actions of the RL agent need to be adjusted. Restraining the actions of the RL agent as little as possible is desirable to avoid a negative impact on the learning performance.

If the obstacles in the environment do not move deterministically but according to a stochastic model, only stochastic statements about potential collision risks are possible. As a consequence, a trade-off between the exploration of the environment and the risk of a collision has to be made. In our work, we consider deterministic and stochastic environments with and without safe resting states.

IV. Approach

A. Basic principle

The basic principle of our approach is illustrated in Fig. 2. The figure shows two time steps of an environment in which a robot has to avoid a ball thrown in its direction. Our goal is to train a task policy, represented by a so-called *task network*, without causing a collision. For this purpose, we make use of a backup policy, represented by a *backup network*, that was trained to avoid collisions beforehand. At both time steps t and $t+1$, the first step is to compute a desired action using the task network. Given the current state of the environment as input, the output of the task network is a probability distribution from which a desired action is sampled. However, before executing the motion resulting from the selected action, the backup network is used to generate a backup trajectory with a duration of N time steps. The composed movement with a duration of $N+1$ time steps is then checked for collisions in a background simulation using a physics simulator. In the case of time step t, no collision is found during the background simulation. Consequently, the motion from time step t to $t+1$ is executed as defined by the action from the task network. Contrary to this, a collision is detected during the background simulation conducted at time step $t+1$. Thus, the action from the task network is replaced by an action from the backup network. Note that the resulting motion from time step $t+1$ to $t+2$ was part of the collision-free trajectory simulated at time step t.

Provided that

- the background simulation accurately reflects reality and
- a collision-free backup trajectory is found at least every $N+1$ time steps,

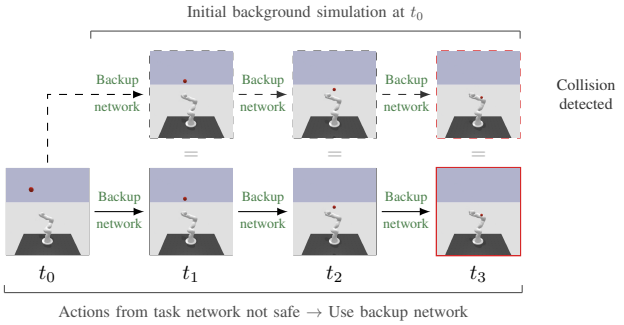
the method described above ensures that the resulting motion of the robot is always collision-free. Ideally, N would be infinite. In practice, N is limited by the computational effort required for the background simulation. If obstacles do not move deterministically, a single background simulation is no longer sufficient to decide if an action is safe. To account for stochastic environments, a so-called risk network can be trained to predict the probability of a collision based on data from previous rollouts of the backup policy. In this case, a trade-off between the amount of adjusted actions and the resulting average time to a collision emerges. The trade-off can be controlled by selecting a suitable threshold value.

In the remainder of this section, we analyze potential failure causes (IV-B), describe how the backup network is trained (IV-C), clarify the risk estimation via risk networks (IV-D) and explain the training of the task network (IV-D).

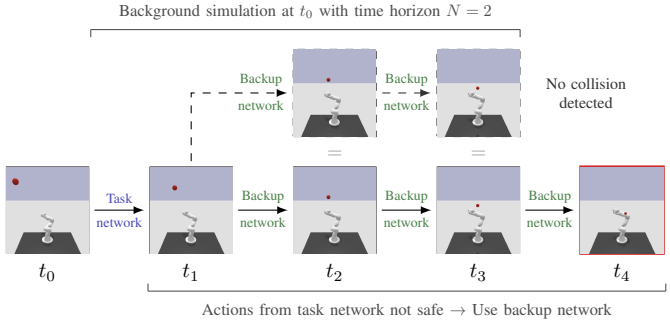
B. Failure mode analysis

In the following, we analyze conditions that can lead to a collision even if a background simulation is performed. As shown in Fig. 3a, a collision can occur if the initial state of the environment is not safe. More precisely, this means that a rollout of the backup policy from the initial state leads to a collision. Collisions can also occur if the time horizon of the background simulation is too short (Fig. 3b). A potential mitigation is to extend the time horizon of the background simulation N. However, a longer time horizon increases the computational effort required for the background simulation. A third possible failure cause becomes apparent when the environment is non-deterministic. In the example shown in Fig. 3c, a new ball is sampled once the previous ball missed the robot. The direction of the new ball is selected randomly and is therefore not known in advance. As a result, the ball direction selected during the background simulation differs from the actual ball direction. In stochastic environments, it is possible to estimate the probability of a collision based on data from previous rollouts of the backup policy. This can be done

(a) Initial state not safe:



(b) Time horizon of the background simulation too short:



(c) Environment non-deterministic:

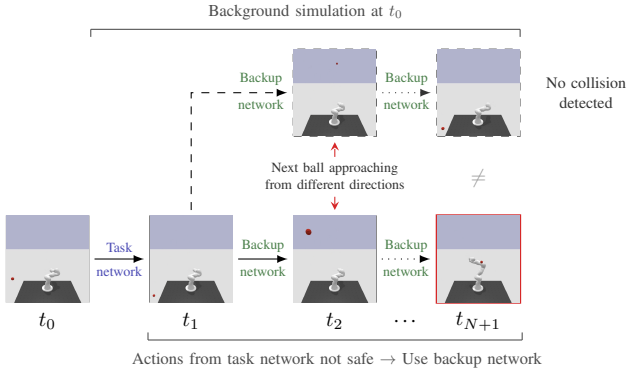


Fig. 3: Potential failure causes when performing a background simulation to detect safety violations.

be training a so-called risk network via supervised learning. When using a risk network, the computational effort no longer depends on the time horizon N, so that real-time execution becomes possible even if a long time horizon needs to be taken into account. Risk networks can also be used to estimate the risk of initial states. On the downside, it is important to note that incorrect risk predictions by the risk network introduce an additional source of error.

C. Training of the backup policy

To train the backup policy π_B via model-free RL, we define a Markov decision process $(\mathcal{S}, \mathcal{A}, P, R)$. The backup policy is represented by a neural network trained to map states $s_t \in \mathcal{S}$ to actions $a_t \in \mathcal{A}$ such that the sum of future rewards is maximized.

1) State space $\mathcal{S}$: Each state $s_t \in \mathcal{S}$ is composed of two parts: The first part $s_{t_{Ki}}$ describes the kinematic state of the robot, and the second part $s_{t_{Mo}}$ defines the state of the moving

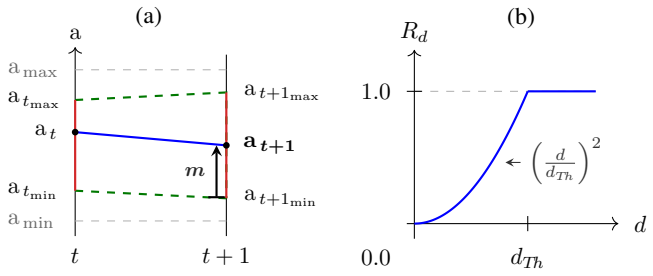


Fig. 4: Action mapping (a) and distance reward (b).

obstacles in the environment. More precisely, $s_{t_{Ki}}$ consists of the current position p_t, velocity v_t and acceleration a_t of each controllable robot joint, normalized to their respective maximum values. Details on the environment-specific part $s_{t_{Mo}}$ are given in section V-A.

2) Action space $\mathcal{A}$: We utilize an action space introduced in [21], which ensures that the kinematic joint limits (1) - (4) are satisfied at all times. An action $a_t \in \mathcal{A}$ consists of one scalar $m \in [-1, 1]$ per controllable robot joint. As shown in Figure 4a, the scalar m specifies a joint acceleration a_{t+1} within a range of feasible accelerations $[\mathrm{a}_{t+1_{\min}}, \mathrm{a}_{t+1_{\max}}]$. To generate continuous acceleration setpoints, the current acceleration a_t is linearly connected to a_{t+1}. Subsequently, velocity and position setpoints for a trajectory controller are computed through integration. By enforcing jerk limits for each robot joint, we ensure that the resulting trajectory is smooth no matter which action is selected.

3) Reward function R: We compute the immediate reward r_t for an action a_t as follows:

$$r_t = \alpha \cdot R_{d_{t+1_{Mo}}} + \beta \cdot R_{d_{t+1_{St}}} + \gamma \cdot R_{d_{t+1_{Sc}}} + R_{T_B}, \quad (5)$$

where α, β and γ are weighing factors. The reward components $R_{d_{t+1_{Mo}}}$, $R_{d_{t+1_{St}}}$ and $R_{d_{t+1_{Sc}}}$ depend on $d_{t+1_{Mo}}$, $d_{t+1_{St}}$ and $d_{t+1_{Sc}}$, the minimum distances to moving obstacles, static obstacles and self-collisions at $t+1$. The shape of the function used to compute $R_{d_{t+1_{Mo}}}$, $R_{d_{t+1_{St}}}$ and $R_{d_{t+1_{Sc}}}$ is shown in Fig. 4b. In the figure, the variable d corresponds to $d_{t+1_{Mo}}$, $d_{t+1_{St}}$ or $d_{t+1_{Sc}}$, while d_{Th} is a fixed threshold value. For distances smaller than d_{Th}, the reward increases as d increases. As a result, $R_{d_{t+1_{Mo}}}$, $R_{d_{t+1_{St}}}$ and $R_{d_{t+1_{Sc}}}$ encourage the robot to keep a distance from moving obstacles, static obstacles and self-collisions, respectively. The last term R_{T_B} is a termination bonus which is always zero except when the episode terminates without a collision.

4) Termination: An episode is terminated after T time steps or earlier if a collision occurs. Note that the immediate reward is never negative, discouraging an early termination.

5) Sampling of initial states: Once trained, we do not update the backup policy. For that reason, it is important to cover a wide range of initial states during the training of the policy. To do so, the moving obstacles are initialized in a random configuration. After that, random joint positions are sampled until a collision-free robot position is found. To find feasible initial joint velocities and accelerations, we first choose random values within the specified kinematic limits (2) - (3) and compute the range of feasible joint accelerations $[\mathrm{a}_{t+1_{\min}}, \mathrm{a}_{t+1_{\max}}]$ as explained in section IV-C2. If the range is empty, the sampling process is repeated.

D. Data-based risk estimation

1) Data generation: The first step towards training a data-based risk estimator is to generate a dataset from rollouts of the backup policy. For that purpose, we utilize a physics simulator and initialize the environment in a random state s_t as described in section IV-C5. We then select a uniformly random action a_t and simulate the next time step until s_{t+1}. Starting from s_{t+1}, a rollout of the backup policy with N time steps is simulated. If a collision occurs during the $N+1$ simulated time steps, a risk signal c_t is defined to be 1.0, otherwise 0.0. For each rollout, we store the tuple $(s_t,\ a_t,\ s_{t+1},\ c_t)$.

2) Training of risk networks: We evaluate two different types of risk networks, both trained with a binary cross-entropy loss using supervised learning. State-action-based risk networks are trained to predict the risk c_t given s_t and a_t, whereas state-based risk networks use s_{t+1} to predict c_t. The use of data-based risk estimators makes it possible to account for the stochastic behavior of moving obstacles. Specifically, a risk network can learn to output a risk prediction between 0.0 and 1.0 depending on the probability of a collision.

E. Risk-aware training of the task policy

As for the backup policy, the training of the task policy is based on model-free RL and a Markov decision process. A state $s_t \in \mathcal{S}$ consists of three parts $s_{t_{Ki}}$, $s_{t_{Mo}}$ and $s_{t_{Ta}}$. While $s_{t_{Ki}}$ and $s_{t_{Mo}}$ correspond to the state components used for the backup network, the third part $s_{t_{Ta}}$ encodes additional task-specific information. An action $a_t \in \mathcal{A}$ is composed of $a_{t_{Ki}}$ and $a_{t_{Ta}}$, where $a_{t_{Ki}}$ determines the movement of the robot joints and $a_{t_{Ta}}$ is an additional task-specific action component. If the risk c_t predicted by the risk network exceeds a predefined risk threshold c_{Th}, the action component $a_{t_{Ki}}$ is replaced by an action from the backup network, which is denoted as $a^B_{t_{Ki}}$. Since both the task network and the backup network utilize the same method to map actions to movements, the kinematic joint limits (1) - (4) are satisfied no matter which policy is executed. The reward function for the training of the task policy depends on the desired learning task. A concrete example for a reaching task is given by equation (6) in section V-D.

The procedure of the risk-dependent action adjustment is shown in Fig. 5. We analyze and compare four different methods to estimate the risk c_t. The first option (A) is to use the state-action-based risk network. The other three options rely on the state-based risk network. In the case of (B1), the current state is used for the risk prediction. A disadvantage of this method is that it tends to adjust actions too late, i.e. when the robot is already in a risky state. This drawback can be avoided by checking the risk of the following state s_{t+1} instead. However, the state s_{t+1} is usually not fully known in advance. While the kinematic part $s_{t+1_{Ki}}$ can be computed using the action from the task network, the state of the moving obstacles $s_{t+1_{Mo}}$ is either considered constant (B2a) or forecasted (B2b), e.g. with a Kalman filter. Note that when checking the risk of the following state s_{t+1}, the risk of the transition from s_t to s_{t+1} is not explicitly considered.

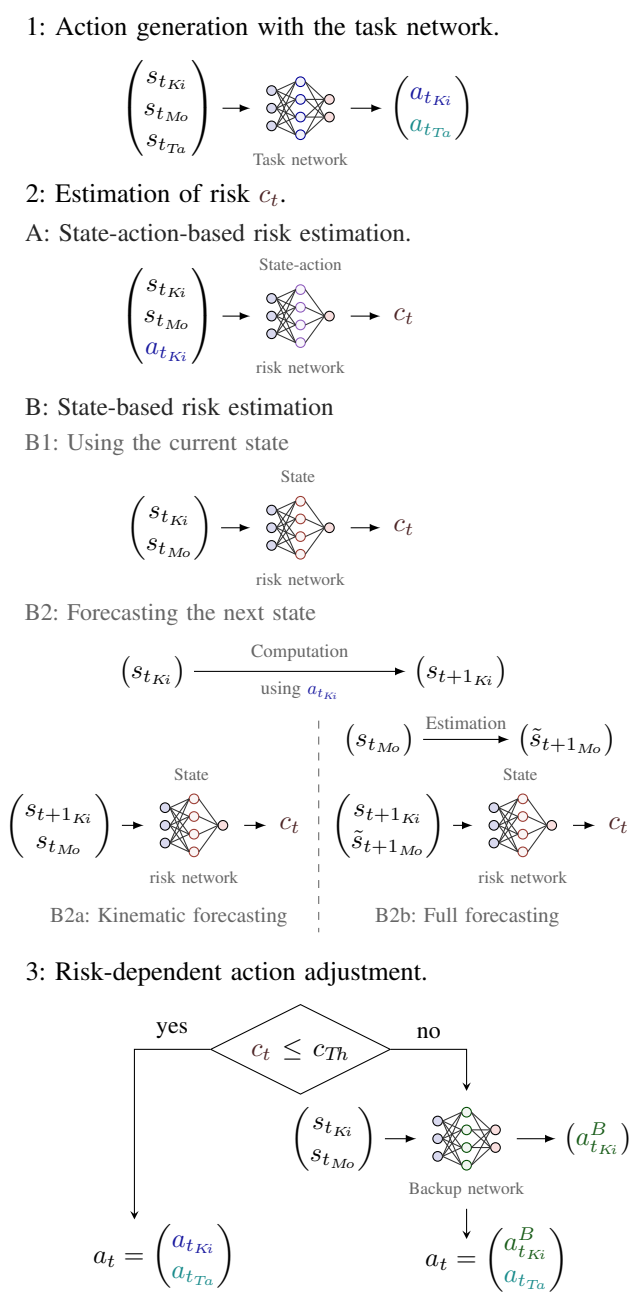
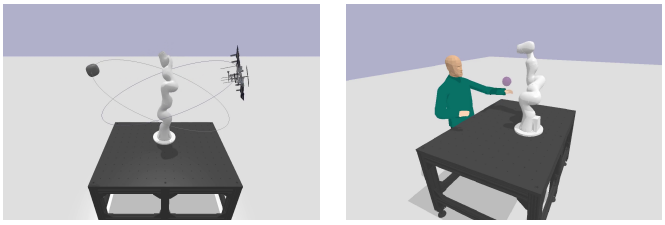


Fig. 5: The figure illustrates the action generation with the task network, four different ways (A, B1, B2a, and B2b) to estimate the corresponding risk, and the risk-dependent action adjustment using the backup network.



(a) Space (b) Human

Fig. 6: Two environments used for our evaluation.

V. Evaluation

A. Evaluation environments

We evaluate our approach in three different environments: A space environment and a human environment, shown in Fig. 6, and a ball environment shown in Fig. 1.

1) Space (deterministic): A space station and an asteroid orbit the robot in a deterministic manner. The environment-specific part of the state $s_{t_{Mo}}$ indicates the position of both objects on their orbit.

2) Ball (stochastic): In this environment, a ball is thrown towards the robot. The environment is stochastic as the initial position and the direction of the ball are selected at random. Once the ball is thrown, it moves along a parabolic trajectory due to gravity. The environment-specific part of the state $s_{t_{Mo}}$ includes the position and the velocity of the ball. A new ball is thrown, as soon as the previous one missed the robot.

3) Human (stochastic): In this environment, a human moves his hand to a target point without considering the motions of the robot. The environment is stochastic since the position of the target point is selected at random. The state component $s_{t_{Mo}}$ indicates the kinematic state of the human arms but not the desired target point of the human.

In all environments, a collision occurs on average after three seconds if a task policy selects random actions without making use of a backup policy (see Table II).

TABLE II: Results for random actions without our method.

Environment	Time until collision	Self-collision	Collision with table	Collision with moving obstacles
• Space	2.6 s	30 %	30 %	40 %
• Ball	3.3 s	38 %	45 %	17 %
• Human	3.4 s	38 %	41 %	21 %

B. Training of the backup policy

To train a backup policy for each environment, we use the on-policy RL-algorithm PPO [22] and a feedforward neural network with two hidden layers. The time between decision steps is set to 0.1 s. During training, each episode is terminated after $T = 20$ time steps or earlier if a collision occurs. As shown in Table III, the fraction of collision-free episodes increases significantly during training. Note that the fraction cannot reach 100 % since the robot is sometimes initialized in states where no evasive motion exists. The rightmost column of Table III shows the time until a collision if the robot is initialized such that no collision occurs during the first 2.0 s. In the space and the human environment, very high values are achieved. In these environments, the backup policy can guide the robot into a region of the workspace where no collisions with moving obstacles occur. In the ball environment there is no such region. As a result, a collision occurs on average after 85 s.

TABLE III: Training results of the backup policy.

Environment	Episodes without collision within 2.0 s		Time until collision
	Untrained agent	Trained agent	
• Space	38 %	91 %	> 40 000 s
• Ball	48 %	91 %	85.4 s
• Human	51 %	94 %	> 40 000 s

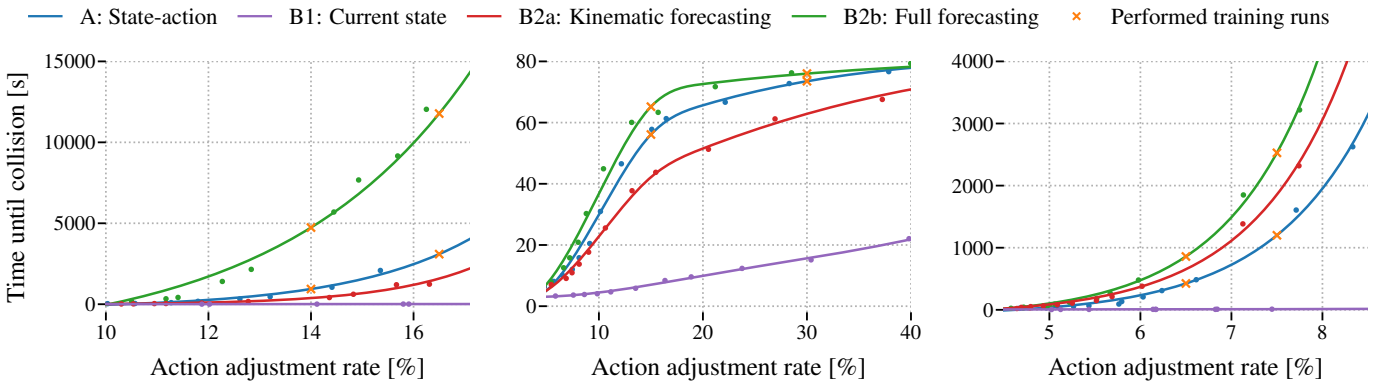
TABLE IV: Random actions with background simulations. Time until collision, Action adjustment rate, Collision with table or self-collision, Computation time per simulation time

Environment	$N = 0$ 0.0 s	$N = 1$ 0.1 s	$N = 5$ 0.5 s	$N = 20$ 2.0 s	$N = 30$ 3.0 s
Space	2.7 s	6.8 s	170.4 s	>2000 s	>2000 s
	4.3 %	6.7 %	7.5 %	7.5 %	7.5 %
	57 %	32 %	0 %	0 %	0 %
	38 %	48 %	119 %	235 %	303 %
Ball					
• Stochastic	3.6 s	10.4 s	64.1 s	160.3 s	116.3 s
	3.4 %	4.7 %	5.5 %	7.1 %	8.4 %
	78 %	43 %	6 %	4 %	9 %
	27 %	42 %	119 %	239 %	290 %
• Deterministic	3.8 s	10.9 s	68.7 s	164.7 s	181.6 s
	3.2 %	4.7 %	5.5 %	7.5 %	8.5 %
	82 %	43 %	0 %	0 %	0 %
	27 %	43 %	103 %	235 %	316 %
Human					
• Stochastic	3.7 s	11.1 s	62.9 s	72.3 s	64.5 s
	3.2 %	4.5 %	4.5 %	4.5 %	4.5 %
	76 %	53 %	13 %	13 %	11 %
	57 %	89 %	205 %	474 %	673 %
• Deterministic	3.7 s	11.3 s	700.6 s	>2000 s	>2000 s
	3.2 %	4.8 %	4.6 %	4.3 %	4.5 %
	74 %	67 %	0 %	0 %	0 %
	53 %	77 %	266 %	524 %	702 %

C. Using background simulations to detect collisions

For each environment, we conduct experiments using random actions and a single background simulation with different time horizons N. The results are shown in Table IV. The so-called action adjustment rate indicates the proportion of random actions that are replaced by actions from the backup policy. We select the initial states of each environment such that a safe backup trajectory with a time horizon of $N = 30$ exists, eliminating the failure case from Fig. 3a. In the stochastic environments, we also perform experiments under the assumption that the environment behaves deterministically, which additionally rules out the failure case from Fig. 3c. For $N = 0$, only the current time step is simulated. In this case and for $N = 1$, the time until a collision increases only slightly. We conclude that a longer time horizon is required to perform an evasive movement with the robot. For $N \geq 5$, self-collisions and collisions with the table hardly occur anymore, meaning that moving obstacles are the dominating reason for collisions. In deterministic environments, collisions with moving obstacles occur less frequently when a longer time horizon is chosen. In stochastic environments, however, this is not necessarily the case, as the informative value of a single background simulation decreases over time. The higher the time horizon of the background simulation, the greater the computational effort. In our experiments conducted with an Intel i9-9900K CPU, the computation time exceeded the simulation time for $N \geq 5$. Consequently, real-time execution requires either more computational power or the usage of computationally efficient risk estimators.

Table V shows a comparison with [5], where braking trajectories are used as backup trajectories. On average, the



(a) Space (deterministic) (b) Ball (stochastic) (c) Human (stochastic)

Fig. 7: The relation between the average time to a collision and the action adjustment rate for a task policy selecting random actions using four different ways to estimate the risk with a risk network as illustrated in Fig. 5.

braking trajectories require $N = 2.9$ time steps to bring the robot joints to a full stop. Once the robot is stopped, self-collisions and collisions with static obstacles do no longer occur. Consequently, all collisions in the experiments shown in Table V are caused by moving obstacles. However, compared to our backup policy trained using model-free RL, the braking trajectories do not evade moving obstacles. As a result, the average time to a collision is low when using braking trajectories as backup trajectories.

TABLE V: Comparative results using braking trajectories [5].

Environment	Time until collision	Adjustment rate	Time horizon N	Collision with moving obstacles
• Space	13.9 s	9.7 %	2.9	100 %
• Ball	21.3 s	8.2 %	2.9	100 %
• Human	31.9 s	7.1 %	2.9	100 %

D. Using risk networks to estimate the risk of a collision

The risk networks used for our evaluation are trained based on previous rollouts of the backup policy with a time horizon of $N = 20$. Once the risk networks are trained, the risk threshold c_{Th} can be used to control the average time to a collision. With $c_{Th} = 1.0$, no actions are adjusted by the backup policy. With $c_{Th} = 0.0$, all actions are adjusted leading to a rollout of the backup policy (see Table III). By running experiments with different risk thresholds, the relation between the action adjustment rate and the time to a collision can be determined. Based on this relation, it is possible to compare different methods for estimating the risk of an action. Fig. 7 shows the relation for a task policy selecting random actions when initializing the environments as described in section V-C.

TABLE VI: Training results for a reaching task with state-action-based (A) / state-based (B2b) risk estimation.

Initial action adjustment rate	Target points per second	Time until collision	Final action adjustment rate
Space			
• 14.0 %	1.01 / 0.94	423 s / 887 s	9.1 % / 8.8 %
• 16.5 %	0.93 / 0.90	428 s / 1647 s	10.0 % / 8.8 %
Ball			
• 15.0 %	0.70 / 0.71	69 s / 69 s	18.2 % / 17.7 %
• 30.0 %	0.52 / 0.51	89 s / 83 s	29.4 % / 31.1 %
Human			
• 6.5 %	0.79 / 0.70	96 s / 114 s	10.0 % / 12.7 %
• 7.5 %	0.49 / 0.63	191 s / 139 s	16.3 % / 16.8 %

In all cases, the state-based risk network with full forecasting (B2b) yields the highest time to a collision at a given action adjustment rate. However, the state-based risk network with full forecasting requires knowledge about the next state of the moving obstacles, which is not needed when using the state-action-based risk network (A). For our evaluation, we assume that the next state of the moving obstacles can be forecasted.

As shown in Table VI, we used PPO [22] to train task policies for a reaching task using the state-action-based risk estimation (A) and the state-based risk estimation (B2b). The backup policy and the risk networks were used during training and evaluation. For the reaching task, the task-specific state component $s_{t_{Ta}}$ encodes the position of a randomly sampled target point and the task policy is rewarded for quickly reaching the target point:

$$r_t = d_{t_{Ta}} - d_{t+1_{Ta}}, \tag{6}$$

where $d_{t_{Ta}}$ and $d_{t+1_{Ta}}$ are the distances between the end effector of the robot and the target point at time step t and $t+1$, respectively. For each experiment, we used a fixed risk threshold c_{Th} so that a specified action adjustment rate was obtained at the beginning of the training process. In Table VI, we additionally indicate the final action adjustment rate obtained after training. While the resulting task performance and the average time to a collision depend on the task policy learned during the training process, we identified the following tendencies in our experiments: Selecting a higher initial action adjustment rate also increased the average time to a collision for trained agents. On the other hand, the training performance, measured by the number of target points reached per second, decreased when selecting a higher initial action adjustment rate. In the space environment, the state-based risk estimation led to significantly fewer collisions than the state-action-based risk estimation. In the other two environments, both risk estimation methods led to similar training results.

E. Benchmarking with a QP-based method

We compare our approach with a QP-based method [1] used during training and evaluation. To keep the number of QP constraints low, we simplify our space environment by removing the table and ignoring self-collisions. In addition, we model the moving obstacles and the end effector of the robot as a sphere. Table VII shows the results of the comparison. It

TABLE VII: QP baseline in a simplified space environment.

	Untrained agent		Trained agent	
	Adjustment rate	Time until collision	Target points per second	Time until collision
• QP method	12.3 %	35.6 s	0.70	2216.6 s
• State-action (A)	7.1 %	855.2 s	0.82	4325.2 s

can be seen, that our method leads to fewer collisions while adjusting fewer actions. Moreover, the trained agent learns to reach a larger number of target points per time.

F. Learning a basketball task

In addition to the reaching task, we use the same backup networks and the state-action-based risk networks to learn a basketball task, where the robot is rewarded for placing balls into basketball hoops moving around the robot. To this end, the task-specific action component $a_{t_{Ta}}$ controls the speed at which a ball should leave a tube attached to the robot. As shown in Table VIII and in our video, the robot manages to learn the task while keeping the risk of a collision low.

TABLE VIII: Training results for a basketball task.

	Untrained agent		Trained agent	
	Hoops scored per second	Time until collision	Hoops scored per second	Time until collision
• Space	0.007	831.5 s	1.84	$>$ 20 000 s
• Ball	0.010	78.4 s	1.38	86.2 s
• Human	0.008	1258.1 s	1.19	$>$ 20 000 s

G. Sim-to-real transfer and real-time capability

Our method produces jerk-limited trajectories that can be tracked by a real robot without overloading the robot joints. In addition, the task policy, the backup policy and the risk estimation require only a small amount of computing power as they are based on neural networks (see Table IX). In our video, we demonstrate a successful sim-to-real transfer of a reaching policy in the space environment using a KUKU iiwa robot. For the experiment, the space station and the asteroid are assumed to move as during the training phase.

TABLE IX: Computation time per simulation time.

Risk network	State-action	State		
	A	B1	B2a	B2b
• Space	8.7 %	8.6 %	8.6 %	8.8 %
• Ball	10.7 %	10.5 %	11.1 %	10.9 %
• Human	18.5 %	17.6 %	18.2 %	18.6 %

VI. Conclusion and Future Work

We presented an approach to learn robot trajectories in the presence of moving obstacles while keeping the risk of collisions low. We confirmed the effectiveness of our approach by successfully learning a reaching task and a basketball task in three different environments and demonstrated real-time capability by running a policy trained in simulation on a real robot. An interesting direction for future research is to investigate measures to reduce the action adjustments caused by the backup policy, e.g., by searching for a safe action close to the desired action of the task policy. It would also be interesting to investigate the impact of uncertainty in sensor measurements as an additional source of stochasticity. Furthermore, we are interested in applying our approach to other robotic systems and other safety constraints, for instance by learning to control a bipedal robot without falling over.

References

[1] 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.

[2] B. Faverjon and P. Tournassoud, "A local based approach for path planning of manipulators with a high number of degrees of freedom," in *Proceedings. 1987 IEEE international conference on robotics and automation*, vol. 4. IEEE, 1987, pp. 1152–1159.

[3] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin, "A general safety framework for learning-based control in uncertain robotic systems," *IEEE Transactions on Automatic Control*, vol. 64, no. 7, pp. 2737–2752, 2018.

[4] K. P. Wabersich and M. N. Zeilinger, "A predictive safety filter for learning-based control of constrained nonlinear dynamical systems," *Automatica*, vol. 129, p. 109597, 2021.

[5] J. C. Kiemel and T. Kröger, "Learning collision-free and torque-limited robot trajectories based on alternative safe behaviors," in *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*. IEEE, 2022, pp. 223–230.

[6] T.-Y. Yang, T. Zhang, L. Luu, S. Ha, J. Tan, and W. Yu, "Safe reinforcement learning for legged locomotion," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 2454–2461.

[7] B. Thananjeyan, A. Balakrishna, S. Nair, M. Luo, K. Srinivasan, M. Hwang, J. E. Gonzalez, J. Ibarz, C. Finn, and K. Goldberg, "Recovery rl: Safe reinforcement learning with learned recovery zones," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4915–4922, 2021.

[8] E. Altman, *Constrained Markov decision processes*. CRC Press, 1999.

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

[10] Y. Liu, J. Ding, and X. Liu, "Ipo: Interior-point policy optimization under constraints," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 4940–4947.

[11] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, pp. 411–444, 2022.

[12] W. Zhao, T. He, and C. Liu, "Model-free safe control for zero-violation reinforcement learning," in *5th Annual Conference on Robot Learning*, 2021.

[13] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," *Advances in neural information processing systems*, vol. 30, 2017.

[14] T. Koller, F. Berkenkamp, M. Turchetta, and A. Krause, "Learning-based model predictive control for safe exploration," in *2018 IEEE conference on decision and control (CDC)*. IEEE, 2018, pp. 6059–6066.

[15] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 3387–3395.

[16] K. Wang, C. Mu, Z. Ni, and D. Liu, "Safe reinforcement learning and adaptive optimal control with applications to obstacle avoidance problem," *IEEE Transactions on Automation Science and Engineering*, vol. 21, no. 3, pp. 4599–4612, 2024.

[17] A. Hans, D. Schneegaß, A. M. Schäfer, and S. Udluft, "Safe exploration for reinforcement learning." in *ESANN*, 2008, pp. 143–148.

[18] T. Fraichard, "A short paper about motion safety," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 1140–1145.

[19] S. Rubrecht, V. Padois, P. Bidaud, M. De Broissia, and M. D. S. Simoes, "Motion safety and constraints compatibility for multibody robots," *Autonomous Robots*, vol. 32, no. 3, pp. 333–349, 2012.

[20] K. Srinivasan, B. Eysenbach, S. Ha, J. Tan, and C. Finn, "Learning to be safe: Deep rl with a safety critic," *arXiv:2010.14603*, 2020.

[21] J. C. Kiemel and T. Kröger, "Learning robot trajectories subject to kinematic joint constraints," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 4799–4805.

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