Online Generation of Safely Executable Robot Trajectories Using Reinforcement Learning

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Jonas Kiemel

Tag der mündlichen Prüfung: 23.07.2024

1. Referent: Prof. Dr.-Ing. Torsten Kröger
2. Referent: Prof. Dr.-Ing. Tamim Asfour

I would like to express my sincere gratitude to my advisors Torsten Kröger and Tamim Asfour for their continuous support, encouragement, and guidance throughout my doctoral studies.
With their mentorship and expertise, Torsten and Tamim not only shaped my own research, but also helped me to grow personally and professionally.

I am also very grateful to Prof. Ludovic Righetti and the DAAD-Stiftung for granting me the opportunity to conduct a research stay at New York University.
My time in New York was a unique experience that broadened my horizons and led to lasting friendships with an exceptional group of researchers from all over the world.

Furthermore, I would like to acknowledge my colleagues in Karlsruhe and all the students I had the pleasure of working with.
Your enthusiasm, camaraderie, and support not only enhanced the productivity of my work, but also gave me a sense of joy every day.

To my family and friends, thank you very much for always being there for me with your love, compassion, and encouragement.
My special thanks go to my parents and my brother.
Your unwavering support has been a source of strength and motivation throughout my entire life.

Finally, I would like to extend my gratitude to everyone who has supported me over the last few years.
Your efforts have not gone unnoticed and are greatly appreciated.

Abstract

Online Generation of Safely Executable Robot Trajectories Using Reinforcement Learning

This thesis addresses the problem of generating safely executable robot trajectories using model-free reinforcement learning (RL). In model-free RL, the performance of a robot is improved over time by utilizing the principle of trial and error. While recent advances in artificial intelligence have made this approach a compelling choice for developing autonomous, self-learning robots, ensuring safety throughout the learning process is still a subject of ongoing research. By presenting novel techniques for generating safely executable robot trajectories in real time, this thesis introduces three key contributions to the field of safe reinforcement learning, which are outlined below.

Using model-free RL, the generation of robot trajectories is typically formalized based on the mathematical framework of a Markov decision process. The goal of the learning process is to find a policy that maps states to actions so that rewards are maximized. The *first* contribution of this thesis addresses the question how actions can be mapped to robot trajectories that strictly adhere to position, velocity, acceleration, and jerk constraints specified for each robot joint. The introduction of an upper and a lower trajectory concept shows that each possible action can be mapped to a constraint-satisfying intermediate trajectory. The proposed mapping technique enables the learning of fast and smooth movements without overloading the robot joints and therefore serves as the foundation for the following two contributions.

In order to incorporate further safety constraints into the learning process, the *second* contribution of this work introduces the concept of background simulations. These simulations are conducted to identify risky actions, which are then replaced using a so-called backup policy. To avoid safety violations for robots with a stable base, a backup policy based on braking trajectories is proposed. Using this backup policy, it is shown that self-collisions and collisions with static obstacles can be strictly prevented.

To efficiently detect risky actions in stochastic environments, the *third* contribution introduces data-based risk estimators. In a first step, model-free RL is used to learn a backup policy that actively avoids safety violations. Subsequently, the backup policy is employed to generate data for a risk estimator trained via supervised learning. The risk estimation is utilized to avoid collisions in diverse settings with moving obstacles. It is shown that the proposed technique effectively reduces collisions while causing little computational overhead.

The presented contributions are extensively evaluated using simulated industrial robots and humanoid robots. By successfully performing experiments with a real industrial robot, it is shown that safely executable trajectories can be generated in real time. The thesis concludes with a discussion on future research directions to advance the development of safe autonomous robots.

Zusammenfassung

Online Generierung von sicher ausführbaren Robotertrajektorien mittels bestärkendem Lernen

Diese Arbeit behandelt das Problem der Generierung von sicher ausführbaren Robotertrajektorien unter Verwendung von bestärkendem Lernen (Reinforcement Learning, RL). Bei modellfreiem bestärkendem Lernen wird das Prinzip von Versuch und Irrtum genutzt, um die Leistungsfähigkeit eines Roboters sukzessive zu steigern. Während jüngste Fortschritte in der künstlichen Intelligenz diese Methode zu einer vielversprechenden Wahl für die Entwicklung autonomer, selbstlernender Roboter machen, ist die Aufrechterhaltung der Sicherheit während des Lernprozesses noch Gegenstand laufender Forschung. Durch die Vorstellung neuartiger Techniken zur Generierung von sicher ausführbaren Robotertrajektorien in Echtzeit führt diese Arbeit drei wesentliche Beiträge in den Forschungsbereich des sicheren bestärkenden Lernens ein, die im Folgenden erläutert werden.

Bei der Verwendung von modellfreiem bestärkendem Lernen wird die Generierung von Robotertrajektorien in der Regel basierend auf einem Markov-Entscheidungsprozess formalisiert. Während des Lernprozesses besteht das Ziel darin, eine sogenannte Policy zu finden, die Zustände so auf Aktionen abbildet, dass Belohnungen maximiert werden. Der *erste* Beitrag dieser Arbeit adressiert die Frage, wie Aktionen auf Trajektorien abgebildet werden können, sodass Positions-, Geschwindigkeits-, Beschleunigungs- und Ruckgrenzwerte für jedes Robotergelenk strikt eingehalten werden. Durch die Einführung des Konzeptes einer oberen und einer unteren Trajektorie wird gezeigt, dass jede mögliche Aktion einer Trajektorie zugeordnet werden kann, welche die festgelegten Grenzwerte einhält. Die vorgeschlagene Zuordnung ermöglicht das Erlernen schneller und flüssiger Bewegungen ohne die Robotergelenke zu überlasten und dient deshalb als Grundlage für die folgenden zwei Beiträge.

Um weitere Sicherheitsanforderungen während des Lernprozesses zu berücksichtigen, führt der *zweite* Beitrag dieser Arbeit das Konzept von Hintergrundsimulationen ein. Diese Simulationen werden durchgeführt, um riskante Aktionen zu identifizieren und anschließend mittels einer sogenannte Backup-Policy zu ersetzen. Für Roboter mit festem Stand wird zur Vermeidung von Sicherheitsverletzungen eine Backup-Policy basierend auf Bremstrajektorien vorgeschlagen. Es wird gezeigt, dass Eigenkollisionen und Kollisionen mit unbeweglichen Hindernissen durch Verwendung dieser Backup-Policy vollständig vermieden werden können.

Zur effizienten Erkennung riskanter Aktionen in stochastischen Umgebungen führt der *dritte* Beitrag datenbasierte Risikoschätzer ein. In einem ersten Schritt wird bestärkendes Lernen verwendet, um eine Backup-Policy zu erlernen, die Sicherheitsverletzungen aktiv vermeidet. Anschließend wird die Backup-Policy eingesetzt, um Daten für einen Risikoschätzer zu generieren, der durch überwachtes Lernen trainiert wird. Die Risikoschätzung wird verwendet, um Kollisionen in Umgebungen mit sich bewegenden

Hindernissen zu vermeiden. Es wird gezeigt, dass die vorgeschlagene Technik Kollisionen effektiv reduziert und nur wenig Rechenleistung benötigt.

Die vorgestellten Beiträge werden unter Nutzung von simulierten Industrierobotern und humanoiden Robotern umfassend evaluiert. Durch die erfolgreiche Durchführung von Experimenten mit einem echten Industrieroboter wird gezeigt, dass sicher ausführbare Trajektorien in Echtzeit generiert werden können. Die Arbeit endet mit einer Diskussion über mögliche Forschungsrichtungen zur Weiterentwicklung sicherer autonomer Roboter.

Contents

1. Introduction

With their exceptional speed and precision, industrial robots have become an indispensable part of automation processes over the past 50 years. At present, the operational stock stands at approximately 3.9 million units, with more than 550 000 robots newly installed in 2022 [118]. Today, industrial robots are primarily employed in clearly defined environments where well-functioning motion sequences can be specified in advance. Emerging trends in automation, however, introduce new challenges for the field of robotics. In the future, robots will be increasingly used to produce customized goods, to collaborate with humans and to assist with everyday household tasks. As the operating environments become more versatile and less structured, programming motion sequences in advance becomes increasingly complex. This also holds true for identifying a suitable mathematical model of the environment needed to utilize techniques from model-based control theory. A promising approach for generating adaptive robot movements without precise knowledge of the operating environment is model-free *reinforcement learning*, a method from the field of machine learning.

When using model-free reinforcement learning, the operating environment is explored based on the principle of trial and error. For that purpose, randomly selected *actions* are mapped to robot movements. Subsequently, a *reward* is assigned to each action based on its effectiveness in fulfilling a desired robot task. During a training phase, a *learning algorithm* seeks to identify actions that maximize the rewards received over time. This way, it is possible to generate optimized robot movements without requiring a model of the environment or demonstrations from a human teacher.

Since around 2015, the combination of reinforcement learning with neural networks – sometimes referred to as *deep reinforcement learning* (DRL) – has received increasing research interest in the context of *artificial intelligence* (AI). Neural networks are a class of function approximators that are loosely inspired by the structure of the human brain. While the concept of artificial neural networks has been around for years, several recent advances have led to a sudden surge in popularity:

- The availability of computing power has increased significantly, in particular through cloud computing but also due to the development of hardware accelerators such as *graphics processing units* (GPUs) and *tensor processing units* (TPUs). Higher computing power speeds up the training process of neural networks and facilitates real-time applications.
- Special software frameworks for machine learning have been developed. These frameworks support a technique called *automatic differentiation*, which simplifies the implementation of gradient descent methods typically used to optimize the trainable parameters of neural networks.
- Several high-profile publications have attracted substantial media attention, thereby increasing the awareness of neural networks and artificial intelligence as a whole.

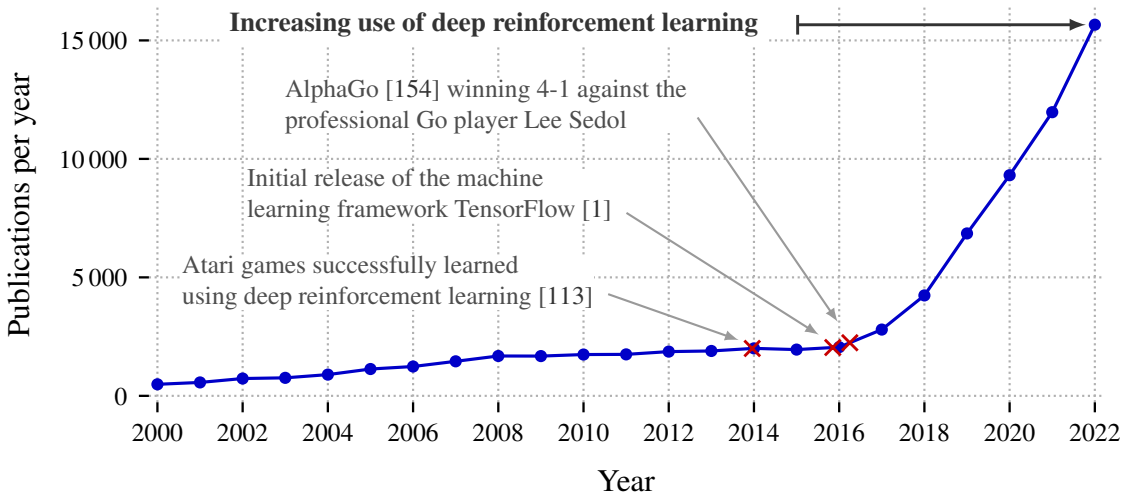


Figure 1.1.: Publications containing the term *reinforcement learning* in the title, in the abstract or in the keywords as found by the scientific search engine Scopus.

Figure 1.1 shows how the number of scientific publications on reinforcement learning has increased over the last few years. While the emergence of deep reinforcement learning opens up new opportunities for robot motion generation, it also presents new research questions. One important issue when controlling the motions of a robot with a neural network is the problem of motion safety. In order to learn well-performing motions with model-free reinforcement learning, a robot has to be able to explore its environment. During this process, neither the robot nor its surroundings must be damaged. This is particularly crucial at the beginning of a training process, where the network parameters are typically initialized at random. By providing methods to ensure motion safety during and after the training phase, this thesis contributes to the long-term goal of making robot movements as efficient and adaptive as those of their biological counterparts.

1.1. Problem description

This thesis addresses the problem of *online trajectory generation* using model-free *reinforcement learning* (RL), with a focus on incorporating safety constraints. The following section introduces the basic principle of online trajectory generation and offers an overview of potential factors that may lead to safety violations. Subsequently, safety constraints that are relevant within the scope of this work are formally defined.

1.1.1. Online trajectory generation using model-free reinforcement learning

In the past, model-free RL has been successfully applied to a wide range of sequential decision problems. A crucial factor for this achievement is the utilization of a common mathematical framework called *Markov decision process* (MDP). By formalizing decision problems as MDPs, it is possible to use the same learning algorithms across various problem domains. In the following, the framework of MDPs is introduced. Afterwards, it

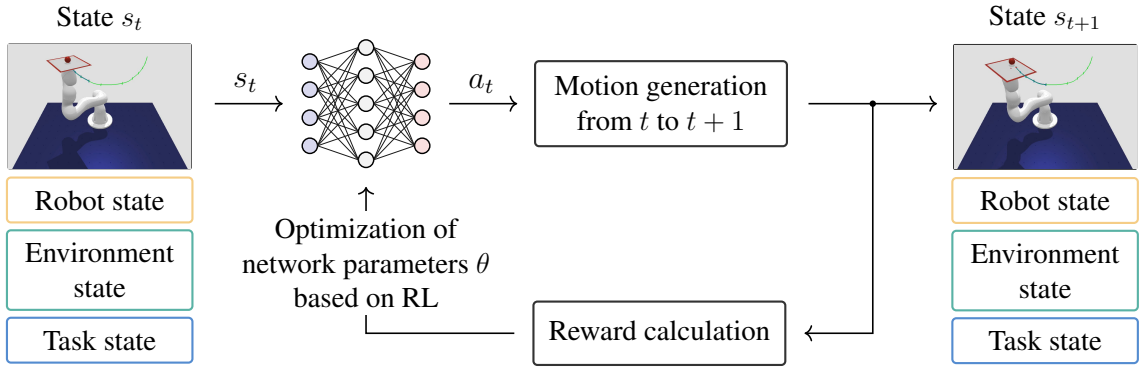


Figure 1.2.: The basic principle of online trajectory generation via reinforcement learning (RL) shown for a ball-on-plate task. In this example, the mapping from states s to actions a is performed by a fully connected neural network.

is shown how this framework can be applied to the specific problem of online trajectory generation. The mathematical descriptions in this section are based on the standard textbook "Reinforcement Learning: An Introduction" by Richard S. Sutton and Andrew G. Barto [160]. The notation used in this thesis aims to align with recent RL publications.

A system modeled as a *Markov decision process* is defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$:

- $\mathcal{S}$ is the state space specifying the set of feasible system states. MDPs are memory-less, meaning that a state $s_t \in \mathcal{S}$ fully describes the state of the system at time step t. Previous states s_{t-1}, s_{t-2}, ... have no influence on the further evolution of the system. This characteristic is called *Markov property*.
- $\mathcal{A}$ is the action space defining the set of actions. At each time step t, an action $a_t \in \mathcal{A}$ is selected.
- P: $\mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ is a function specifying the state-transition probabilities with $P(s' \mid s, a) = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$. When using model-free RL, the state-transition probabilities are typically unknown.
- R: $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. The reward function $R(s, a)$ provides a scalar value, called reward, indicating how well the selection of action a in state s performs with respect to the desired goal of the decision process.
- $\gamma \in [0, 1]$ is a discount factor to trade-off future rewards against immediate rewards.

When using model-free RL in the context of MDPs, the objective is to find a mapping from states to actions that maximizes the sum of discounted rewards over time G_t:

$$G_t = \sum_{k=0}^{\infty} \gamma^k \cdot R(s_{t+k}, a_{t+k}) \tag{1.1}$$

In the context of RL, the mapping is called policy and the sum defined by (1.1) is called return. A policy $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$ can be stochastic with $\pi(a|s)$ being the probability of selecting action a in state s:

$$\pi(a|s) = \Pr\{a_t = a \mid s_t = s\} \tag{1.2}$$

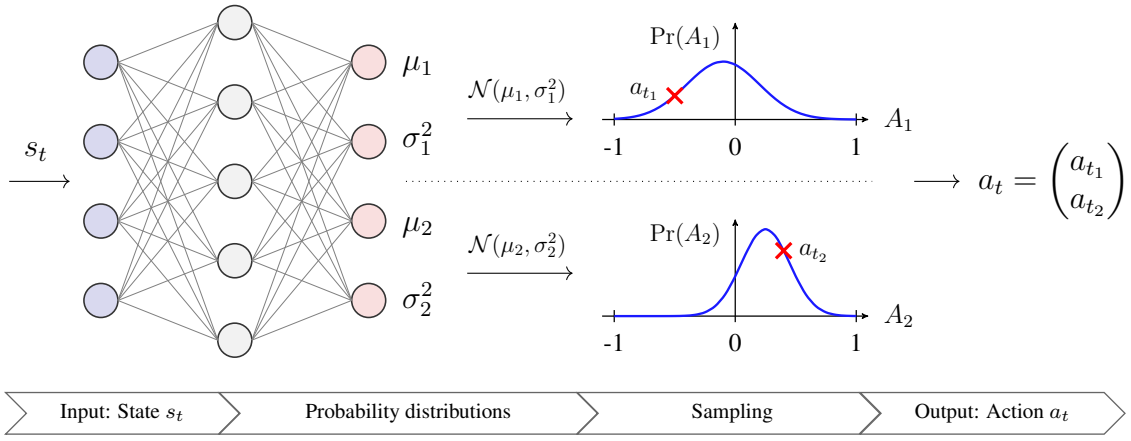


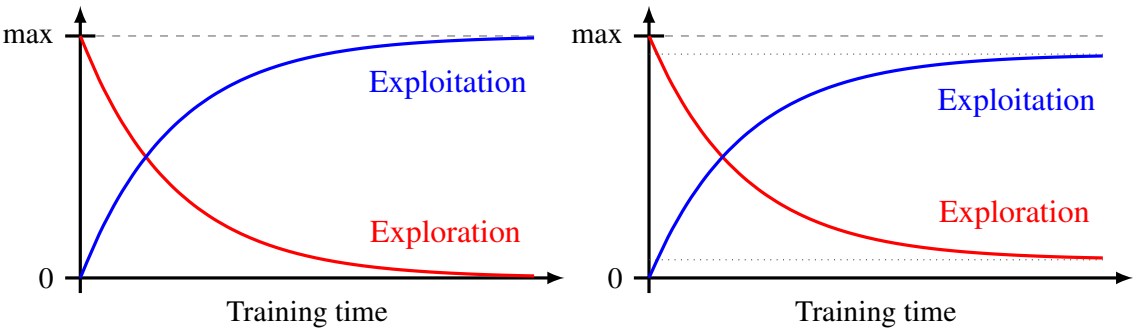
Figure 1.3.: The computation of actions with an actor network illustrated for a two-dimensional continuous action space.

Based on the example of a ball-on-plate task, Figure 1.2 illustrates how the framework of MDPs can be applied to generate robot trajectories in real time. The goal of the robot is to quickly follow a reference path while keeping a ball balanced on a plate. For online trajectory generation with model-free RL, each state $s \in \mathcal{S}$ typically contains information on the robot, the environment and the desired task. In this example, the robot is described by the kinematic state of its joints, while the environment is characterized by the current position and velocity of the ball on the plate. The task, on the other hand, is defined by future waypoints along the reference path.

At each time step t, the state s_t is mapped to an action a_t using a neural network with trainable parameters θ. While the decision process is discrete in time, robot motions are time-continuous. Therefore, the action a_t has to be mapped to a continuous motion from time step t to time step $t + 1$. While not strictly necessary, the time difference between t and $t + 1$ is typically constant. After executing the generated motion, the state s_{t+1} is reached and the procedure can be repeated. During the training phase, each action is rewarded based on its effectiveness in achieving the desired learning task. An algorithm from the realm of model-free RL can then be used to adjust the trainable parameters of the neural network so that well-performing actions are generated.

The goal of this thesis is to find a mapping from actions to motions that allows the robot to utilize its kinematic potential while ensuring that neither the robot nor its surroundings are damaged during the motion execution. The mapping should be applicable to a variety of tasks and should not overly restrict the speed or the workspace of the robot. In theory, the action space $\mathcal{A}$ of an MDP can be either discrete or continuous. To allow for a wide range of potential robot movements, a continuous action space is used in this work.

In order to gain a deeper understanding of how safety violations may arise during the training process of a neural network, Figure 1.3 illustrates the computation of actions when using an actor network to parameterize the policy π. While this thesis makes use of fully connected feedforward networks, the presented concepts can also be applied to other network architectures or policy representations. In the specific example shown in Figure 1.3, the actor network parameterizes a policy with a two-dimensional continuous action space using Gaussian distributions. Given the state s_t as input, the network outputs



(a) Final policy is deterministic.

(b) Final policy is stochastic.

Figure 1.4.: Typical courses of exploration and exploitation during the training phase of an actor network when using model-free RL.

a mean μ and a variance σ^2 for each dimension of the action space. These parameters specify Gaussian distributions from which the action a_t is sampled. The exploration of the environment is controlled by the variances of the Gaussian distributions.

At the beginning of a training phase, the network parameters θ are typically initialized at random. Consequently, random actions are generated. Without special safety precautions, random actions are likely to cause safety violations. As shown in Figure 1.4, the exploration of the environment decreases during the training phase while the exploitation of the knowledge already acquired increases. By defining a suitable reward function, the probability of safety violations can be reduced during the course of training. However, effectively avoiding undesired behaviors by adjusting the reward function might require a careful tuning process to avoid negative effects on other objectives of the desired task.

Once the training phase is finished, the network parameters θ are no longer adjusted. As shown in Figure 1.4, the final policy can be either deterministic or stochastic. In case of the actor network shown in Figure 1.3, a deterministic policy can be enforced by setting the variance σ^2 of each action dimension to zero. Consequently, a state s_t is always mapped to the same action a_t. While many robotic tasks benefit from deterministic policies, there are also cases where a stochastic policy is preferred. For instance, a humanoid robot might appear more human and less robotic if it moves in a slightly stochastic way. Although stochastic policies are more likely to generate unsafe actions, safety violations can also occur when following a deterministic policy. During the training phase, the RL algorithm aims to find network parameters θ that maximize the *average* task performance. This optimization procedure, however, does not guarantee a good performance in every single state of the system.

The probability of safety violations might also increase when deploying a policy in an environment that differs from its training environment. A common example for such a domain change is a so-called *sim-to-real transfer*, visualized in Figure 1.5. Generating training data for model-free RL is often easier and faster when using a physics simulator. However, due to modeling errors, the real world differs from the simulation environment. This effect is known as *sim-to-real gap*. When taking protective measures to prevent safety violations in the real world, a further aspect that must be taken into consideration is the real-time capability of the required calculations.

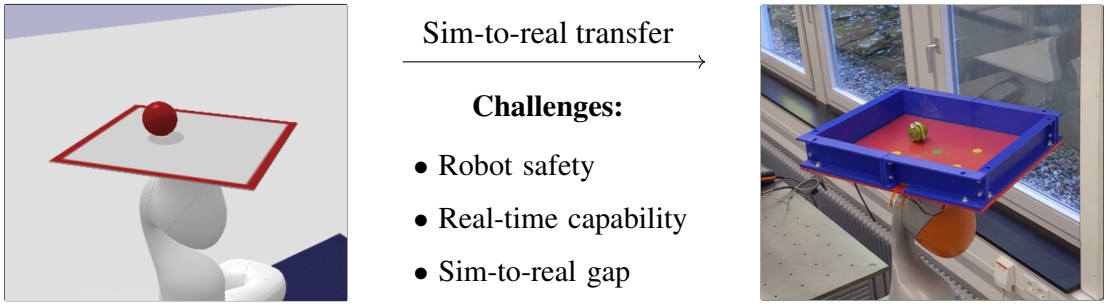


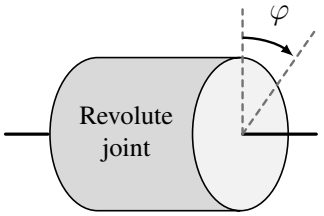
Figure 1.5.: A domain change from a physics simulator to the real world, known as *sim-to-real transfer*, might increase the probability of safety violations.

In summary, ensuring motion safety requires protective measures during and after the training phase. The mapping from actions to motions must be applicable to every potential action $a \in \mathcal{A}$ as random actions are selected at the beginning of the training phase. When collecting training data in the real world or when conducting a sim-to-real transfer, all calculations need to be real-time capable.

1.1.2. Incorporation of safety constraints

This part of the problem description specifies the safety constraints that are relevant within the scope of this thesis and introduces the framework of *constrained Markov decision processes* (CMDPs) [4]. In addition, different levels of constraint satisfaction are introduced.

Common robotic manipulators are composed of multiple individual parts, known as links, which are connected in series via revolute or prismatic joints. Revolute joints allow a robot to rotate one link against another along a common axis. The angle of rotation φ is the only *degree of freedom* (DOF) of a revolute joint. In this thesis, the following kinematic constraints are considered for revolute joints:



$$\mathrm{p}_{\min} \leq \varphi \leq \mathrm{p}_{\max} \tag{1.3}$$
$$\mathrm{v}_{\min} \leq \dot{\varphi} \leq \mathrm{v}_{\max} \tag{1.4}$$
$$\mathrm{a}_{\min} \leq \ddot{\varphi} \leq \mathrm{a}_{\max} \tag{1.5}$$
$$\mathrm{j}_{\min} \leq \dddot{\varphi} \leq \mathrm{j}_{\max}, \tag{1.6}$$

where p, v, a, and j stand for position, velocity, acceleration, and jerk, respectively.

Note that the symbols of the kinematic limits are rendered upright to distinguish accelerations a from actions a. Compliance with the kinematic limits is important to avoid excessive stress to the robot joints. The maximum values for each joint are typically specified by the robot manufacturers. Some industrial robots are equipped with a continuous revolute joint at the end of their kinematic chain to enhance the flexibility of an attached tool. For continuous joints, the position limits specified in equation (1.3) do not apply. In the literature, jerk constraints are not always taken into account. Considering jerk constraints, however, can help to reduce vibrations, increase the lifespan of mechanical components and improve the tracking accuracy of trajectory controllers [46, 87, 108]. The kinematic constraints (1.3) - (1.6) can be applied analogously to prismatic joints, which

enable a translational movement between two robot links. One example is the prismatic joint used to adjust the shoulder height of the humanoid robot ARMAR-6 [12].

Typical robot joints are not only subject to kinematic constraints, but also to dynamic constraints. For revolute joints, the torque τ is limited as follows:

$$\tau_{\min} \leq \tau \leq \tau_{\max} \tag{1.7}$$

The force of a prismatic joint is constrained analogously. For the sake of simplicity, the term *torque* is used for both revolute joints and prismatic joints in this thesis.

Apart from joint constraints, collisions are an important cause for safety violations. In this work, three different types of collisions are considered:

- **Self-collisions** refer to collisions between a moving part of a robot and another part of the robot, which can be either moving or static. If multiple robots are controlled by the same instance, collisions between the robots can also be considered as self-collisions.
- **Collisions with static obstacles** are collisions between a moving part of a robot and an immovable obstacle.
- **Collisions with moving obstacles** describe a collision between a robot and a moving obstacle. If two robots are controlled independently, each robot can regard the other robot as a moving obstacle. Similarly, humans can be considered as moving obstacles.

In the following, d_{self}, d_{static}, and d_{moving} denote the closest distance to a self-collision, to a collision with a static obstacle, and to a collision with a moving obstacle, respectively. Collisions do not occur if the following conditions are fulfilled at all times:

$$d_{\text{self}} > 0 \tag{1.8}$$
$$d_{\text{static}} > 0 \tag{1.9}$$
$$d_{\text{moving}} > 0 \tag{1.10}$$

In practice, it can be reasonable to enforce certain safety distances:

$$d_{\text{self}} > d_{\text{safety}_{\text{self}}} \tag{1.11}$$
$$d_{\text{static}} > d_{\text{safety}_{\text{static}}} \tag{1.12}$$
$$d_{\text{moving}} > d_{\text{safety}_{\text{moving}}} \tag{1.13}$$

This way, it is possible to compensate for errors that might be caused by inaccurate geometric models or by a time-discrete computation of the distances d_{self}, d_{static}, and d_{moving}.

Using the framework of *constrained Markov decision processes* (CMDPs) [4], safety constraints can be formally incorporated into the learning process. In the literature, a distinction is made between cumulative constraints and instantaneous constraints [99, 190]. Cumulative constraints enforce the sum of a constraint cost received over time to be smaller than a specified threshold. In contrast, instantaneous constraints must always be satisfied. The safety constraints considered in this work are instantaneous constraints. Moreover, it is important to emphasize that time-continuous motions must adhere to the safety constraints at all times, not just at the discrete time steps of the underlying MDP.

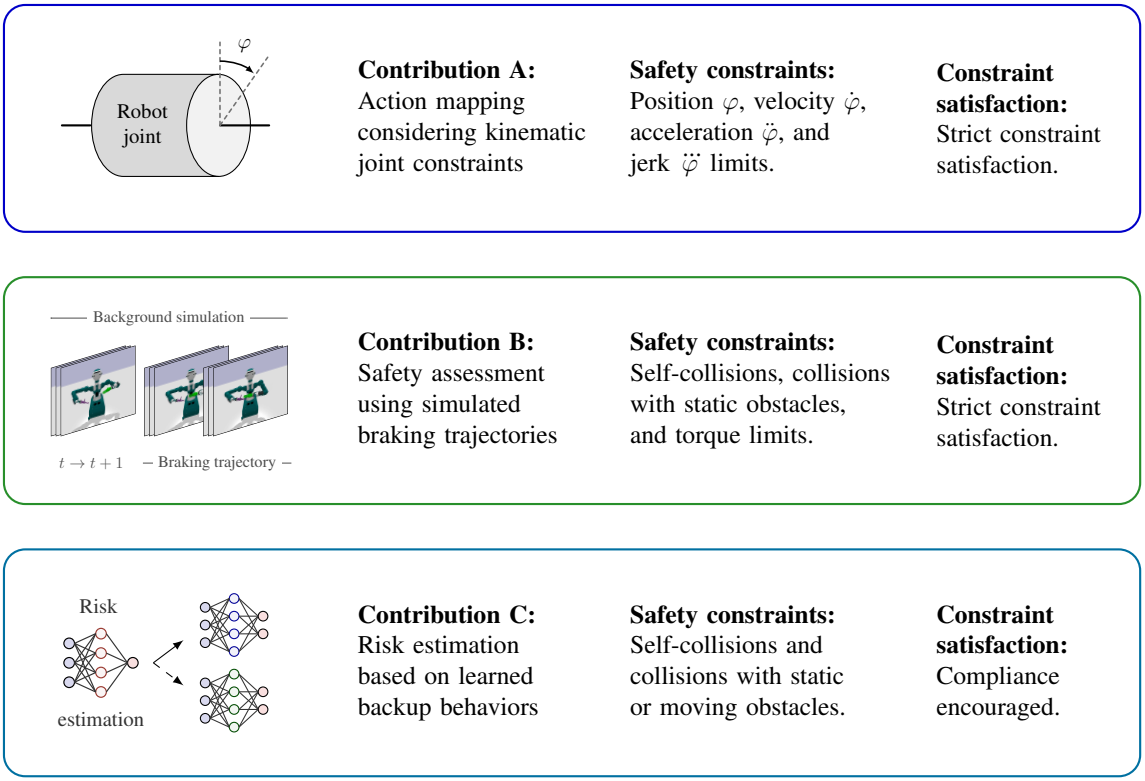


Figure 1.6.: An overview of the main contributions presented in this thesis.

In the literature on CMDPs, another distinction is made with respect to the level of constraint satisfaction [25]. Hard constraints guarantee strict constraint satisfaction. A probabilistic constraint is met with a certain probability $\in [0, 1]$. Soft constraints, on the other hand, encourage constraint satisfaction without providing any guarantees. In the context of motion safety, there is typically a strong preference for hard constraints. However, providing strict safety guarantees requires certain assumptions about the future evolution of the environment. This work aims to identify conditions under which motion safety can be guaranteed. In environments where these conditions do not apply, compliance with the safety constraints should be encouraged.

1.2. Contributions

This thesis proposes methods to enhance motion safety when learning robot trajectories using model-free RL. As can be seen in Figure 1.6, the main contributions of this work are divided into three parts, referred to as *contribution A*, *contribution B*, and *contribution C*. An exemplary robot scenario for each contribution is shown in Figure 1.7.

- **Contribution A** describes a mapping from actions to motions which ensures that the resulting trajectory setpoints satisfy the kinematic joint constraints given by the equations (1.3) - (1.6). Using the action mapping, a robot can be safely operated close to its kinematic limits. In order to additionally account for collisions, *contribution A* can be combined with *contribution B* or *contribution C*.

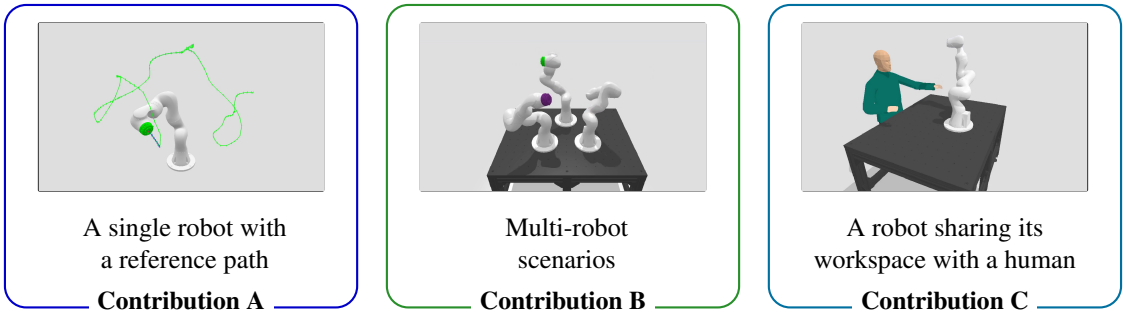


Figure 1.7.: The figure shows exemplary environments in which the methods presented in this thesis can be used to prevent safety violations.

- **Contribution B** introduces a method to ensure motion safety by simulating the execution of braking trajectories. Under certain conditions, this method is able to completely prevent self-collisions and collisions with static obstacles. As shown in Figure 1.7, multi-robot scenarios are a possible case of application for this contribution.
- **Contribution C** makes use of learned backup behaviors to additionally consider collisions with moving obstacles. First, a backup policy is trained to actively avoid safety violations. Subsequently, a data-based risk estimator is employed to quickly assess the risk associated with a specific motion. Using this method, compliance with the safety constraints is encouraged but not strictly guaranteed.

In the following, each of the contributions is described in more detail.

1.2.1. Action mapping considering kinematic joint constraints

In order to generate robot trajectories based on the framework of MDPs, actions $a \in \mathcal{A}$ need to be mapped to feasible robot motions. One contribution of this thesis is an action mapping which ensures that the kinematic joint limits defined by the equations (1.3) - (1.6) are satisfied. While the method is comprehensively described in chapter 3, a brief summary is given below. An action a_t is used to define a robot trajectory from time step t to time step $t+1$. At each time step t and for every joint, a continuous range of acceleration setpoints $[\mathrm{a}_{t+1_{\min}}, \mathrm{a}_{t+1_{\max}}]$ is computed. Knowing this range, a valid trajectory from t to $t+1$ can be generated by linearly connecting the current joint acceleration a_t with any acceleration $\mathrm{a}_{t+1} \in [\mathrm{a}_{t+1_{\min}}, \mathrm{a}_{t+1_{\max}}]$. Based on the action a_t, the desired acceleration setpoint $\mathrm{a}_{t+1} \in [\mathrm{a}_{t+1_{\min}}, \mathrm{a}_{t+1_{\max}}]$ is specified. The proposed action mapping not only ensures that the kinematic constraints are fulfilled within the time interval from t to $t+1$, but also that at least one feasible trajectory exists for each future point in time. This concept is sometimes paraphrased as reasoning over an *infinite time horizon* [40]. The effectiveness of the technique is evaluated by learning policies for various robot tasks. For instance, reference trajectories are adjusted so that a ball is balanced on a plate. By conducting a sim-to-real transfer, it is shown that the method can be used to generate well-performing trajectories for real industrial robots. In order to consider further safety constraints, the action mapping can be combined with one of the contributions outlined in the following.

1.2.2. Safety assessment using simulated braking trajectories

Motion safety can be ensured if at least one feasible way to continue a robot trajectory is known at all times. Provided that there are no moving obstacles in the environment and that the base of a robot is stable, collisions can no longer occur if all robot joints are brought to a standstill. With these considerations in mind, this thesis presents a method to prevent safety violations during and after the training of a policy by ensuring that a robot can be safely stopped at all times. A comprehensive explanation of the technique is provided in chapter 4. In short, a trajectory from t to $t+1$ is computed at time step t using an action generated by a neural network. Before executing the corresponding movement, a subsequent braking trajectory is computed. The resulting trajectory from t to the time at which the robot reaches a standstill is checked for safety violations using a physics simulator. If no safety violation is detected, the movement resulting from the selected action is carried out. Otherwise, a braking trajectory is executed to bridge the time from t to $t+1$. As the braking trajectory has already been checked for safety violations at an earlier time step, it can be executed safely. In this thesis, the method is used to avoid collisions and violations of torque limits. As the action mapping from *contribution A* is utilized, the kinematic constraints given by the equations (1.3) - (1.6) are also taken into account. By learning a reaching task, the approach is evaluated for environments with up to three industrial robots and for a humanoid robot with a stable base. The computing power required for the background simulations depends on the duration of the braking trajectories. Since the time needed to stop a robot is usually short, the computational effort remains moderate. As demonstrated by a successful sim-to-real transfer with an industrial robot, the required background simulations can be performed in real time. If moving obstacles are present, the method cannot be applied since collisions may occur even if the robot is stopped. However, as outlined below, this limitation can be addressed by utilizing learned backup behaviors instead of braking trajectories.

1.2.3. Risk estimation based on learned backup behaviors

The basic idea of this contribution is related to the safety assessment based on simulated braking trajectories described above. However, instead of relying on braking trajectories, optimized backup trajectories are used. The backup trajectories are computed by a second RL policy trained to generate movements that do not lead to safety violations. This way, it is possible to consider additional safety constraints. Specifically, the technique is employed to avoid collisions in environments with moving obstacles. Details on the implementation are given in chapter 5. Compared to braking trajectories, the backup trajectories used to avoid moving obstacles are significantly longer. As a result, more computational power is required to conduct background simulations in a physics simulator. In addition, moving obstacles may behave stochastically. For that reason, a single background simulation is no longer sufficient to correctly assess the risk of a particular action. In order to reduce the computational effort, a neural network is employed as a data-based risk estimator. Based on the proposed technique, safety violations can be effectively reduced right from the start of a training process. However, due to the prediction error of the risk estimator and the stochastic behavior of the environment, strict constraint satisfaction can no longer be guaranteed. In this thesis, the technique is evaluated for a reaching task and for a basketball task. By performing a successful sim-to-real transfer, it is shown that the required calculations can be carried out in real time.

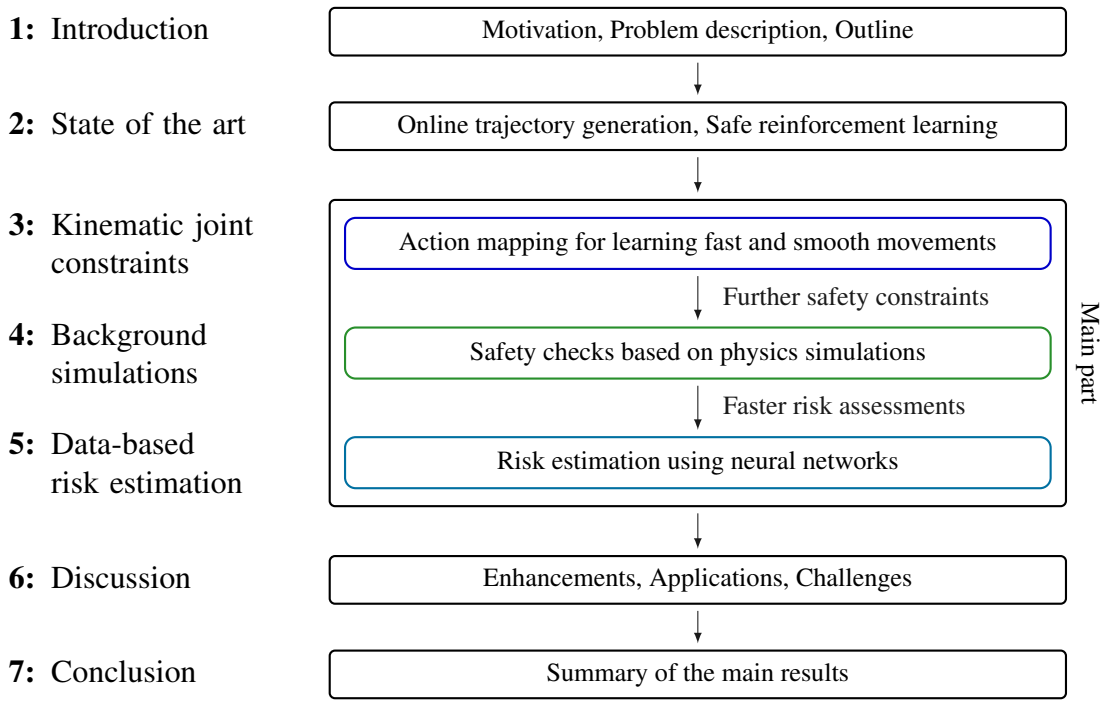


Figure 1.8.: The structure of this thesis.

1.3. Structure of the thesis

In the following, the structure of this thesis is outlined by briefly summarizing the content of each subsequent chapter. A graphical overview is provided in Figure 1.8.

Chapter 2 provides an in-depth analysis of the current state of the art and outlines previous studies related to the topic of this thesis. First, the chapter discusses various approaches for generating robot trajectories in real time. Subsequently, the focus is placed on model-free reinforcement learning, which serves as the foundation for this thesis. The second part of the chapter categorizes and compares techniques related to the field of safe reinforcement learning. In the analysis, particular attention is given to methods suitable for avoiding safety violations during the entire training process of a policy.

Chapter 3 delves into the problem of learning robot trajectories under consideration of kinematic joint constraints. Specifically, a mapping technique is presented which ensures that all possible actions of a policy are translated into constraint-satisfying trajectories. By utilizing the presented action mapping, fast and dynamic robot movements can be learned without overloading the robot joints. Therefore, the action mapping is used for all subsequent learning experiments in this thesis.

Chapter 4 introduces the concept of performing background simulations to maintain safety during the training process of a task policy. The purpose of the background simulations is to identify potentially unsafe actions, which are adjusted using a backup policy. In order to avoid collisions and torque limit violations in environments without moving obstacles, a backup policy based on braking trajectories is proposed. Subsequently, the concept is extended to environments with moving obstacles. For that purpose, a backup policy trained to actively avoid collisions is employed.

Chapter 5 introduces data-based risk estimators as a measure to reduce the computational effort associated with background simulations. First, the backup policies from the preceding chapter are employed to produce training data. Subsequently, neural networks are trained to detect risky actions using supervised learning. Benefiting from the low resource requirements of the neural networks, it is shown that safely executable trajectories can be generated in real time.

Chapter 6 discusses the contributions of the thesis, highlighting possibilities for improvement, potential application areas, and remaining challenges.

Chapter 7 concludes the thesis by emphasizing and summarizing the most important results of the work.

2. State of the art and related work

This chapter provides an overview of the state of the art by outlining various approaches and concepts that are related to the topic of this thesis. The first part of this chapter focuses on the aspect of generating robot trajectories during motion execution. The second part addresses the specific problem of ensuring safety when using model-free reinforcement learning.

2.1. Online generation of robot trajectories

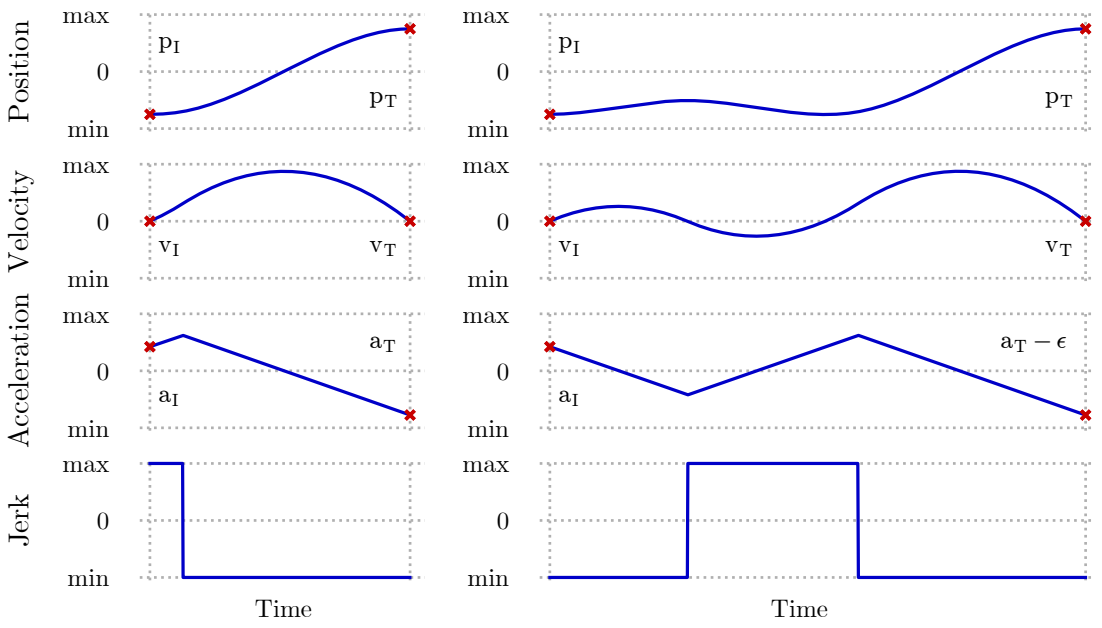
Robot trajectories can either be computed offline or online. When calculating trajectories offline, the entire movement of a robot is known before execution. In offline scenarios, the calculation time for a trajectory can exceed the duration of the movement. In addition, the trajectory can be checked for safety violations in advance. An example for offline trajectory optimization based on reinforcement learning is [81]. This thesis focuses on the problem of generating robot trajectories in real time. Compared to offline techniques, online trajectory generation makes it possible to consider sensory feedback and to adjust the objective of the robot during motion execution. For example, if a robot balances a ball, it is possible to react to the current position of the ball. Similarly, when the goal of a robot is to follow a reference path, the reference path can be adjusted during motion execution. As the resulting movements are not fully known in advance, avoiding safety violations is challenging. In order to execute motions on a real robot, another aspect that must be taken into account is the real-time capability of the required calculations.

In the following, different techniques to compute robot trajectories during motion execution are outlined. First, the computation of time-optimal trajectories to a kinematic target state is discussed. Subsequently, methods from the fields of model-based control, learning from demonstration, and reinforcement learning are presented.

2.1.1. Time-optimal trajectories to a kinematic target state

A common problem in robotics is to compute a time-optimal trajectory from an initial kinematic state to a target state considering velocity, acceleration, and jerk constraints as defined by (1.4) - (1.6). For this specific problem, trajectories can be efficiently computed in real time [102, 55, 24, 87]. Two exemplary trajectories for a single robot joint are shown in Figure 2.1. In the figure, the kinematic states are denoted as tuples $(\mathrm{p}, \mathrm{v}, \mathrm{a})$, where p, v, and a stand for position, velocity, and acceleration, respectively.

To generate time-optimal trajectories, existing software libraries can be used. For example, the *Reflexxes* motion library [86] can be used to compute trajectories to a target



(a) Target state (p_T, v_T, a_T)

(b) Target state (p_T, v_T, $a_T - \epsilon$)

Figure 2.1.: Two trajectories starting from the same initial kinematic state (p_I, v_I, a_I) computed using *Ruckig* [18]. Adjusting the target acceleration a_T by a small value ϵ can have a strong influence on the generated trajectory.

state specified by a target position and a target velocity. The library also provides different techniques to synchronize trajectories for multiple robot joints. A more recent library called *Ruckig* [18] additionally supports target accelerations as part of the target state. While the libraries simplify the calculation of trajectories to a target state, the problem of finding suitable target states for a desired task is left to the user. For point-to-point movements and for braking trajectories, the specification of a target state is straightforward as the target velocity and the target acceleration can be set to zero. Potential collisions, however, are not taken into account by the libraries. For one-dimensional paths, it is possible to compute intermediate target states that lead to a fast traversal of the path [79]. A potential approach to find suitable target states for more complex tasks is reinforcement learning. However, when learning target states, three variables need to be specified per robot joint. Moreover, small variations of the target state can lead to significantly different trajectories. This effect is demonstrated in Figure 2.1 by subtracting a small value $\epsilon > 0$ from the target acceleration a_T. Similar to the aforementioned motion libraries, the approach presented in this thesis ensures that the resulting trajectories do not violate kinematic joint limits. However, only one variable per robot joint is required to specify the next time interval of a trajectory.

2.1.2. Model-based control

Robot movements are typically carried out using a two-stage approach. First, setpoints for a lower-level controller are computed considering the objectives of a desired robot task. Subsequently, the torque generated by each motor of the robot is adjusted so that the desired setpoints are tracked. Techniques from model-based control can be used for both purposes. In the first part of this section, common lower-level control strategies

are introduced. The second part focuses on a technique called *optimal control*, which can be used to generate robot trajectories that are optimal with respect to a predefined cost function.

2.1.2.1. Control strategies for lower-level motion control

In the field of robotics, several lower-level control strategies for robotic manipulators are well-established. The control concepts outlined below are described in the textbooks [101] and [151], to which the reader is referred for further details. From a mathematical point of view, motion control relies on the dynamic model of robotic manipulators:

$$\tau = M(\varphi)\,\ddot{\varphi} + C(\varphi, \dot{\varphi})\,\dot{\varphi} + \tau_g(\varphi) + \tau_{ext} \tag{2.1}$$

In this equation, $M(\varphi)$ is a positive-definite mass matrix, $C(\varphi, \dot{\varphi})\,\dot{\varphi}$ is used to model centrifugal and Coriolis forces, $\tau_g(\varphi)$ represents the impact of gravity and τ_{ext} models the influence of external forces. The vector τ stands for the torque produced by the motors of the robot, while the vector φ represents the joint positions. As can be seen from equation (2.1), the joint positions φ are influenced by the motor torques τ. Consequently, it is possible to control the motions of a robot by specifying torque setpoints τ_d for a torque controller. Note that the subscript $_d$ is used to distinguish setpoints from actual values. Many robots utilize electric motors to produce torque. In this case, a torque setpoint corresponds to a desired motor current. The motor current can be controlled by adjusting the voltage applied to the motor, which is generated by a power electronics unit.

In the context of *end-to-end* learning, learning techniques have been successfully employed to generate torque setpoints for a desired robot task. *End-to-end* learning aims at combining perception and control by learning a direct mapping from high-dimensional input signals to low-level control outputs. Successful *end-to-end* learning from pixels to torques has been demonstrated using model-based RL [177, 92] and model-free RL [97]. However, learning torque setpoints can have a negative effect on the task performance when transferring a policy trained in simulation to a real robot. Although the dynamics of robotic manipulators are well-understood, accurately simulating the effects of motor torques is difficult. For example, the mass matrix M is often approximated since exact inertia data is not accessible. In addition, common physics simulators use simplifications, e.g., when modeling friction, which introduce further discrepancies.

The problems mentioned above can be mitigated by making use of a *joint trajectory controller*. Instead of learning torques, trajectory setpoints $(\mathrm{p}_d, \mathrm{v}_d, \mathrm{a}_d) = (\varphi_d, \dot{\varphi}_d, \ddot{\varphi}_d)$ are generated. Based on these setpoints, a model-based trajectory controller computes torque setpoints τ_d that reduce the tracking error with respect to the desired trajectory. A straightforward approach for this task is the use of a *proportional–integral–derivative controller* (PID controller) [151]:

$$\tau_d = \mathrm{K_P}\,(\varphi_d - \varphi) + \mathrm{K_I} \int (\varphi_d - \varphi)\; dt + \mathrm{K_D}\,(\dot{\varphi}_d - \dot{\varphi}) \tag{2.2}$$

$\mathrm{K_P}$, $\mathrm{K_I}$ and $\mathrm{K_D}$ are positive-definite gain matrices that can be tuned to adjust the behavior of the controller. While PID controllers are easy to implement, a better tracking performance can by achieved by using a *computed-torque-like controller* [126, 136, 151]. In

the following equation of the controller, the circumflex ˆ indicates that the corresponding parameters are estimated rather than precisely known:

$$\tau_d = \hat{M}(\varphi) \left(\ddot{\varphi}_d + \mathrm{K_P} \left(\varphi_d - \varphi \right) + \mathrm{K_I} \int \left(\varphi_d - \varphi \right) \, dt + \mathrm{K_D} \left(\dot{\varphi}_d - \dot{\varphi} \right) \right) + \hat{C}(\varphi, \dot{\varphi}) \, \dot{\varphi} + \hat{\tau}_g(\varphi) \tag{2.3}$$

In addition to learning trajectory setpoints in joint space, it is also possible to specify robot motions in Cartesian space [90, 72]. For example, the framework of *operational space control* (OSC) [100, 75] can be used to control a robot with respect to the Cartesian position and orientation of its end-effector. While some robotic tasks can be described more intuitively in Cartesian space, learning trajectories in joint space provides the benefit that neither kinematic redundancies nor singularities need to be explicitly addressed.

2.1.2.2. Trajectory generation based on optimal control

Given a model of a dynamic system, the goal of *optimal control* is to find a trajectory that is optimal with respect to a predefined cost function J. Historically, major scientific contributions to the field of optimal control have been made by Richard Bellmann [15] and Lev Pontryagin [149]. In the following, the problem of optimal control is described mathematically. The notation is adopted from a survey paper by Mayne et al. [110]. In optimal control, a state $x_t \in \mathcal{X}$ defines the current state of a dynamic system, while a control input $u_t \in \mathcal{U}$ can be used to influence the future development of the system. The impact of the control variable on the state is described by a model f. For the time-discrete case, this relation can be expressed as follows:

$$x_{t+1} = f(x_t, u_t) \tag{2.4}$$

A trajectory is a sequence of states $\{x_t, x_{t+1}, \ldots, x_N\}$ resulting from a control sequence $\{u_t, u_{t+1}, \ldots, u_{N-1}\}$. Given an initial state x_t, the cost function J can be used to compute a cost value for each control sequence $\{u_t, u_{t+1}, \ldots, u_{N-1}\}$. The goal of the optimal control problem is to find a control sequence that minimizes the resulting cost.

A special case of optimal control, where the system dynamics are linear and the cost function has a quadratic form, is called *linear-quadratic optimal control* [172]. In this case, an optimal control law can be derived analytically based on *Riccati equations* [70]. The resulting controller is called *linear–quadratic regulator* (LQR). For linear-quadratic problems, it is possible to compute trajectories that are optimal with respect to an infinite time horizon. In the case of more general dynamics and cost functions, iterative approaches can be employed to find approximate solutions for finite time horizons [94, 170].

Since the system dynamics are known, optimal trajectories can be computed in an open loop without using feedback. In practice, however, the system behavior may deviate from the model due to modeling errors and external disturbances. The control loop can be closed by measuring the current state of the system and computing a consecutive optimal trajectory with a finite time horizon at each discrete time step. This approach is known as *model predictive control* (MPC) [110]. Using MPC, it is possible to compute robot movements during motion execution. Typical algorithms to approximate optimal trajectories for non-linear dynamics include *iterative LQR* (iLQR) [94], *iterative*

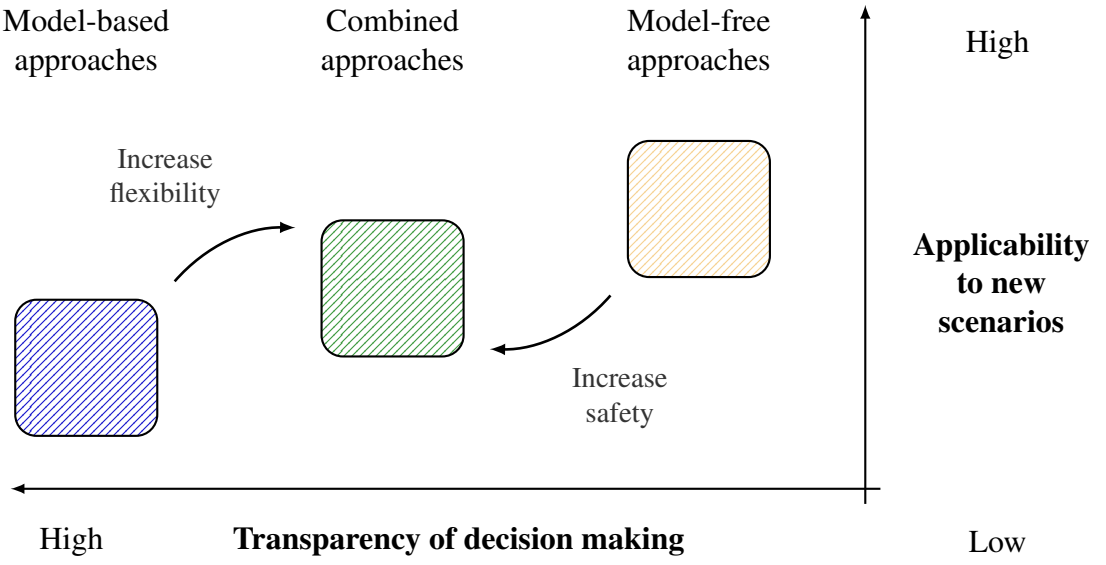


Figure 2.2.: Model-based approaches and model-free approaches have distinct strengths and weaknesses. Combined approaches attempt to narrow the gap between both concepts.

linear–quadratic–Gaussian control (iLQG) [170] and *differential dynamic programming* (DDP) [109]. It is also possible to utilize general-purpose solvers for non-linear optimization problems such as *sequential quadratic programming* (SQP) [182, 48] or *interior point methods* [41, 175]. In this case, it is straightforward to additionally consider equality and inequality constraints. However, general-purpose solvers tend to be less computationally efficient, which is unfavorable since MPC must be performed in real time. When using DDP, extensions are available to ensure that the control input u stays within predefined limits [165, 107]. Alternatively, lower-level controllers can be used to encourage constraint satisfaction [73]. One approach is to utilize cascades of *quadratic programs* (QPs), a special class of optimization problems for which efficient solvers are available [33, 59]. Since this approach does not ensure the existence of a solution, different priorities are assigned to the individual constraints. In [59], a high priority is given to torque constraints and kinematic joint limits, while lower priorities are assigned to objectives like motion tracking or balancing. If no solution can be found for the complete control problem, constraints with a low priority are disregarded. A similar approach is taken by the *stack-of-tasks* framework [105], which has been successfully used for various applications, including collision avoidance with a humanoid robot [158].

When comparing optimal control with the problem formulation of model-free RL presented in section 1.1.1, several similarities can be identified. While the notation differs, both approaches utilize the concept of states. Similarly, the control input u in optimal control corresponds to actions a in model-free RL. In addition, both methods aim to find a control policy that is optimal with respect to a cost function or a reward function. However, as shown in Figure 2.2, there are also fundamental differences between model-based and model-free approaches. Most importantly, model-free approaches do not require to specify a model f. In addition, the reward function $R(s, a)$ in model-free RL does not need to be differentiable with respect to the state s or action a. Consequently, model-free approaches offer a high degree of flexibility with regard to the target scenario. Model-based approaches, on the other hand, are less flexible but more transparent in terms of the decision-making process. In particular, it is possible to formally verify safety properties,

e.g., based on Hamilton-Jacobi reachability analysis [14] or control barrier functions [6]. Combined approaches attempt to address the shortcomings of purely model-based or purely model-free approaches. One line of research focuses on increasing the flexibility of model-based approaches by incorporating techniques from machine learning [62]. If the system dynamics are not fully known in advance, a model consisting of a known and an unknown part can be used to realize MPC. The unknown part can be represented by a Gaussian process that is updated based on previously collected data [120, 61]. It is also possible to learn the system dynamics using neural networks [34, 133]. When utilizing MPC, the computational effort depends on the prediction horizon of the trajectory computed at each time step [191]. By learning a value function that estimates a final cost, it is possible to increase the prediction horizon of MPC while keeping the computational effort limited [191, 64]. Another line of research aims to enhance the safety of model-free approaches by incorporating model knowledge. A detailed review of previous studies in the field of *safe reinforcement learning* is provided in section 2.2.

2.1.3. Learning from demonstration

Learning from demonstration (LfD) is a machine learning technique that can be used to learn robot trajectories based on demonstrations from a human expert. Suitable demonstrations can be recorded using a motion capture system or a teleoperation system. Alternatively, it is possible to move the links of a robot by hand, a technique known as *kinesthetic teaching* [58]. When using demonstrations, it is not necessary to specify a model of the system dynamics or a task-specific optimization function.

In the literature, different mathematical representations have been employed to reconstruct robot trajectories from human demonstrations. A via-point representation is used in [112] to learn a tennis serve and in [176] to regenerate handwritten characters. Based on *hidden Markov model*s (HMMs), gestures are reproduced in [26], while dual-arm manipulation tasks for humanoid robots are addressed in [10]. *Dynamical movement primitive*s (DMPs) [65, 66] are non-linear attractor systems that can be used to learn control policies for discrete and periodic movements. In [65], DMPs are employed to learn a tennis swing and a periodic drumming task based on human demonstrations. The spatial and temporal invariance of DMPs makes it possible to adjust the goal position of the tennis swing or the frequency of the drumming movement. Based on a reaching task and a ball throwing task, the generalization abilities of DMPs are further investigated in [173]. Obstacles can be avoided by adding an additional perturbation term to the mathematical formulation of DMPs [123, 124], e.g., based on artificial potential fields [74]. In [124], skills like grasping, placing, and releasing are learned using a slightly modified DMP formulation. The skills can be parameterized with respect to their start position, end position, and duration. By composing multiple skills sequentially, more complex tasks can be performed. Further motion representations include *probabilistic movement primitive*s (ProMPs) [121, 122] to model trajectory distributions based on stochastic movements and *via-points movement primitive*s (VMPs) [193, 192] to explicitly consider intermediate waypoints. By combining neural networks with movement primitives, it is possible to utilize the representational power of neural networks while preserving the theoretical properties of movement primitives. Given an image of a digit as input, Gams et al. [42] train a neural network to output a DMP for writing the digit with a robot. In [192], a mixture density network [20] is used to parameterize VMPs for a ball throwing task shown in Figure 2.3. Instead of utilizing

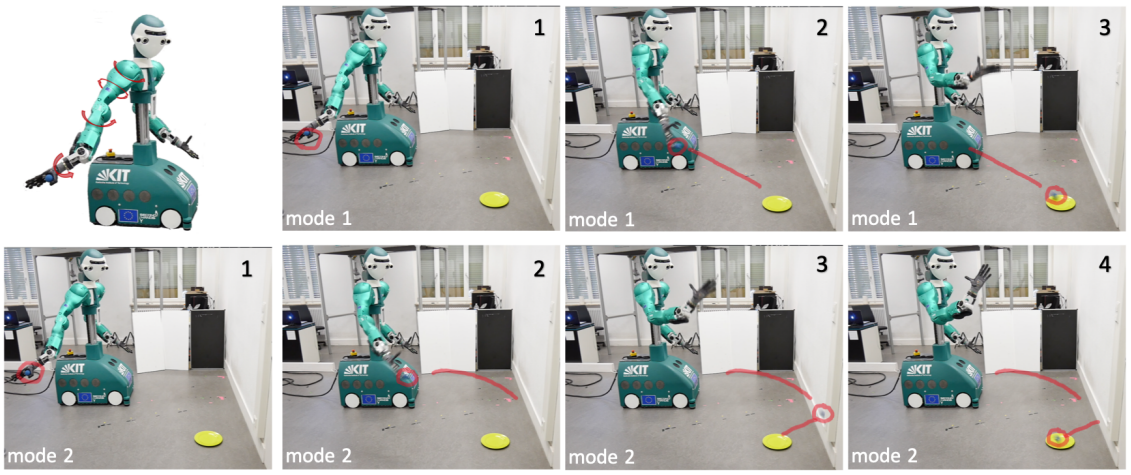


Figure 2.3.: Based on human demonstrations, a ball throwing task with a humanoid robot is learned in [192]. A mixture density network is used to parameterize suitable movement primitives. The network can encode multiple ways of completing the task, which are referred to as *modes*. Two exemplary *modes* are shown in the figure, which is taken from [192].

movement primitives, it is also possible to train a policy that directly maps states to actions [188, 39, 38]. This approach is known as *behavioral cloning*. Appropriate actions can be recorded using a teleoperation system [39, 38] or virtual reality devices [188].

Compared to model-free RL, learning from demonstration offers the advantage that no reward function needs to be specified. Depending on the desired task, however, providing suitable demonstrations can be challenging. When recording human demonstrations with a motion capturing system, a *retargeting* process is required to map the limbs of a human to the links of a robot. Due to differences in the kinematic structure, a movement that works well for a human is not necessarily well suited for a robot. In the context of *behavioral cloning*, policies trained based on expert demonstrations tend to be poor at recovering from errors [139, 140]. Once the policy makes a mistake, the robot might encounter an area of the state space that was not part of the training data. As a result, errors tend to compound rather than being corrected by the policy. The aforementioned problems can be avoided by making use of model-free RL. In this case, human demonstrations can be used to find a suitable reward function via *inverse reinforcement learning* [9]. Alternatively, the training process of model-free RL can be accelerated, e.g., by priming the policy π or a state-value function V based on data collected from a human expert [145, 60].

2.1.4. Reinforcement learning

Techniques from the field of *reinforcement learning* (RL) can be divided into two categories: *Model-based* RL and *model-free* RL. In the first part of this section, methods related to model-based RL are briefly outlined. The second part takes a detailed look at model-free RL, which serves as the basis for the methods proposed in this thesis.

2.1.4.1. Model-based reinforcement learning

The term *model-based* RL refers to methods that utilize data from the environment to learn a model of the system dynamics. A comprehensive survey on model-based RL is provided by Moerland et al. [115]. An exemplary model-based algorithm for learning robot movements is called *guided policy search* (GPS) [91, 92]. The algorithm assumes that the system dynamics are locally linear. Based on previously collected trajectories, a model of the dynamics is fitted in an iterative process. Using the model, trajectories are generated by applying methods from optimal control. In a subsequent step, a neural network is trained to reproduce these trajectories. Compared to directly using optimal control, the neural network introduces the ability to generalize to new situations. In addition, the network can be trained to reproduce the trajectories while having limited access to sensory feedback. According to Moerland et al. [115], methods that fit a model first are typically more data-efficient than model-free techniques but tend to achieve a lower final performance.

The term model-based RL is sometimes also used to refer to methods that utilize model knowledge to learn a policy or value function. For example, AlphaZero [155], a technique to learn board games like Go or chess, is considered as a model-based RL algorithm. While the rules of these games are known, reinforcement learning is used to guide a *Monte Carlo tree search* (MCTS). In the context of *safe reinforcement learning*, model knowledge can be used to increase the level of safety during the learning process. Further details on this line of research are provided in section 2.2.

2.1.4.2. Model-free reinforcement learning

The basic idea of *model-free* RL is to learn a policy by trial and error without learning a model first. To this end, the learning problem is typically formalized as a *Markov decision process* $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, a mathematical framework introduced in section 1.1.1. A policy π with $\pi(a|s) = \Pr\{a_t = a \mid s_t = s\}$ can either be learned directly or derived from a value function. Based on a textbook by Sutton and Barto [160], both concepts are introduced below. In the context of model-free RL, the state-value function $V(s)$ and the action-value function $Q(s, a)$ are important expressions. The state-value function $V_\pi(s)$ describes the expected return G when starting in state s and following a policy π:

$$V_\pi(s) = \mathbb{E}_\pi\left[G_t \mid s_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k \cdot R(s_{t+k}, a_{t+k}) \,\middle|\, s_t = s\right] \tag{2.5}$$

The action-value function $Q_\pi(s, a)$ describes the expected return G when starting in state s, selecting action a and following policy π afterwards:

$$Q_\pi(s, a) = \mathbb{E}_\pi\left[G_t \mid s_t = s, a_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k \cdot R(s_{t+k}, a_{t+k}) \,\middle|\, s_t = s, a_t = a\right] \tag{2.6}$$

It is possible to express $Q_\pi(s, a)$ based on $V_\pi(s)$:

$$Q_\pi(s, a) = \mathbb{E}\left[R(s_t, a_t) + \gamma \cdot V_\pi(s_{t+1}) \mid s_t = s, a_t = a\right] \tag{2.7}$$

Similarly, $V_\pi(s)$ can be derived from $Q_\pi(s,a)$:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \cdot Q_\pi(s,a) \tag{2.8}$$

An optimal policy π_* leads to the highest possible expected return for all $s \in \mathcal{S}$. When following an optimal policy, the resulting optimal state-value function is denoted as $V_*(s)$. Correspondingly, the optimal action-value function is denoted as $Q_*(s,a)$. While model-free RL algorithms attempt to find an optimal policy, strict optimally is rarely achieved under real conditions [160]. In practice, finding a well-performing policy that approximates π_* is usually regarded as sufficient. There are various factors that affect which RL algorithm is suitable for a particular application. An important aspect to consider is whether the action space $\mathcal{A}$ is discrete or continuous. When learning Atari games [113], the number of possible actions is limited so that a discrete action space can be used. In contrast, the number of feasible robot movements is typically unlimited, making a continuous action space a natural choice. Nevertheless, it is also possible to discretize the range of feasible actions. For instance, in-hand manipulation with a robotic hand can be learned using discrete actions [8]. In the context of RL algorithms, another distinction is made between *on-policy* methods and *off-policy* methods. On-policy methods use data retrieved under the current policy for policy updates. Off-policy methods, on the other hand, can reuse previously collected data stored in a *replay buffer*. While the performance of on-policy algorithms tends to be more stable, off-policy methods are typically more sample-efficient [52]. The techniques presented in this thesis can be used in conjunction with any RL algorithm that supports continuous state spaces and continuous action spaces.

Off-policy algorithms Off-policy algorithms typically derive a policy by approximating the optimal action-value function $Q_*(s,a)$. This idea dates back to an algorithm called *Q-learning* introduced by Watkins [179] in 1989. For the specific case of the optimal action-value function $Q_*(s,a)$, equation (2.7) can be rewritten as:

$$Q_*(s,a) = \mathbb{E}\left[R(s_t,a_t) + \gamma \cdot V_*(s_{t+1}) \mid s_t = s, a_t = a\right] \tag{2.9}$$

According to Bellman [15], an optimal policy should select an optimal action in each state. Consequently, the optimal state-value function $V_*(s)$ can be expressed as follows:

$$V_*(s) = \max_{a \in \mathcal{A}} Q_*(s,a) \tag{2.10}$$

Inserting equation (2.10) into (2.9) leads to the so-called *Bellman equation* for $Q_*(s,a)$:

$$Q_*(s,a) = \mathbb{E}\left[R(s_t,a_t) + \gamma \cdot \max_{a_{t+1} \in \mathcal{A}} Q_*(s_{t+1},a_{t+1}) \,\middle|\, s_t = s, a_t = a\right] \tag{2.11}$$

The Bellman equation can be used to approximate $Q_*(s,a)$ in an iterative way [179, 178]:

$$Q_i(s_t,a_t) = (1-\alpha) \cdot Q_{i-1}(s_t,a_t) + \alpha \cdot \left[R(s_t,a_t) + \gamma \cdot \max_{a_{t+1} \in \mathcal{A}} Q_{i-1}(s_{t+1},a_{t+1})\right], \tag{2.12}$$

where α is called *learning rate* and i indicates the current iteration. As one of the breakthroughs in *deep reinforcement learning* (DRL), Mnih et al. [113, 114] proposed an algorithm called *deep Q-network* (DQN) that utilizes neural networks to approximate the optimal action-value function $Q_*(s,a)$. Based on the Bellman equation (2.11), it is possible

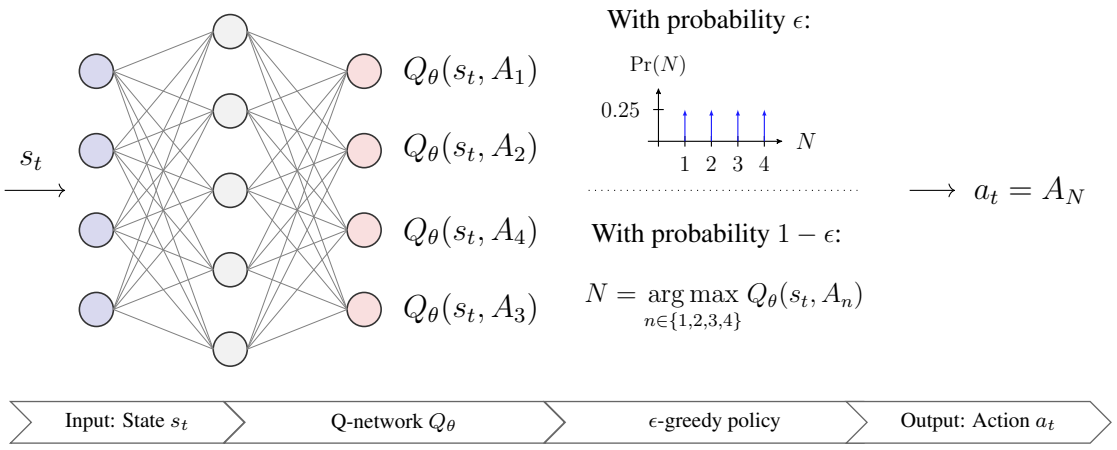


Figure 2.4.: The off-policy algorithm DQN can be applied to learning problems with a discrete action space such as $\mathcal{A} = \{A_1, A_2, A_3, A_4\}$. The algorithm utilizes an ϵ-greedy strategy to derive a policy based on the output of a neural network trained to approximate the optimal action-value function $Q_*(s, a)$.

to define a loss function L_θ that is differentiable with respect to the trainable parameters θ of a neural network Q_θ [114]:

$$L_\theta = \mathbb{E}\left[\left(R(s_t, a_t) + \gamma \cdot \max_{a_{t+1} \in \mathcal{A}} Q_T(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t)\right)^2\right] \tag{2.13}$$

In this equation, Q_T represents a second neural network, called *target network*, which is used to improve the stability of DQN. A loss function like (2.13) is also known as *mean-squared Bellman error* (MSBE). The expected value $\mathbb{E}$ is approximated by randomly sampling data from a *replay buffer*, which stores previously collected experiences $(s_t, a_t, R(s_t, a_t), s_{t+1})$. Using *gradient descent*, the network parameters θ of Q_θ are updated in an iterative manner so that the loss function (2.13) is minimized. The parameters of the target network are not updated during the gradient descent but copied from Q_θ at regular intervals. DQN is an algorithm for discrete action spaces. As shown in Figure 2.4, the neural network Q_θ receives a state s_t as input and outputs an approximated action-value for each discrete action. A policy is derived from the action-values using an ϵ-greedy strategy. More precisely, a random action is selected with a probability of $\epsilon \in [0, 1]$, while the action with the highest action-value is chosen otherwise. As a result, the value selected for ϵ controls the amount of exploration. For continuous action spaces, the network architecture from Figure 2.4 cannot be used as the number of actions is unlimited. Instead, a neural network Q_θ can be trained to output an approximate value for $Q_*(s, a)$ given a state s and an action a as input. In this case, however, finding the action with the highest action-value is no longer straightforward. The algorithm *deep deterministic policy gradient* (DDPG) [153, 97] addresses this issue by introducing an actor network, which maps a state s to a desired action a. The actor network is trained to output actions that maximize the action-values $Q(s, a)$ provided by the network Q_θ. While the mapping learned by the actor network is deterministic, the environment is explored by adding noise to the generated actions. Approaches that learn an actor policy and a value function approximation are called *actor-critic methods* [85]. In contrast to DDPG, the algorithm *soft actor-critic* (SAC) [52, 53] directly learns a stochastic actor policy. The exploration is controlled based on an objective function that not only considers the ex-

pected sum of future rewards but also the *entropy* $\mathcal{H}$ of the policy, which is a measure for its stochasticity.

On-policy algorithms In 1992, Williams [181] proposed an on-policy algorithm called *REINFORCE* that directly optimizes a policy without learning a value function first. The core idea of the algorithm is to estimate a *policy gradient* using data retrieved under the current policy. When representing the policy π by a neural network, the policy gradient provides an update direction for the network parameters θ that increases the expected return of the policy. In the literature, different ways to estimate the policy gradient g have been proposed [181, 85, 161]. According to Schulman et al. [147], there is usually a trade-off between the *bias* and the *variance* of policy gradient estimators. The higher the variance, the more data is required to estimate the policy gradient. Bias, on the other hand, leads to incorrect gradient estimates even if a large amount of data is provided. In order to reduce the variance of policy gradient estimators, modern on-policy algorithms typically approximate the *advantage function* $A_\pi(s, a)$ [161, 147]:

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s) \tag{2.14}$$

The advantage function $A_\pi(s, a)$ indicates whether taking action a in state s performs better or worse than following policy π. To estimate the advantage function, a neural network is usually trained to approximate the state-value function $V_\pi(s)$. Given the resulting approximation $\hat{V}(s)$, the following relation can be used to estimate the advantage of an action a_t [147]:

$$\hat{A}(s_t, a_t) = R(s_t, a_t) + \gamma \cdot \hat{V}(s_{t+1}) - \hat{V}(s_t) \tag{2.15}$$

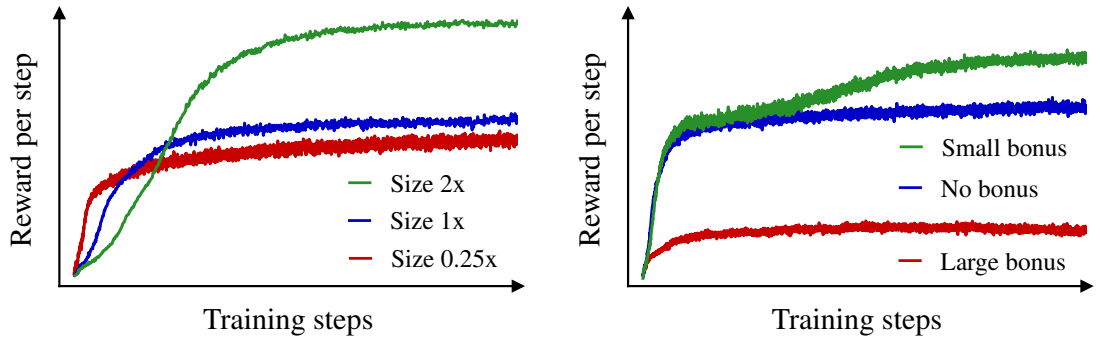
Based on $\hat{A}(s_t, a_t)$, it is possible to estimate the policy gradient as follows [148]:

$$\hat{g} = \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}(s_t, a_t)\right], \tag{2.16}$$

where ∇_θ indicates the computation of a gradient with respect to θ. The expected value $\mathbb{E}$ can be approximated using data sampled under the current policy π_θ, which is parameterized by an actor network with network weights θ. Modern RL algorithms like *trust region policy optimization* (TRPO) [146] and *proximal policy optimization* (PPO) [148] do not directly make use of equation (2.16) but introduce a surrogate objective function L_θ:

$$L_\theta = \mathbb{E}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\,\hat{A}(s_t, a_t)\right] \tag{2.17}$$

In equation (2.17), $\pi_{\theta_{\text{old}}}(a_t|s_t)$ represents the policy before updating the network parameters θ. To estimate the policy gradient g, the surrogate objective L_θ is differentiated with respect to θ. In practice, the gradient computation is typically carried out by a machine learning framework like *TensorFlow* [1] or *PyTorch* [125] using *automatic differentiation*. To ensure a stable and reliable learning process, each update of the network parameters should only change the policy to a limited extent [146, 148]. The difference between two probability distributions can be quantified based on their *Kullback–Leibler divergence* (KL divergence) [88]. In order to avoid large policy updates, TRPO maximizes the surrogate objective L_θ subject to a constraint on the approximated KL divergence between π_θ and $\pi_{\theta_{\text{old}}}$ [146]. Contrary to that, PPO utilizes an unconstrained surrogate



(a) Batch size **(b)** Entropy bonus

Figure 2.5.: Several hyperparameters influence the learning performance of RL algorithms like PPO. The figure shows the average reward per decision step during the training of a reaching task with an industrial robot when selecting different values for the batch size (a) and the entropy bonus (b).

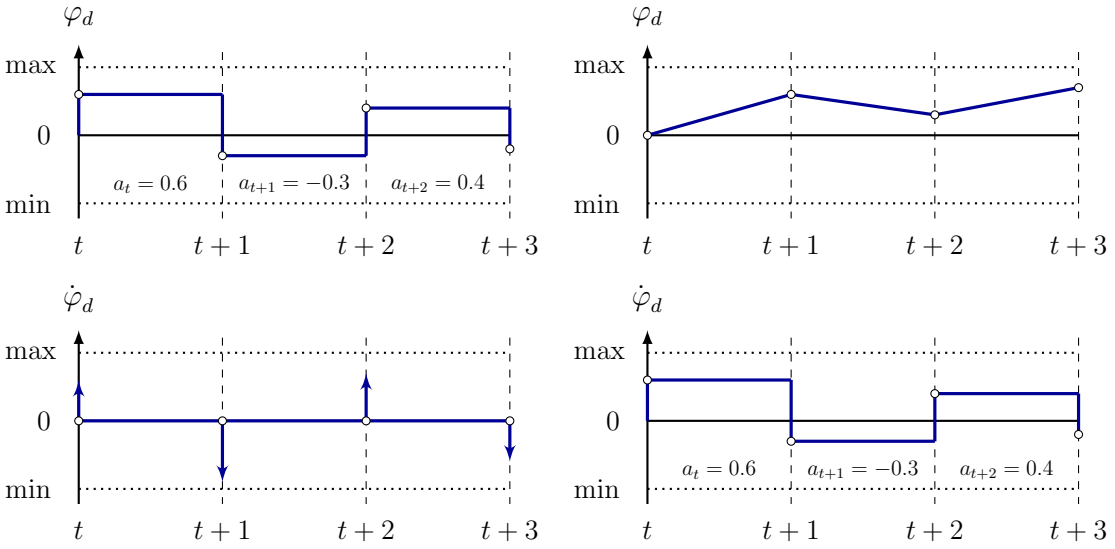
objective. However, large updates are discouraged by clipping the fraction $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ in equation (2.17) or by adding an adaptive penalty term based on the KL divergence [148].

In this thesis, the on-policy algorithm PPO is used because it offers a good compromise between data efficiency, stability, and computational complexity. As shown in Figure 1.3, the policy is represented by an actor network. For every dimension of a continuous action space, the actor network outputs the parameters μ and σ^2 of a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$. The desired action is determined by sampling a value from each distribution. During the training process, the actor network learns to control the amount of exploration by outputting suitable variances σ^2. Nevertheless, the learning performance can sometimes be improved by encouraging a greater degree of exploration. This can be achieved by adding a bonus term based on the entropy $\mathcal{H}$ of the policy to the surrogate objective L_θ.

The performance of RL algorithms like PPO does not only depend on the selected task and the collected training data but also on additional *hyperparameters* that specify the details of the optimization procedure. While it is possible to automatically tune the hyperparameters [16, 187], the computing power required for this process is often prohibitively high. For that reason, the hyperparameters are usually selected by a human expert based on practical experience. Figure 2.5 shows the resulting learning curves for a reaching task when adjusting two exemplary hyperparameters. On the left side, the influence of the *batch size* on the learning performance is illustrated. The batch size is a hyperparameter controlling the amount of training data collected for each policy update. Selecting a larger batch size usually increases the final learning performance. However, more training data is required until the learning performance converges. The right side of the figure visualizes the impact of an *entropy bonus* added to the surrogate objective L_θ. In this example, the additional exploration encouraged by a small entropy bonus helps to improve the learning performance. A large bonus, on the other hand, has a negative impact on the performance as the optimizer focuses on increasing the entropy rather than the expected return of the policy.

Policy representations A policy π specifies the strategy of an RL agent by determining the action a that should be taken in state s. For problems with a discrete state space $\mathcal{S}$, the policy can be represented by a simple table [160]. In case of continuous state spaces, however, the number of feasible states is infinite and the policy is typically represented by a function approximator with trainable parameters θ. When learning robot movements, it is possible to derive a policy based on movement primitives trained via reinforcement learning [130]. In recent years, however, neural networks have proven to be particular powerful and versatile policy representations for motion generation. How the neural network is used to derive a policy depends on the selected RL algorithm. In case of DQN [114], a policy is derived based on an ϵ-greedy strategy using a neural network trained to approximate the optimal action-value function $Q_*(s, a)$. When using DDPG [97], an actor network is trained to map a state to a specific action in a deterministic way. The actor network used for PPO [148] outputs the parameters of a probability distribution from which the desired action is sampled. For the experiments conducted in this thesis, *fully connected feedforward networks* are used. As can be seen in Figure 2.4, the neurons of a fully connected network are arranged in several layers. The first layer, highlighted in blue, is referred to as *input layer*, while the last layer, highlighted in red, is known as *output layer*. Intermediate layers are called *hidden layers*. The term *fully connected network* indicates that each neuron is connected to every neuron of the following layer. The output signal of a neuron is determined by its input signals, the trainable network parameters θ and a predefined non-linear activation function. Further details on the mathematical foundations of neural networks can be found in a textbook by Goodfellow et al. [50]. Depending on the size and the structure of the state space $\mathcal{S}$, different network architectures are suitable for representing the policy. When using images as input data [114, 97, 135], it is common to utilize *convolutional neural networks* (CNNs) rather than fully connected networks. Another important aspect to consider is whether the *Markov property* is fulfilled. The Markov property implies that the behavior of the environment does not depend on previous states $s \in \mathcal{S}$ [160]. To describe systems that do not strictly satisfy the Markov property, the mathematical framework of *partially observable Markov decision processes* (POMDPs) [13, 116] can be utilized. Contrary to an ordinary MDP, the state of a POMDP is not directly accessible to the agent. While the agent receives an *observation* at each decision step, the optimal policy might depend on previous observations. In the context of *deep reinforcement learning*, different strategies are employed to cope with partial observability. One approach is to utilize a *recurrent neural network* (RNN) as policy representation [143, 8, 152]. In contrast to feedforward networks, RNNs operate with an internal memory, for example based on *gated recurrent units* (GRUs) [29] or a *long short-term memory* (LSTM) [63]. The memory can be used to integrate knowledge about previous observations into the output signal. As an alternative, it is also possible to employ a memoryless feedforward network. In this case, partial observability can be taken into account by using an input signal that contains a fixed number of previous observations [53, 72, 95].

Action representations for motion generation When learning robot trajectories, it is important to specify how an action is mapped to a movement. If a continuous action space is used, it is common to work with normalized actions. More precisely, each dimension of an action usually has a value range from -1 to 1. To ensure that the output signals of an actor network do not exceed this range, a *squashing function* like *tanh* can be used. Alternatively, the gradients applied during a policy update can be adjusted



(a) Action specifies constant setpoints φ_d

(b) Action specifies constant setpoints $\dot{\varphi}_d$

Figure 2.6.: Two ways to map actions a to trajectory setpoints shown for a single joint.

if an output signal would otherwise fall outside the valid range [56]. The normalized actions are typically used to calculate setpoints for a lower-level motion controller such as those introduced in section 2.1.2.1. One approach is to directly specify torque setpoints without relying on an additional trajectory controller [97, 127]. During two decision steps, the requested torque is typically kept constant. In theory, directly controlling motor torques offers a high flexibility with respect to the desired learning task. As an additional advantage, the motion control does not depend on the implementation details of a trajectory controller. However, torque setpoints are susceptible to environmental disturbances that may be caused by external forces or a sim-to-real transfer. By computing setpoints for a trajectory controller, the robustness against disturbances can be increased. For example, it is possible to specify desired Cartesian positions and orientations of the end-effector [69, 90, 72, 119]. However, *singular configurations*, in which a desired Cartesian movement cannot be executed, may occur [151]. Problems resulting from singularities can be circumvented by learning motions in joint space [51, 67]. One way is to compute joint position setpoints φ_d for a PID controller that are kept constant during two decision steps [127]. However, as can be seen in Figure 2.6, the resulting velocity setpoints $\dot{\varphi}_d$ are not bounded. Consequently, the actual values cannot closely follow the setpoints and the executed movement strongly depends on the parameters of the PID controller. To obtain smoother movements, the position setpoints can be passed through a low-pass filter [129, 95]. However, when using filters, the responsiveness of the robot is reduced and the resulting setpoints may depend on previous actions. As shown in Figure 2.6, the selected action can also be used to define constant velocity setpoints $\dot{\varphi}_d$. The corresponding position setpoints φ_d are calculated by integration. In this case, the position setpoints φ_d are continuous, but the acceleration setpoints $\ddot{\varphi}_d$ are still unbounded. By selecting a constant jerk setpoint $\dddot{\varphi}_d$ between decision steps, it is possible to additionally enforce continuous velocities $\dot{\varphi}_d$ and continuous accelerations $\ddot{\varphi}_d$. Based on this idea, chapter 3 introduces an action representation that ensures compliance with the kinematic joint limits (1.3) - (1.6).

Data acquisition Training data for reinforcement learning can either be collected in a simulation environment or obtained using real robots. Data collection with real robots offers the advantage that the training environment typically corresponds to the actual operating environment. Contrary to this, generating data in simulation usually requires a *sim-to-real transfer* after training. However, collecting data in the real world is accompanied by a number of challenges. A common issue is the time required to gather sufficient training data. As an example, four month were needed to collect around 800 hours of training data for a vision-based grasping task [69]. Using multiple robots to speed up the training is possible, but involves high investment costs. Another issue is the initialization of the training environment. If a human is required to reset the environment after each trial, the robots cannot be left unattended during the training phase. For some tasks, it is possible to design the environment in such a way that little human intervention is needed [51, 93, 69]. Another approach is to learn how to automatically reset the environment during the training phase [35]. However, for tasks such as welding or grinding, where objects are permanently modified, efficient real-world data acquisition is still an open research problem. Lastly, exploring the environment with real robots raises safety issues, especially at the beginning of the training phase where an RL agent typically behaves randomly. The problems mentioned above can be mitigated by collecting data in a simulation environment. Popular physics simulators for reinforcement learning in robotics include PyBullet [30], MuJoCo [171], and Isaac Gym [103]. While generating sufficient data is usually a computational intensive task, the time required for a training process can be significantly reduced using *parallel processing* and *cloud computing*. In addition, no human intervention is required during the data collection and safety violations do not lead to physical damage. However, when transferring networks trained in simulation to the real world, the limited accuracy of the physics simulation must be taken into account.

Sim-to-real transfer As collecting training data in the real world is challenging, neural networks are often trained based on data generated by a physics simulator. However, it is usually difficult to accurately reproduce the operating environment of a robot. Modeling errors can be caused by incomplete knowledge of the environment, but also by mathematical approximations used to reduce the computational effort of the physics simulation. As a consequence, controlling a real robot with a network that was trained in simulation can have a negative impact on the task performance. In addition, a domain change might increase the probability of safety violations. To mitigate the negative effects of a *sim-to-real transfer*, different strategies have been proposed in the literature. One line of research aims to improve the accuracy of the simulation, for instance by performing a system identification [72, 183] or by additionally modeling the latency of motor controllers [164]. As an alternative, measures to increase the robustness of the policy can be implemented. For example, disturbing forces can be applied to the robot during the training phase [134, 164]. The basic idea of *dynamics randomization* is to select random values for parameters that are not exactly known during the training process [128, 27]. While dynamics randomization is effective in increasing robustness, it can also lead to overly conservative policies that do not achieve a good task performance [164, 184]. When using dynamics randomization, it can be advantageous to represent the policy by a *recurrent neural network* (RNN) as the internal memory can help to adjust the policy to the current operating environment [8, 152]. To increase the robustness of neural networks that process images as input data, the visual appearance of the environment can be randomized [169, 67, 8].

Finally, it is also possible to improve the task performance after a sim-to-real transfer by fine-tuning the policy using a small amount of real-world data [142, 68].

Limitations Although model-free reinforcement learning has proven to be a versatile and powerful technique, there are still certain caveats that impact its practical applicability. Usually, the data efficiency of model-free RL algorithms is relatively low. As a result, collecting a sufficient amount of training data can be both time-consuming and costly. In addition, the training results are sensitive to a broad range of hyperparameters. A systematic tuning of the hyperparameters is particularly difficult when using on-policy algorithms since new training data must be recorded for each training run. Another issue is the definition of a suitable reward function for a specific application. In practice, it can be difficult to define rewards that encourage the desired behavior without introducing loopholes that can be exploited during the optimization process. Moreover, a task may involve several competing objectives at the same time. For instance, a robot that is trained to follow a reference path should be rewarded for moving quickly and for remaining close to the reference path. However, if the robot moves faster it becomes more difficult to stay close to the reference path. Consequently, both objectives must be weighed against each other, which usually requires a manual adjustment of weighting factors in the reward function. While the aforementioned limitations apply to model-free RL in general, additional restrictions arise in the context of learning robot trajectories. Most importantly, existing approaches typically do not utilize the full kinematic potential of industrial robots. While learning slow robot motions is an effective strategy to avoid severe safety incidents, robots in an industrial environment are expected to operate in a highly time-efficient way. Common action representations, however, are not well suited for producing fast movements as they do not explicitly consider the kinematic capabilities of the robot joints. As a consequence, the resulting controller setpoints may not be tracked accurately and the robot joints may be exposed to excessive stress. Apart from joint limit violations, it is also important to avoid collisions during and after the training phase. The latter is particularly challenging when moving obstacles are involved, for instance in the context of *human–robot collaboration* (HRC). An overview of existing methods for *safe reinforcement learning* and their corresponding limitations can be found in section 2.2.

2.1.5. Summary

There are several approaches for generating robot trajectories in real time, each with its own strengths and weaknesses. Time-optimal trajectories to a kinematic target state can be computed using existing motion libraries. However, apart from simple cases like point-to-point movements, finding suitable target states for a desired task is typically non-trivial. In addition, the generated trajectories are not necessarily collision-free. If a differentiable model of the system dynamics and a corresponding cost function are known, model predictive control can be used to calculate optimized trajectories. While model-based approaches offer a high degree of transparency and interpretability, they lack the flexibility of model-free learning techniques. In addition, the computing power required to achieve real-time capability can be relatively high, especially if a large prediction horizon is selected. Based on human demonstrations, it is possible to generate trajectories without requiring a system model or a task-specific objective function. However, recording suitable demonstrations can be challenging. In addition, compounding errors can prevent a policy

trained on expert demonstrations from recovering after an incorrect decision. Model-free reinforcement learning is a flexible machine learning technique that seeks to find an optimized policy through trial and error. The environment is explored during a training phase, in which the performance of each action is assessed by a scalar reward. While model-free RL has been successfully applied to a wide range of tasks, ensuring a safe exploration of the environment is still a challenging problem. When learning fast robot trajectories, the exploration process can easily lead to joint limit violations or collisions. As a result, special protective measures are required to utilize the flexibility of model-free RL without risking damage to the robot and its surroundings.

2.2. Safe reinforcement learning

When using model-free RL, knowledge is acquired based on the principle of trial and error. For practical applications, the exploration of the environment is often subject to safety constraints. The specific research area of *safe reinforcement learning* investigates methods to reduce the likelihood of safety violations. A survey paper on this topic is provided by Brunke et al. [25]. Techniques for enhancing safety can be roughly divided into two categories: Practitioners tend to address safety issues in a task-specific way, typically based on a careful design of the environment, the action space, and the reward function. Contrary to this, theoretical approaches examine the problem from a more mathematical perspective, often based on the framework of *constrained Markov decision processes* (CMDPs) [4]. Both approaches are explained in more detail below.

2.2.1. Practical approaches

Since safety issues are a common problem in reinforcement learning, practitioners have developed a number of measures to address this area of concern. The level of safety can be enhanced by lowering the risk of safety violations or by minimizing their effects. In practice, safety considerations are often integrated into the design of the learning task and the environment. An overview of different strategies is given below.

Design of the environment A crucial aspect in terms of safety is the environment of the robot. The better the environment is known in advance, the easier it is to avoid safety violations. In practice, the workspace of the robot is often restricted to a well-defined area without obstacles. When learning movements with bipedal robots, it is common to attach the robot to a support frame that prevents the robot from falling over if it loses its balance [95, 32]. Another widely used approach is to carry out the training process in a simulation environment, where safety violations do not cause damage. In this case, however, special care must be taken to avoid safety problems after a sim-to-real transfer. Depending on the design of the environment, it may also be possible to pre-train a policy based on human demonstrations [145], thereby avoiding the random exploration at the beginning of the training phase.

Design of the action space A careful design of the action space can also contribute to increasing safety in reinforcement learning. Specifying controller setpoints in Cartesian space, for example, makes it relatively easy to assess and restrict the operating range

of the robot. In some cases, it is possible to learn setpoints for a lower-level controller that already offers safety features [72]. The mapping from actions to movements can also be used to influence the smoothness and the maximum velocity of the resulting robot motions. Smooth robot motions are important to adhere to kinematic joint constraints. Slow movements, on the other hand, simplify safety checks and reduce the potential impact of collisions.

Design of the reward function During the training phase, the RL algorithm attempts to maximize the expected sum of future rewards. As a consequence, safety violations can be discouraged by assigning a low reward to actions that lead to undesired behaviors. Similarly, it is possible to terminate an episode if a safety violation occurs so that no further rewards can be received. While both of these strategies are effective in reducing the likelihood of safety violations, they do not provide strict safety guarantees and have no effect at the beginning of the training phase.

Task-specific safety heuristics It is also possible to implement task-specific heuristics in order to increase the safety of the learning process. For example, Gu et al. [51] define a bounding sphere that should not be left by the end-effector of the robot. Under normal conditions, actions are mapped to joint velocities. However, if the end-effector would leave the bounding sphere, the commanded action is adjusted so that the end-effector stays within the sphere. While task-specific heuristics can be effective, they typically require additional implementation effort.

2.2.2. Theoretical approaches

The following section introduces methods that address the problem of safe reinforcement learning from a theoretical perspective. This is often done based on the mathematical framework of CMDPs [4]. Compared to the definition of normal MDPs provided in section 1.1.1, CMDPs additionally introduce a set of cost functions $\mathcal{C}$. Similarly to the reward function, a cost function $C_i \in \mathcal{C}$ with C_i: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ assigns a cost to each state transition. A survey paper on policy optimization using CMDPs is provided by Liu et al. [99]. Following the terminology of this survey, cost functions can be used to specify *cumulative constraints* or *instantaneous constraints*. A common class of cumulative constraints enforces an upper bound c_i on the expected sum of costs received over time:

$$\mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \cdot C_i(s_{t+k}, a_{t+k}, s_{t+k+1}) \,\middle|\, s_t = s, a_t = a \right] \leq c_i \tag{2.18}$$

An example for such a constraint is the energy consumption of a mobile robot, which must not exceed the capacity of its battery [99]. Instantaneous constraints, on the other hand, are also called *state-wise constraints*, as they must be fulfilled at each state transition:

$$C_i(s_t, a_t, s_{t+1}) \leq c_i \tag{2.19}$$

A survey on the particular topic of state-wise constraints is provided by Zhao et al. [190]. While the safety constraints considered in this thesis can be regarded as instantaneous, it is non-trivial to assign an appropriate cost to each state transition. Whether an action

leads to a safety violation does not only depend on the current state, but also on the policy followed afterwards. Moreover, the occurrence of safety violations can be probabilistic if the environment behaves stochastically. As a result, the cost of a state transition can be interpreted as the risk of causing a safety violation at a later point in time.

Figure 2.7 provides an overview of different theoretical approaches to address the problem of safe reinforcement learning. The classification is loosely based on a taxonomy introduced by García and Fernández [44]. Existing methods typically adjust either the *optimization objective* or the *selected actions* to reduce the probability of constraint violations. Further details on both directions are given below.

2.2.2.1. Adjusting the optimization objective

The standard optimization objective of model-free RL is to maximize the expected sum of discounted rewards received over time. Consequently, practitioners assign low rewards to undesired actions or terminate an episode if a safety violation occurs. In the context of *on-policy algorithms*, the optimization objective can be adjusted to avoid large policy updates or to encourage a greater amount of exploration. Similarly, it is possible to discourage unsafe behaviors by adjusting the optimization objective of the RL algorithm. In the following, different ways to modify the optimization objective are outlined. A common disadvantage of these methods is that they are not effective in preventing safety violations at the beginning of the training phase.

Learning a pessimistic policy Since common RL algorithms aim to maximize the *expected* return, they implicitly search for a policy that is optimized with respect to the average behavior of a stochastic environment. However, in order to reduce the likelihood of safety violations, it can be beneficial to look at future developments from a *pessimistic* perspective. As an example, consider a robot that should move to a certain position. It is assumed that there is a fast way that occasionally leads to collisions and a slow way that is always collision-free. On average, selecting the fast way leads to a higher return. However, a *pessimistic policy* might choose the slow way in order to avoid a low return in the rare event of a collision. To account for such rare but fatal events, Heger [57] proposed *$\hat{Q}$-learning*, a modified version of Q-learning [179] that optimizes a policy with respect to the *worst-case* return. Similarly, Gaskett [45] introduced an algorithm called *β-pessimistic Q-learning*, which uses an additional parameter β to control the level of pessimism. While pessimism can help reduce safety violations, it can also lead to a poor task performance [45, 44].

Incorporating a risk metric To encourage the learning of a safe policy, it is also possible to integrate a risk metric into the objective function of an RL algorithm. However, finding a suitable risk metric can be challenging. What is regarded as risk depends on the desired learning task and the respective safety constraints. In the financial sector, it is common to use the variance of the return as a metric to assess the risk of an investment strategy [106, 150]. For Markov decision processes, the resulting problem of finding a trade-off between the mean and the variance of the return is well studied [156, 104]. In the specific context of reinforcement learning, Tamar et al. [162] proposed a policy gradient algorithm controlling the variance of the return by adding a penalty term to the

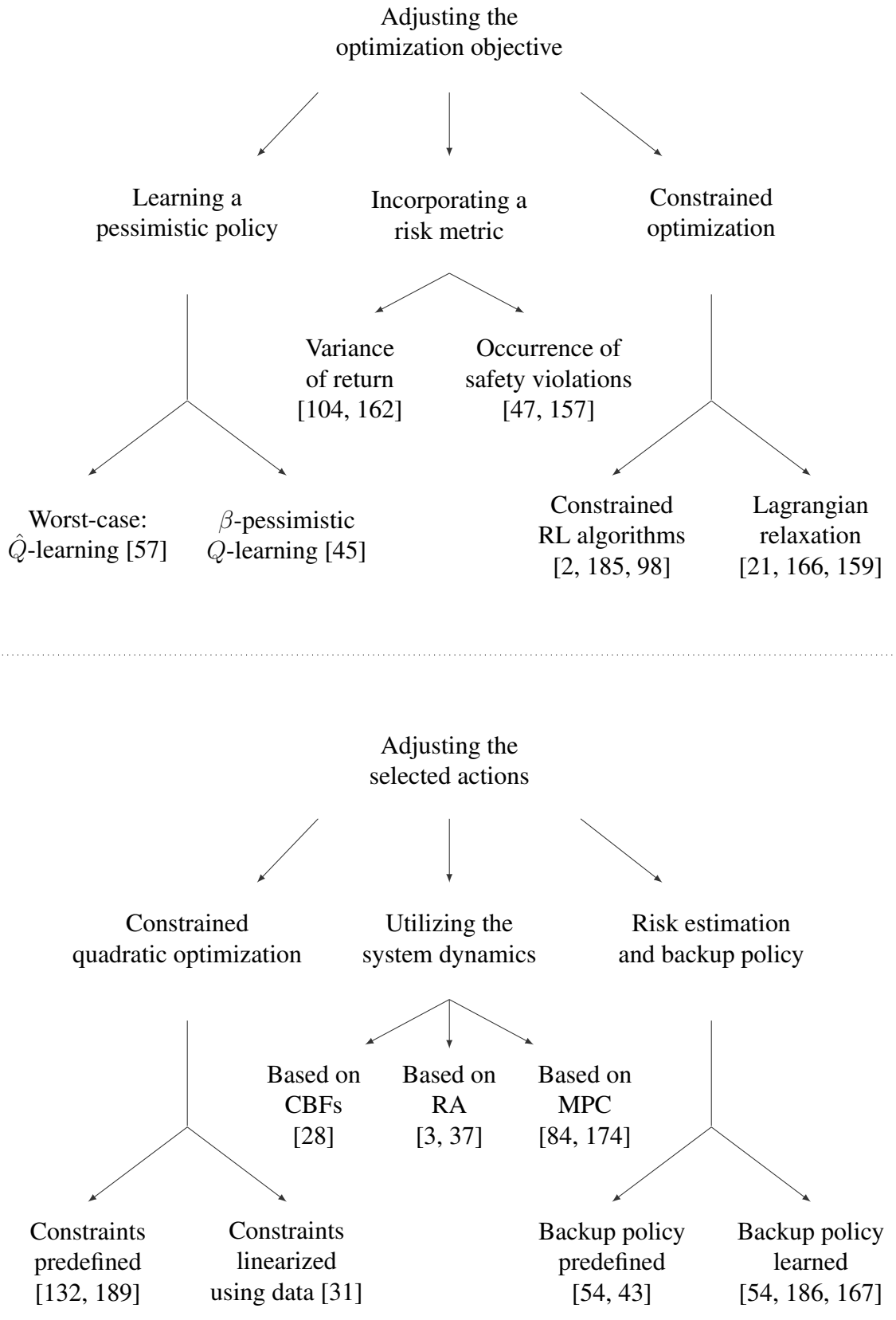


Figure 2.7.: An overview of existing methods to address the problem of safe reinforcement learning. The classification is inspired by García and Fernández [44].

objective function. For the safety constraints considered in this thesis, however, it is more reasonable to use a risk metric based on the likelihood of safety violations. Geibel and Wysotzki [47] define the risk as a function $\bar{Q}_\pi(s, a)$, indicating the probability of ending in an error state when selecting action a in state s and following the current policy π afterwards. They also propose a *discounted risk*, where less weight is given to error states occurring in the distant future. In order to learn a risk-averse policy, the optimization objective of Q-learning is adjusted to consider both the action-value function $Q_\pi(s, a)$ and the risk function $\bar{Q}_\pi(s, a)$. A similar risk metric is employed by Srinivasan et al. [157]. As a first step, a policy π and its corresponding risk function $\bar{Q}_\pi(s, a)$ are learned during a pre-training phase. Subsequently, the risk function is incorporated into the optimization objective of the RL algorithm SAC [52]. In this way, the probability of safety violations can be reduced when learning more complex tasks during a fine-tuning phase.

Constrained optimization While normal RL algorithms use unconstrained optimization objectives, it is also possible to utilize methods from the field of *constrained optimization* to incorporate safety constraints. One example is the RL algorithm *constrained policy optimization* (CPO) [2], which addresses cumulative constraints as described by equation (2.18). Based on a trust region approach and several approximations, CPO provides an upper bound on the expected constraint cost and the worst-case performance of a policy update. A drawback of CPO is the use of second-order derivatives, which can be difficult to compute [98]. The algorithm *projection-based constrained policy optimization* (PCPO) [185] is closely related to CPO. However, an additional projection step is introduced, which improves the recovery from constraint-violating policy updates and contributes to a faster convergence of the training process. *Interior-point policy optimization* (IPO) [98] encourages constraint satisfaction by adding penalties based on logarithmic barrier functions to the unconstrained optimization objective of PPO [148]. Similarly, it is possible to transform a constrained optimization problem into an unconstrained problem using *Lagrangian relaxation* [21, 166, 159]. A performance comparison for several constrained RL algorithms is provided by Ray et al. [137].

2.2.2.2. Adjusting the selected actions

Methods that adjust risky actions offer the advantage that safety violations can be prevented right from the start of a training process. Under certain assumptions, it is even possible to provide specific safety guarantees. Adapting risky actions typically involves two sub-problems. The first step is to identify an action as potentially risky. In a second step, a less risky alternative must be found. In order to assess the risk of an action, a certain amount of knowledge about the environment is required. This knowledge can be incorporated by defining a task-specific safety heuristic, by providing a dynamics model, or by performing background simulations in a physics simulator. As an alternative, it is also possible to identify risky actions based on data collected in the environment. When using such a learning-based approach, however, special care must be taken to ensure that the robot is not damaged while collecting the required data. A less risky alternative action can be found by solving an optimization problem, by utilizing methods from model-based control, or by providing a backup policy. An important aspect when determining and adjusting risky actions is the time horizon taken into consideration. As stated by Fraichard [40], a robotic system should reason over an infinite time horizon or up to the

time needed to reach a safe goal state in order to ensure motion safety. In practice, however, this condition can be difficult to fulfill, especially if the environment does not behave deterministically. As a consequence, safety violations can often not be entirely prevented. In the following, three common approaches to adjust risky actions are described in more detail. An overview is also given in Table 2.1.

Constrained quadratic optimization The basic idea of this approach is to find the closest action a_t to the original action a_T that satisfies certain state-wise constraints [31]:

$$\arg\min_{a_t} \frac{1}{2}\|a_t - a_T\|^2$$

$$\text{s.t.} \quad C_i(s_t, a_t, s_{t+1}) \leq c_i \quad \forall\, C_i \in \mathcal{C} \tag{2.20}$$

As shown in equation (2.20), this is typically achieved using a constrained optimization problem with a quadratic objective function. If the constraints C_i are linear and solely depend on s_t and a_t, the optimization problem can be efficiently addressed by solving a *quadratic program* (QP). In order to satisfy these requirements, Dalal et al. [31] use a linear approximation of the constraints C_i. The approximation is based on a neural network trained using data collected prior to the actual training phase. As the resulting quadratic optimization problem can be treated as a special layer of a neural network [7], this safety mechanism is also called *safety layer*. Pham et al. [132] introduced a safety layer to avoid collisions with a robot arm. The definition of the constraints C_i is based on a collision avoidance method proposed by Faverjon and Tournassoud [36]. More precisely, the method enforces a minimum distance d_s between two potentially moving points by constraining the Cartesian velocity $\dot{d}$ between the points:

$$\dot{d} \geq -\xi \frac{d - d_s}{d_i - d_s} = \dot{d}_{\min} \tag{2.21}$$

In equation (2.21), d is the current distance between the points and ξ is a positive coefficient that controls the convergence speed. The inequality constraint is only active if the distance d is smaller than an *influence distance* d_i. Based on the Jacobi matrix, a linear connection between the Cartesian velocity of a robot point and the joint velocities can be established. If the relation between the selected action and the resulting joint velocities is linear, the optimization problem (2.20) can be efficiently solved using a QP. In theory, the distance d in equation (2.21) should be defined with respect to the closest points of two objects. In practice, however, complex geometric shapes are often approximated by bounding spheres, as the closest points of non-strictly convex objects can move discontinuously [71]. Figure 2.8 shows how the method is used to prevent a collision between a blue obstacle and a robot link approximated by a green sphere. One important requirement is that the minimum Cartesian velocity $\dot{d}_{\min}$ enforced by (2.21) must be reachable without exceeding the kinematic abilities of the robot joints. If multiple obstacles are observed, all inequality constraints must be satisfied at the same time. As long as a solution to the optimization problem exists, common safety layer approaches will find a valid action. However, safety layers typically do not ensure that at least one valid action exists at each subsequent decision step. As a consequence, the resulting QPs might have no solution and safety violations can continue to appear. To address this problem, Zhao et al. [189] proposed an *implicit safe set algorithm* (ISSA) that avoids collisions between mobile robots in a two-dimensional plane. The basic idea is to introduce a constraint which ensures

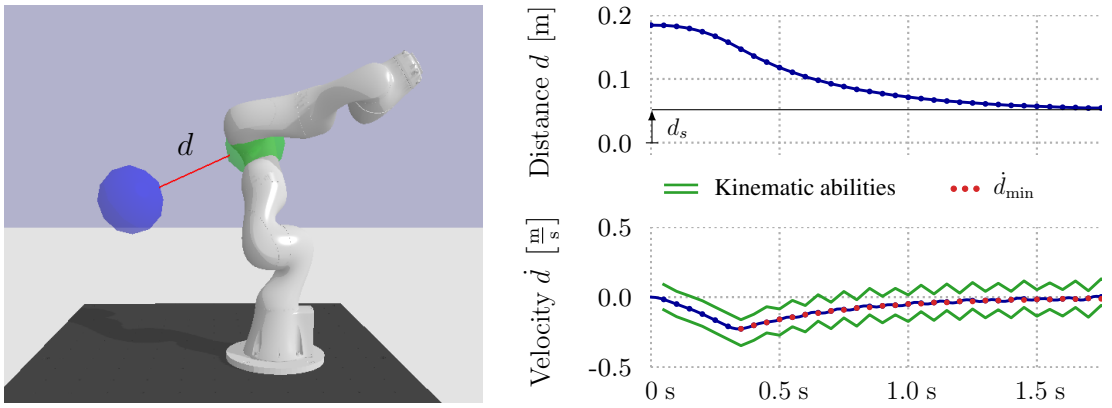


Figure 2.8.: Based on a collision avoidance method proposed by Faverjon and Tournassoud [36], it is possible to define inequality constraints for a safety layer that enforce a minimum distance d_s between two points [132]. In this example, the method is used to avoid a collision between the blue and the green sphere.

that the set of feasible actions in state s_{t+1} is non-empty. More precisely, an energy function $\phi(s)$ is used to determine whether a feasible action exists in state s_{t+1}. While the approach offers strict safety guarantees, there are two caveats: First, the definition of the energy function $\phi(s)$ is task-specific and based on several assumptions. Second, the resulting optimization problem cannot be handled by a simple QP solver as it depends on $\phi(s_{t+1})$. Consequently, the problem is solved using a custom line search algorithm, which requires access to a dynamics model or a simulator to determine s_{t+1} based on s_t and a_t.

Utilizing the system dynamics When adjusting an action a_t, it is usually not sufficient to focus on potential safety violations during the state transition from s_t to s_{t+1}. Instead, the subsequent state transitions must also be taken into account. In the field of model-based control, there are several methods to make sure that a dynamic system always stays within a safe subset of the state space. These methods can also be applied in the context of model-free RL, provided that a suitable model $f(s, a)$ is known:

$$s_{t+1} = f(s_t, a_t) \tag{2.22}$$

While this requirement limits the inherent flexibility of model-free RL, there can still be advantages compared to using a purely model-based approach. For example, the reward function of an RL agent can be designed in a flexible way, as it does not need to be differentiable with respect to the selected action. One approach to enforce safety in model-based control are *control barrier functions* (CBFs) [180, 5, 6]. In order to avoid safety violations during the learning process of a model-free RL agent, Cheng et al. [28] utilize a controller based on CBFs. More specifically, the method is used to keep an inverted pendulum upright without leaving a certain angular range and to optimize the fuel efficiency of a car while avoiding collisions. The system dynamics are partially known and assumed to be linear with respect to the selected action. System uncertainties are learned from data using a *Gaussian process* (GP). In theory, the approach can provide strict safety guarantees. In practice, however, safety violations may occur with a probability of δ due to uncertainties in the system dynamics. As safety is guaranteed with a probability of $(1-\delta)$, the constraint satisfaction is considered as *probabilistic*. A similar approach to ensure safety is proposed by Berkenkamp et al. [17]. The method assumes knowledge about

an initial policy that is safe within a certain area of the state space. By utilizing a suitable Lyapunov function, additional data is collected to reduce system uncertainties modeled by a GP. The improved model can then be used to expand the safe area of the state space. Using this approach, an RL policy is trained to balance an inverted pendulum without letting the pendulum fall down. Another line of research uses *reachability analysis* (RA) to enable safe reinforcement learning [49, 3, 37]. The basic idea is to determine a safe set of the state space and a corresponding safety controller based on a two-player differential game. In the interior of the safe set, an RL agent can execute arbitrary actions without causing safety violations. At the boundary, the safety controller prevents the system from leaving the safe area. In the differential game setup, the safety controller tries to keep the system in a safe state, while its opponent tries to cause constraint violations. If a suitable dynamics model is known, the differential game can be addressed using Hamilton–Jacobi methods [14]. Fisac et al. [37] utilize this approach to avoid collisions with a quadrotor. System uncertainties are modeled by a GP. Under certain assumptions, the method can guarantee probabilistic constraint satisfaction. However, to reduce the computational effort during real-time execution, a local approximation of the original problem is proposed. Using this approximation, constraint satisfaction is still encouraged, but no longer guaranteed. Safety in reinforcement learning can also be enforced by utilizing techniques from *model predictive control* (MPC). Koller et al. [84] use a safety controller based on MPC to keep an inverted pendulum upright and to balance a pole attached to a cart. Similarly, Wabersich and Zeilinger [174] utilize an MPC-based safety filter to swing up an inverted pendulum and to avoid ground contact with a quadrotor. In summary, methods that combine model-free RL with a dynamics model are effective in avoiding safety violations and can guarantee probabilistic constraint satisfaction under certain assumptions. However, the requirement to specify a suitable model limits the potential range of applications. In practice, model-based approaches are often applied to inverted pendulums or quadrotors.

Risk estimation and backup policy The basic idea of this research direction is to replace risky actions using a backup policy. More precisely, an RL agent following a task policy π_T might suggest to take action a_t^T in state s_t. However, action a_t^T is only executed, if the estimated risk $\bar{Q}(s_t, a_t^T)$ does not exceed a predefined threshold $\bar{q}_{Th}$. Otherwise, a backup policy π_B is used to generate an alternative action a_t^B:

$$a_t = \begin{cases} a_t^T & \text{if } \bar{Q}(s_t, a_t^T) \leq \bar{q}_{Th} \\ a_t^B & \text{otherwise} \end{cases} \tag{2.23}$$

In the context of this work, the risk of an action refers to the occurrence of safety violations. However, whether a safety violation occurs, typically does not only depend on the current state s_t and the desired action a_t^T but also on the policy followed afterwards. One approach is to estimate the risk $\bar{Q}_{\pi_B}(s_t, a_t^T)$, where the subscript indicates that the backup policy π_B is followed after entering s_{t+1}. As an alternative, it is also possible to assume that the current policy π, which results from the interaction between π_T and π_B, is used after executing a_t^T. In this case, the backup policy can be derived based on the resulting risk function $\bar{Q}_\pi(s, a)$. Hans et al. [54] estimate the risk of a state transition by estimating the minimum reward occurring when executing action a_t^T and subsequently following the backup policy π_B. For this estimation, a least squares approximation based on previously collected data is used. Two potential options for specifying a backup policy are examined. The first option is to utilize an existing controller that is safe but not optimal in terms of

the resulting task performance. Alternatively, the backup policy is learned using data provided by a teacher. The approach is evaluated by optimizing the power output of a gas turbine without causing undesired effects. Garcia and Fernández [43] use backup policies based on predefined controllers to learn tasks like car parking, pole balancing, and helicopter hovering. However, a different definition of risk is used. More precisely, the algorithm constructs a set of states that are known to be safe. The backup policy is used if the minimum Euclidean distance between the current state and a state in the safe set is larger than a predefined threshold. Yang et al. [186] use model-free RL to learn a backup policy π_B that keeps a quadruped robot balanced. Whether an action from the backup policy is executed depends on task-specific criteria that are designed using knowledge about the system dynamics. Eysenbach et al. [35] simultaneously learn a forward policy and a reset policy using model-free RL. While the forward policy is trained to maximize the task performance, the reset policy attempts to return the environment to its initial state. When collecting real-world data, the reset policy can be used to avoid manual resets of the environment. As entering a non-reversible state can be interpreted as a safety violation, the reset policy can be seen as a special kind of backup policy. Bharadhwaj et al. [19] use a conservative risk function $\bar{Q}_C(s, a)$, which tends to overestimate the actual risk of an action. If a desired action a_t^T does not satisfy the condition $\bar{Q}_C(s_t, a_t^T) < \bar{q}_{Th}$, further actions are sampled from π_T until the condition is met. Thus, the backup policy can be considered as derived from the risk function. However, the proposed strategy does not guarantee that a suitable action will be found. Moreover, the probability of an incorrect risk classification increases with the number of sampling attempts. In practice, the sampling is repeated a certain number of times and the action with the lowest risk is selected, even if the threshold value $\bar{q}_{Th}$ is exceeded. Thananjeyan et al. [167] estimate the risk function $\bar{Q}_\pi(s, a)$ when following the current policy π. Using this risk function, a backup policy is derived. The training procedure of the backup policy is similar to the off-policy algorithm DDPG, which derives an actor policy based on the action-value function $Q_\pi(s, a)$. As the current policy π depends on π_T, the risk function $\bar{Q}_\pi(s, a)$ and the backup policy need to be updated during the training of π_T. In contrast, methods utilizing a fixed backup policy π_B can be used to learn different tasks without having to update the risk function $\bar{Q}_{\pi_B}(s, a)$.

2.2.3. Summary

In order to find optimized actions, RL agents need to explore their environment autonomously. During this process, the occurrence of safety violations is a widespread problem. In the context of *safe reinforcement learning*, various methods have been developed to ensure safety during and after the training phase. Existing approaches differ in terms of their implementation effort, range of application, and level of constraint satisfaction. Practitioners tend to address safety issues in a task-specific manner, often without a precise mathematical analysis of the problem. If possible, the environment is designed so that safety violations rarely occur or do not cause major damage. Alternatively, penalty terms are added to the reward function so that the RL agent learns to avoid unsafe actions during the course of training. Theoretical approaches, on the other hand, address safety problems from a more mathematical perspective, often based on the framework of CMDPs. Typically, safety violations are avoided by modifying either the optimization objective or the selected actions. The optimization objective of an RL algorithm can be adjusted by incorporating a risk metric or by utilizing techniques from constrained optimization. However, these techniques usually show limited effectiveness in the early stages of the

training phase. Alternatively, it is possible to modify risky actions, for instance based on constrained quadratic optimization. While constrained optimization problems can be efficiently solved using quadratic programs, avoiding conflicting constraints is challenging. To ensure the existence of at least one feasible action, one line of research attempts to keep the system within a safe subset of the state space. If a model of the system dynamics is known, this can be achieved using techniques like Hamilton–Jacobi reachability analysis or model predictive control. However, the need to provide a suitable model limits the potential scope of model-free RL. Based on a risk metric and a backup policy, a similar intention can be pursued without having to specify a model of the system dynamics. The risk metric and the backup policy can be learned from data or specified based on model knowledge. This approach can also be used to avoid collisions when learning robot trajectories in joint space. However, existing methods focus on different applications and do not explicitly consider the kinematic capabilities of robotic manipulators.

Table 2.1.: Different methods to adjust risky actions and their specific characteristics.

Reference	Constraint satisfaction	Predefined component	Learned component	Demonstrated applications
Constrained quadratic optimization:				
• Dalal et al. [31]	encouraged	Constraints depending on s_{t+1}	Linear mapping from s_t and a_t to s_{t+1}	Avoiding certain areas with a ball and a spaceship
• Pham et al. [132]	encouraged	Constraints to avoid collisions	-	Avoiding collisions with an industrial robot
• Zhao et al. [189]	strict	Energy function $\phi(s)$	-	Avoiding collisions with mobile robots
Utilizing the system dynamics:				
• Cheng et al. [28]	probabilistic	CBF-based safety controller	Model uncertainties using a GP	Inverted pendulum and avoiding collisions with a car
• Fisac et al. [37]	probabilistic	RA-based safety controller	Model uncertainties using a GP	Avoiding ground collisions with a quadrotor
• Koller et al. [84]	probabilistic	MPC-based safety controller	Model uncertainties using a GP	Inverted pendulum and cart-pole balancing
Risk estimation and backup policy:				
• Hans et al. [54]	encouraged	Backup policy using an existing controller	Risk function $\bar{Q}_{\pi_B}(s,a)$	Power optimization of a gas turbine
• Yang et al. [186]	encouraged	Criteria for using the backup policy	Backup policy using model-free RL	Locomotion with a quadruped robot
• Bharadhwaj et al. [19]	encouraged	Strategy to sample actions based on $\bar{Q}_C(s,a)$	Conservative risk function $\bar{Q}_C(s,a)$	Navigation, manipulation, and locomotion tasks
• Thananjeyan et al. [167]	encouraged	-	Risk function $\bar{Q}_\pi(s,a)$ and backup policy	Navigation and manipulation tasks

3. Learning trajectories subject to kinematic joint constraints

When learning robot movements, the generated trajectory setpoints need to satisfy kinematic joint constraints specified by the robot manufacturer. Not adhering to the constraints can lead to inaccurate tracking of the desired trajectory setpoints or to overloading of the robot joints. In this chapter, an action mapping for model-free reinforcement learning is presented, which ensures that the position, velocity, acceleration, and jerk limits of each robot joint are always respected. First, the problem of considering kinematic joint constraints is formalized and its relation to existing work is discussed. Next, the proposed action mapping is presented and its underlying mathematical principles are explained. Finally, the method is evaluated by learning three different robot tasks using model-free RL. The learning objective of the first task is to generate a movement that maximizes the average absolute velocity of each robot joint without exceeding kinematic limits. In this context, the proposed action mapping is compared to adding penalty terms to the reward function, a method commonly applied by practitioners to avoid undesired learning outcomes. The second task is to quickly follow a reference path without violating joint limits. Apart from staying close to the reference path, additional learning objectives can be considered. For instance, an industrial robot can learn to balance a ball on a plate while following the reference path. The third evaluation task deals with a similar ball-on-plate scenario. However, a reference trajectory is used, providing information on the path and the temporal behavior of the desired movement. For both reference paths and reference trajectories, the evaluation includes balancing experiments with a real industrial robot. The content of this chapter is based on three previously published publications. More precisely, the action mapping to consider kinematic joint constraints was introduced in [76]. The usage of the action mapping to track reference paths was proposed in [78]. Finally, the approach to adjust reference trajectories was initially described in [82].

3.1. Problem description

In order to learn robot trajectories using model-free RL, a *Markov decision process* $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ is defined to model the decision-making process. As explained in section 1.1.1, an RL agent selects an action $a_t \in \mathcal{A}$ at each decision step t. For the specific problem of *online trajectory generation*, an action a_t specifies a desired robot motion from the current decision step t to the following decision step $t+1$. The time between decision steps is typically constant. In section 2.1.2.1, different control strategies for lower-level motion control are introduced. This work assumes that a *joint trajectory controller* is used and that the corresponding trajectory setpoints φ_d, $\dot{\varphi}_d$, and $\ddot{\varphi}_d$ are generated based on the actions of an RL agent. As explained in section 1.1.2, common robot joints are subject to kinematic constraints as specified by (1.3) - (1.6). In this chapter, the generated trajectory

setpoints are assumed to be accurately tracked by the *joint trajectory controller*. Consequently, the kinematic constraints (1.3) - (1.6) can be regarded as satisfied if the setpoints comply with the following constraints at all times:

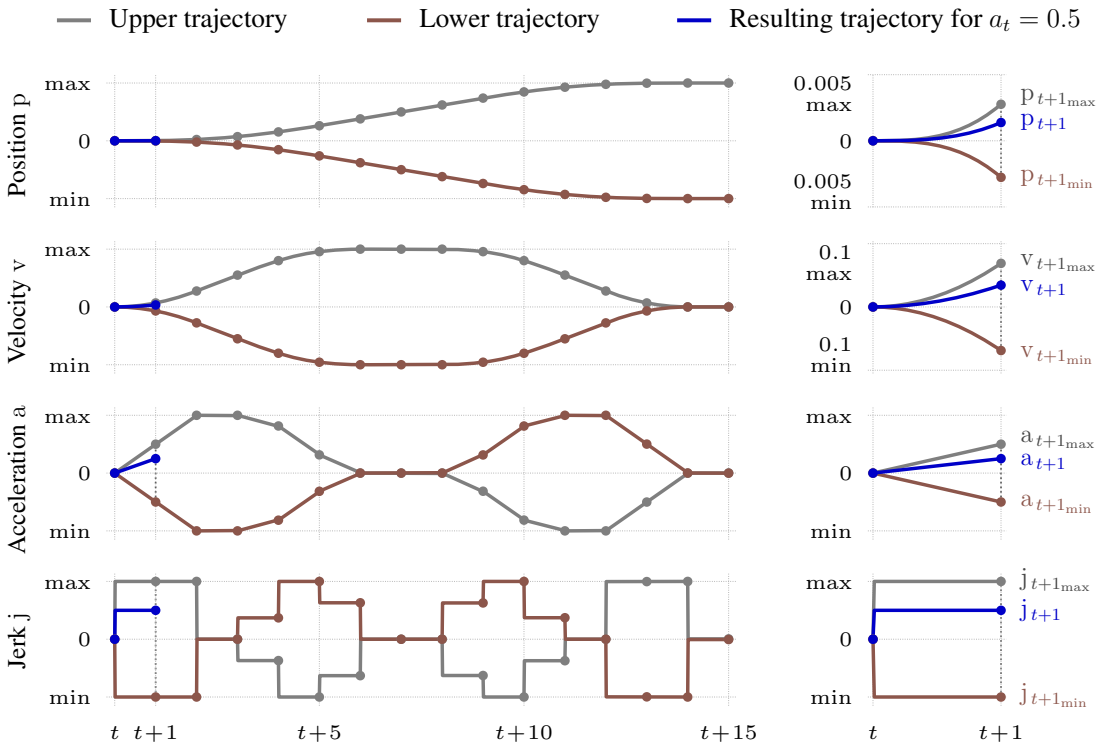
$$\mathrm{p}_{\min} \leq \varphi_d \leq \mathrm{p}_{\max} \tag{3.1}$$
$$\mathrm{v}_{\min} \leq \dot{\varphi}_d \leq \mathrm{v}_{\max} \tag{3.2}$$
$$\mathrm{a}_{\min} \leq \ddot{\varphi}_d \leq \mathrm{a}_{\max} \tag{3.3}$$
$$\mathrm{j}_{\min} \leq \dddot{\varphi}_d \leq \mathrm{j}_{\max} \tag{3.4}$$

This chapter addresses the problem of mapping actions from an RL agent to trajectory setpoints so that the kinematic constraints (3.1) - (3.4) are always respected. The maximum and minimum values are usually specified by the robot manufacturer. However, depending on the desired learning task, lower values can be selected, e.g., to additionally restrict the workspace or the velocity of the robot. An accurate tracking of the setpoints is only possible if the generated trajectory is collision-free. In addition, the robot joints must be able to produce the torque requested by the trajectory controller. To make sure that these assumptions are met, a method to avoid collisions and torque limit violations is introduced in chapter 4.

3.2. Relation to previous studies

As explained in section 2.1.4.2 and shown in Figure 2.6, previous studies typically specify a position setpoint φ_d or a velocity setpoint $\dot{\varphi}_d$ based on the action chosen by an RL agent. Between decision steps, the selected setpoint is kept constant. In modern model-free RL, actions are typically generated based on the output signals of a neural network. When using a continuous action space, a squashing function like *tanh* can be used to ensure that an action does not exceed a certain value range. Consequently, it is straightforward to ensure that the position constraint (3.1) is met when mapping actions to constant position setpoints. However, this mapping leads to a violation of the velocity constraint (3.2), as the position setpoints are discontinuous. When specifying constant velocity setpoints, the velocity limit is easy to meet. However, since the velocity setpoints are discontinuous, the resulting acceleration setpoints are unbounded. In addition, care must be taken to ensure that the resulting position setpoints, which can be computed by integration, stay within the position limits. To address the problem of discontinuities, an action can be mapped to a constant jerk setpoint $\dddot{\varphi}_d$. The unbounded derivative of the jerk usually does not pose a practical problem. However, the key research question that arises is how to map an action to a jerk setpoint without violating the position, velocity, and acceleration limits.

Previous studies have examined compliance with kinematic joint constraints in the context of generating time-optimal trajectories. As outlined in section 2.1.1, motion libraries like *Reflexxes* [86] can be used to compute a time-optimal trajectory to a kinematic target state while satisfying the constraints (3.1) - (3.4). However, directly specifying a kinematic target state (φ_{d_T}, $\dot{\varphi}_{d_\mathrm{T}}$, $\ddot{\varphi}_{d_\mathrm{T}}$) based on an action of an RL agent is not straightforward for three reasons: First, the required time to reach the desired target state varies, while previous studies in RL typically assume a constant time between decision steps. Second, φ_{d_T}, $\dot{\varphi}_{d_\mathrm{T}}$, and $\ddot{\varphi}_{d_\mathrm{T}}$ can be selected independently. In previous studies, however, an action usually consists of a single scalar value per robot joint. Third, as shown in



(a) Upper and lower trajectory shown until standstill.

(b) Detailed view for t to $t+1$

Figure 3.1.: With the proposed action mapping, an action $a_t \in [-1, 1]$ determines a trajectory from t to $t+1$, which lies in between a lower and an upper trajectory.

Figure 2.1, small deviations in the target state can have a strong influence on the generated trajectory. However, reinforcement learning is usually easier if a small deviation in the selected action leads to a small deviation in the generated trajectory. In this work, concepts from the time-optimal generation of robot trajectories are used to address the above-mentioned research problem of mapping an action to a constraint-satisfying trajectory. As usual in previous studies, the time between decision steps is assumed to be constant and the movement between two decision steps is specified by a single scalar per robot joint.

3.3. An action mapping considering kinematic constraints

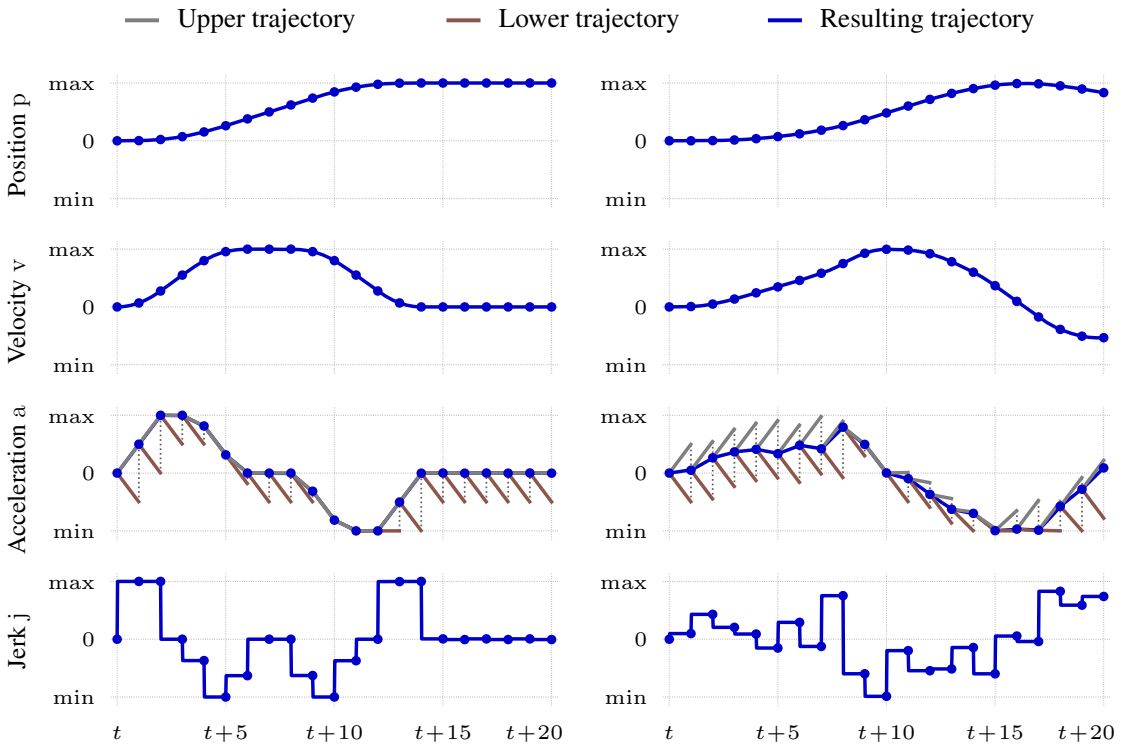
In this work, an action mapping is proposed ensuring that the resulting trajectories do not violate the kinematic constraints (3.1) - (3.4). Each action consists of one scalar $\in [-1, 1]$ per robot joint. This value range is common when using neural networks and can be enforced using the squashing function *tanh*. In Figure 3.1, the basic principle of the action mapping is visualized for a one-dimensional trajectory. For the specific problem considered in this work, the position p, velocity v, acceleration a, and jerk j correspond to the setpoints φ_d, $\dot{\varphi}_d$, $\ddot{\varphi}_d$, and $\dddot{\varphi}_d$ of a robot joint. In the one-dimensional case, an action consists of a single scalar. However, the proposed principle can be applied to robotic systems with any number of joints by using a multi-dimensional action vector.

The basic idea is to make use of a lower trajectory and an upper trajectory, which depend on the current kinematic state ($\mathrm{p_t}$, $\mathrm{v_t}$, $\mathrm{a_t}$). The trajectories need to comply with the kinematic constraints (3.1) - (3.4) over an infinite time horizon. As shown in Figure 3.1, this condition can be considered fulfilled if both trajectories end in a stationary state without violating a constraint. At time step t, an action a_t determines the resulting trajectory for the following time interval from t to $t+1$. When selecting $a_t = -1$, the resulting trajectory follows the lower trajectory. Similarly, an action $a_t = 1$ results in the upper trajectory being followed. All intermediate action values lead to a trajectory lying between the lower and the upper trajectory. More precisely, the selected action determines the relative distance to the lower and upper trajectory at each point in time. As illustrated in Figure 3.1 and derived in section A of the appendix, the selected relative distance applies equally to the position, velocity, acceleration, and jerk setpoints. However, for reasons of presentation, the following explanations focus on the resulting acceleration setpoints. In particular, the desired acceleration setpoint a_{t+1}, which results from action a_t, is computed as follows:

$$\mathrm{a}_{t+1} = \mathrm{a}_{t+1_{\min}} + \frac{1+a_t}{2} \cdot \left(\mathrm{a}_{t+1_{\max}} - \mathrm{a}_{t+1_{\min}}\right), \tag{3.5}$$

where $\mathrm{a}_{t+1_{\min}}$ and $\mathrm{a}_{t+1_{\max}}$ are the accelerations of the lower and the upper trajectory at time step $t+1$. In the specific example shown in Figure 3.1, an action $a_t = 0.5$ is selected. The factor $\frac{1+a_t}{2} = 0.75$ in equation (3.5) indicates that the distance of the resulting trajectory to the lower trajectory is set to $75\,\%$ of the distance between the lower and the upper trajectory. In this work, the jerk of the upper and the lower trajectory is constant between decision steps. As a result, the jerk of the trajectory resulting from action a_t is also constant and the acceleration a_t at time step t can be linearly connected to the computed acceleration a_{t+1} at time step $t+1$. Knowing the course of the acceleration setpoints, the corresponding position, velocity, and jerk setpoints can be calculated by integration or differentiation. A trajectory that always lies between two other trajectories will not go beyond the maximum value or the minimum value of these two trajectories. Consequently, the resulting trajectory complies with the kinematic conditions (3.1) - (3.4) if the upper and lower trajectory do so.

Once time step $t+1$ is reached, the upper and lower trajectory are recomputed based on the kinematic state ($\mathrm{p_{t+1}}$, $\mathrm{v_{t+1}}$, $\mathrm{a_{t+1}}$). To determine the next time interval of the resulting trajectory, the procedure described above is repeated using action a_{t+1}. The proposed action mapping requires that a valid upper and lower trajectory is known at each decision step. In the borderline case, however, both trajectories can coincide, so that there is only one option to continue the trajectory, which is chosen independently of the selected action. As shown in Figure A.1 in the appendix, keeping the same relative distance to the initial upper and lower trajectory as during the time interval from t to $t+1$ will also produce a valid trajectory for the subsequent time intervals. For that reason, there is at least one feasible way to continue the movement at $t+1$ if an upper and lower trajectory is known for time step t. Figure 3.2 shows two exemplary trajectories resulting from choosing $a = 1.0$ or a random action $a \in [-1, 1]$ at each decision step. When selecting $a = 1.0$, the upper trajectory is followed. Consequently, the resulting trajectory in Figure 3.2a, corresponds to the upper trajectory shown in Figure 3.1. At time step $t+10$, the upper and lower trajectory coincide, as otherwise either $\mathrm{p_{max}}$ or $\mathrm{a_{min}}$ would be exceeded. When training a policy using model-free RL, actions at the beginning of the training process are typically selected at random. Figure 3.2b, shows an exemplary trajectory resulting from random



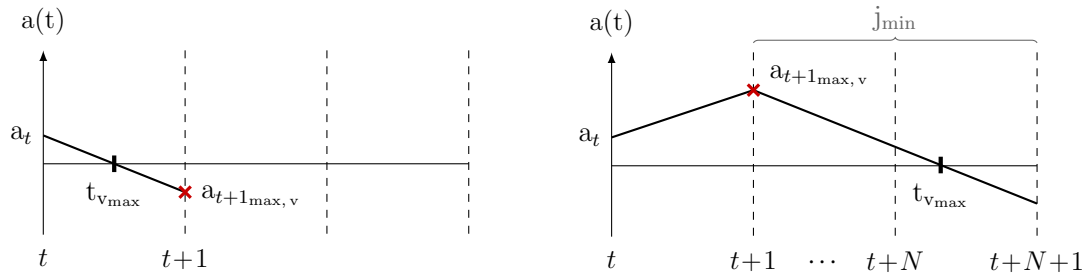
(a) Selecting $a = 1.0$ at each decision step. **(b)** Selecting a random action at each decision step.

Figure 3.2.: Trajectories when selecting $a = 1.0$ and random actions $a \in [-1, 1]$ at each decision step. In both cases, the resulting trajectories come close to the kinematic limits, but do not exceed them. The upper and lower trajectories are recomputed at each decision step. For reasons of clarity, they are only shown for accelerations and only for the first subsequent time interval.

actions. It can be seen that the generated random trajectory comes close to the kinematic limits. However, as expected, the limits are not exceeded.

3.4. Determining an upper and a lower trajectory

The presented action mapping is based on an upper and lower trajectory, which must both satisfy the desired kinematic constraints. However, this requirement does not yet fully specify how these trajectories should be selected. As can be seen in Figure 3.1b, the range of feasible kinematic setpoints at $t + 1$ depends on the lower and upper trajectory computed at time step t. For example, the resulting acceleration a_{t+1} will be within the range $[\mathrm{a}_{t+1_{\min}}, \mathrm{a}_{t+1_{\max}}]$, where $\mathrm{a}_{t+1_{\min}}$ and $\mathrm{a}_{t+1_{\max}}$ are the accelerations of the lower and the upper trajectory at time step $t + 1$. To ensure that the action mapping can generate a wide range of trajectories, the range $[\mathrm{a}_{t+1_{\min}}, \mathrm{a}_{t+1_{\max}}]$ should be as large as possible. For this reason, the method presented below aims to find the smallest possible value for $\mathrm{a}_{t+1_{\min}}$ and the highest possible value for $\mathrm{a}_{t+1_{\max}}$. The implementation used in this thesis ensures that the jerk between decision steps is always constant. Consequently, the course of the lower and upper trajectory is fully specified for the time interval from t to $t + 1$ when knowing $\mathrm{a}_{t+1_{\min}}$ and $\mathrm{a}_{t+1_{\max}}$. Since both trajectories are recomputed at each



(a) Selected acceleration profile for $v_t \geq v_{max} - \frac{1}{2} \cdot a_t \cdot \Delta t$

(b) Selected acceleration profile for $v_t < v_{max} - \frac{1}{2} \cdot a_t \cdot \Delta t$

Figure 3.3.: Based on the illustrated acceleration profiles, an equation to compute $a_{t+1_{max,v}}$ can be derived. Which acceleration profile to select depends on the velocity v_t and the acceleration a_t at the current time step t. The figure is adapted from [82].

decision step, it is not necessary to compute their further course for subsequent time intervals. As a result, the problem of finding a suitable upper and lower trajectory at time step t reduces to finding $a_{t+1_{min}}$ and $a_{t+1_{max}}$. In the following, the computation of $a_{t+1_{max}}$ is explained. However, the same principle can be used to compute $a_{t+1_{min}}$. With the proposed action mapping, the acceleration $a_{t+1_{max}}$ should be the highest possible acceleration at time step $t+1$. Assuming that $a_{t+1_{max}}$ is selected, there must be a way to continue the trajectory at $t+1$ without violating the specified jerk, acceleration, velocity, and position limits. To determine $a_{t+1_{max}}$, a maximum acceleration $a_{t+1_{max,j}}$, $a_{t+1_{max,a}}$, $a_{t+1_{max,v}}$, and $a_{t+1_{max,p}}$ is computed for each of the individual constraints. The smallest of these values is assigned to $a_{t+1_{max}}$:

$$a_{t+1_{max}} = \min(a_{t+1_{max,j}},\ a_{t+1_{max,a}},\ a_{t+1_{max,v}},\ a_{t+1_{max,p}}) \tag{3.6}$$

Determining $a_{t+1_{max,j}}$ and $a_{t+1_{max,a}}$ is straightforward:

$$a_{t+1_{max,j}} = a_t + j_{max} \cdot \Delta t \tag{3.7}$$

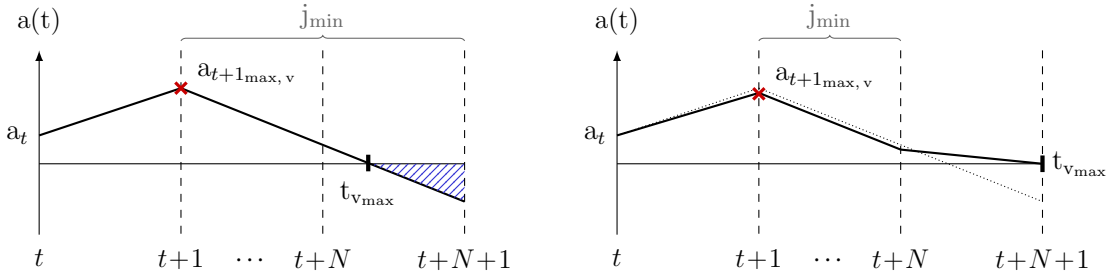
$$a_{t+1_{max,a}} = a_{max}, \tag{3.8}$$

where Δt is the time between t and $t+1$. The procedure for calculating $a_{t+1_{max,v}}$ and $a_{t+1_{max,p}}$ is more complicated, as a case analysis must be conducted. For the following calculations, the variable t indicates a continuous time. The continuous course of accelerations is denoted as $a(t)$. Without loss of generality, the initial time step t is assigned to $t = 0$. Consequently, the initial acceleration a_t, velocity v_t, and position p_t can also be referred to as a_0, v_0, and p_0. As shown in Figure 3.3, two different cases need to be distinguished to determine $a_{t+1_{max,v}}$. In the first case, the velocity limit v_{max} is reached between t and $t+1$, in the second case it is reached after $t+1$. To ensure that the maximum velocity v_{max} is not exceeded, the acceleration must be zero at the moment v_{max} is reached. This condition can be expressed by the following two equations:

$$v_0 + \int_0^{t_{v_{max}}} a(t)\ dt = v_{max} \tag{3.9}$$

$$a(t_{v_{max}}) = 0, \tag{3.10}$$

where $t_{v_{max}}$ is the continuous time at which v_{max} is reached. Based on the acceleration profiles shown in Figure 3.3, the equation system (3.9) - (3.10) can be solved for $a_{t+1_{max,v}}$.



(a) As indicated by the area hatched in blue, the velocity at time step $t+N+1$ is smaller than v_{max} when using this profile.

(b) By selecting a slightly smaller value for $a_{t+1_{max,v}}$, oscillations can be avoided as v_{max} is reached at a discrete time step.

Figure 3.4.: To avoid oscillations after reaching v_{max}, the acceleration profile shown in (a) is slightly adjusted so that v_{max} is reached at a discrete time step. The figure is adapted from [82].

In this work, the Python library SymPy [111] for symbolic mathematics is used to derive the following formulas:

$$a_{t+1_{max,v}} = \begin{cases} a_t \cdot \left(1 - \frac{1}{2} \cdot \frac{a_t \cdot \Delta t}{v_{max} - v_t}\right) & \text{if } v_t \geq v_{max} - \frac{1}{2} \cdot a_t \cdot \Delta t \\ \frac{j_{min} \cdot \Delta t}{2} \cdot \left(1 - \sqrt{1 + \frac{8 \cdot (v_t - v_{max}) + 4 \cdot a_t \cdot \Delta t}{j_{min} \cdot \Delta t^2}}\right) & \text{otherwise} \end{cases} \tag{3.11}$$

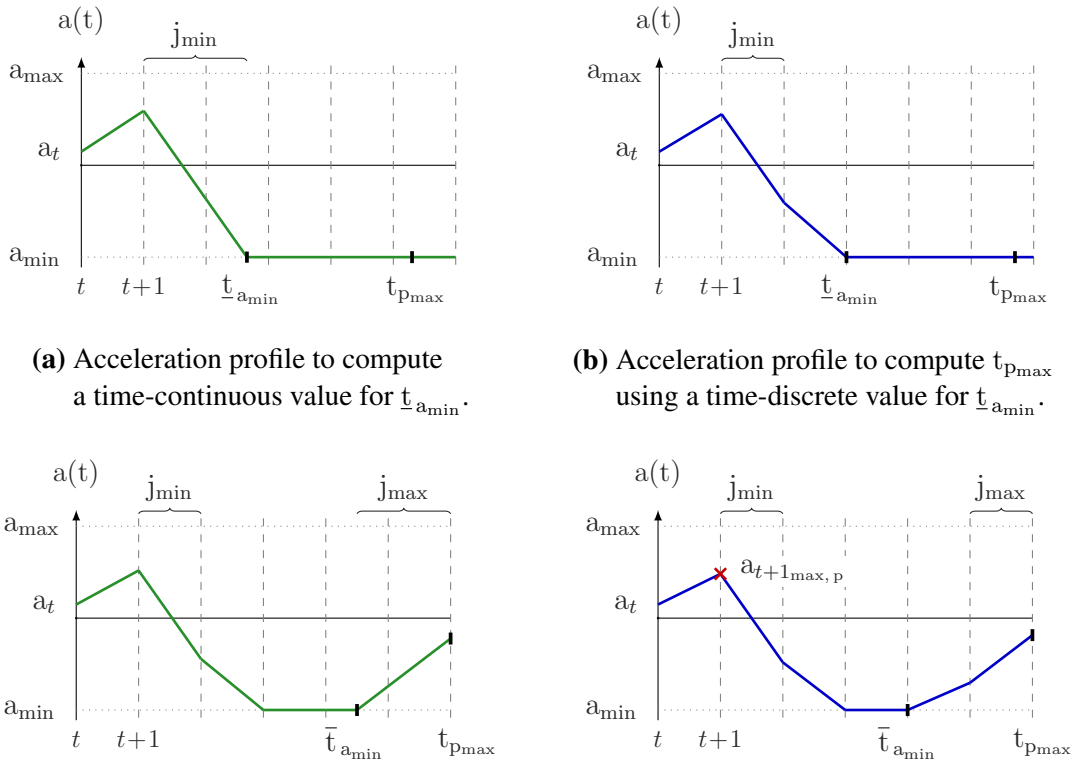
While the presented formulas lead to a valid solution for $a_{t+1_{max,v}}$, the resulting upper trajectory might start to oscillate when being close to v_{max}. To prevent these oscillations, the acceleration profile shown in Figure 3.3b is slightly adjusted so that $t_{v_{max}}$ coincides with a discrete time step. The resulting profile can be seen in Figure 3.4b. As with the initial profile, an analytical formula to calculate $a_{t+1_{max,v}}$ can be derived using SymPy.

In order to additionally consider position limits, $a_{t+1_{max,p}}$ is computed. The basic principle is similar to the calculation of $a_{t+1_{max,v}}$. Assuming that a_{t+1} is set to $a_{t+1_{max,p}}$, there must still be a way to avoid violating the position limit p_{max}. Given that $a_{t+1_{max,p}}$ should be the highest feasible acceleration, p_{max} should be reached but not exceeded. Consequently, the velocity at $t_{p_{max}}$, the time at which p_{max} is reached, must be zero:

$$p_0 + v_0 \cdot t_{p_{max}} + \int_0^{t_{p_{max}}} \int_0^{t} a(t)\, dt\, dt = p_{max} \tag{3.12}$$

$$v_0 + \int_0^{t_{p_{max}}} a(t)\, dt = 0 \tag{3.13}$$

Compared to computing $a_{t+1_{max,v}}$, a larger number of different cases must be distinguished. In Figure 3.5, the stepwise computation of $a_{t+1_{max,p}}$ is illustrated for one exemplary kinematic state (p_t, v_t, a_t). In a first step, the acceleration profile shown in Figure 3.5a is used to determine the continuous time $\underline{t}_{a_{min}}$, at which the minimum acceleration a_{min} is reached. To ensure that the resulting acceleration at $t+1$ is as high as possible, the minimum jerk j_{min} is applied between $t+1$ and $\underline{t}_{a_{min}}$ As of $\underline{t}_{a_{min}}$, the acceleration is held at a_{min}. Since the jerk between time steps is required to be constant, this profile cannot be used to compute $a_{t+1_{max,p}}$. However, as shown in Figure 3.5b, a profile can be derived, in which $\underline{t}_{a_{min}}$ coincides with the next discrete time step. While



(a) Acceleration profile to compute a time-continuous value for $\underline{t}_{a_{min}}$.

(b) Acceleration profile to compute $t_{p_{max}}$ using a time-discrete value for $\underline{t}_{a_{min}}$.

(c) Acceleration profile to compute $\overline{t}_{a_{min}}$ using a time-discrete value for $t_{p_{max}}$.

(d) Acceleration profile to compute $a_{t+1_{max,\,p}}$ using a time-discrete value for $\overline{t}_{a_{min}}$.

Figure 3.5.: The stepwise computation of $a_{t+1_{max,\,p}}$ shown for one exemplary kinematic state $(p_t,\, v_t,\, a_t)$. While the acceleration profile shown in (b) would already lead to a valid solution for $a_{t+1_{max,\,p}}$, the steps (c) and (d) are performed to ensure that p_{max} is reached at a discrete time step, which reduces oscillations. The figure is adapted from [76].

this profile could already be used to compute $a_{t+1_{max,\,p}}$, oscillations can be reduced if p_{max} is reached at a discrete time step. Therefore, $t_{p_{max}}$ is shifted to the next discrete time step. The intermediate step shown in Figure 3.5c is performed to derive the time-discrete acceleration profile shown in Figure 3.5d. Finally, this profile is used to compute the desired acceleration $a_{t+1_{max,\,p}}$, which is marked by a red cross.

Once $a_{t+1_{max,\,j}}$, $a_{t+1_{max,\,a}}$, $a_{t+1_{max,\,v}}$, and $a_{t+1_{max,\,p}}$ are determined, $a_{t+1_{max}}$ can be calculated using equation (3.6). If the maximum acceleration $a_{t+1_{max}}$ is selected, the resulting trajectory follows the upper trajectory. As shown in Figure 3.1, the upper trajectory finally reaches the maximum position p_{max} at a velocity of zero. While the jerk between decision steps is kept constant, the upper trajectory resembles a time-optimal trajectory to the target state $(p_T = p_{max},\, v_T = 0,\, a_T = 0)$, which establishes a relation between the proposed action mapping and the time-optimal trajectories introduced in section 2.1.1. The calculation of $a_{t+1_{min}}$ is carried out analogously to $a_{t+1_{max}}$. After computing $a_{t+1_{min,\,j}}$, $a_{t+1_{min,\,a}}$, $a_{t+1_{min,\,v}}$, and $a_{t+1_{min,\,p}}$, the largest of these values is assigned to $a_{t+1_{min}}$:

$$a_{t+1_{min}} = \max(a_{t+1_{min,\,j}},\, a_{t+1_{min,\,a}},\, a_{t+1_{min,\,v}},\, a_{t+1_{min,\,p}}) \tag{3.14}$$

Knowing the current kinematic state $(p_t,\, v_t,\, a_t)$, the range $[a_{t+1_{min}},\, a_{t+1_{max}}]$, and the selected action a_t, the resulting trajectory from t to $t+1$ is fully specified. Hence, the next step is to evaluate the proposed action mapping in the context of model-free RL.

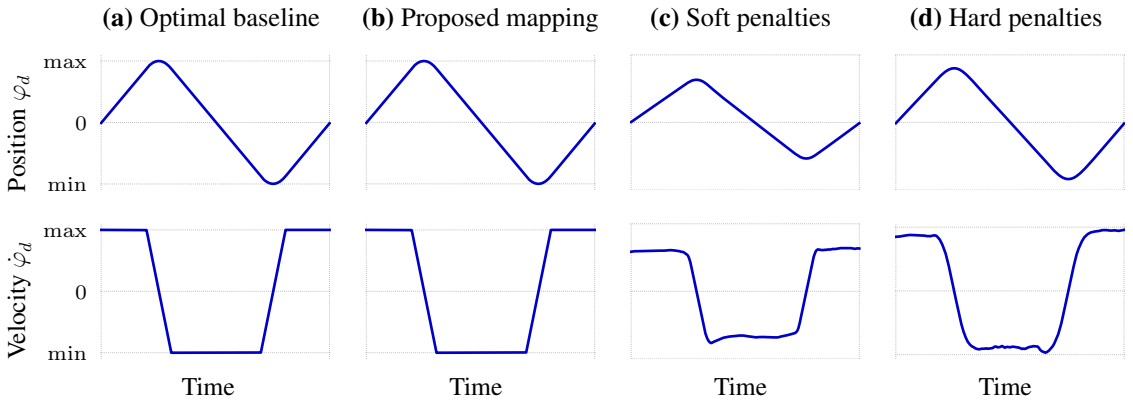


Figure 3.6.: In the *velocity maximization task*, a robot is trained to maximize the average absolute velocity of each robot joint. The figure shows the resulting periodic movements for one exemplary joint. As can be seen in (a), an optimal trajectory oscillates between the position limits. When using the proposed action mapping, the resulting trajectory shown in (b) is very similar to the optimal baseline. In contrast, soft or hard penalties discourage the robot from operating close to the kinematic limits, which reduces the task performance.

3.5. Evaluation of the proposed action mapping

In the following section, the proposed action mapping is evaluated by learning three robot tasks using model-free RL. As a first step, it is analyzed whether the action mapping can be used to learn trajectories that fully utilize the kinematic capabilities of the robot joints without exceeding them. For that purpose, a 7-DOF *KUKA iiwa* robot is trained to generate movements that maximize the average absolute joint velocity over time. While in this task, each joint can be considered as independent, the next two tasks require temporal coordination between the robot joints. More specifically, the proposed action mapping is used to learn how to *track reference paths* and to *adjust reference trajectories*. In addition to evaluating the learning performance achieved in simulation, a sim-to-real transfer is conducted for these two tasks. This involves evaluating whether the required calculations can be carried out in real time and whether the generated trajectories can be executed on a real robot.

3.5.1. Velocity maximization

The goal of the *velocity maximization task* is to learn trajectories for a 7-DOF industrial robot that maximize the average absolute joint velocity over time without violating the kinematic constraints (3.1) - (3.4). In order to learn the task via model-free RL, a Markov decision process $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ is defined. A state $s_t \in \mathcal{S}$ contains the current position, velocity, and acceleration setpoint $(\varphi_{d_t}, \dot{\varphi}_{d_t}, \ddot{\varphi}_{d_t})$ of each robot joint. The setpoints are normalized with respect to the specified kinematic limits. Consequently, the resulting state for a robot with seven joints is a vector of 21 dimensions, each of them $\in$ [-1,1]. An action $a_t \in \mathcal{A}$ consists of one scalar $\in$ [-1,1] per robot joint. As explained earlier in this chapter, the scalar is used to determine an acceleration setpoint $\ddot{\varphi}_{d_{t+1}} := \mathrm{a}_{t+1}$ according to equation (3.5). For this purpose, the range of feasible setpoints $[\mathrm{a}_{t+1_{\min}}, \mathrm{a}_{t+1_{\max}}]$ is computed as explained in section 3.4.

Table 3.1.: Quantitative evaluation of the *velocity maximization task*, comparing the proposed action mapping with an optimal baseline and the use of penalties. The specified velocities and positions refer to normalized absolute values. For each method, 1000 trajectories with a duration of 5 seconds are computed. The violation rate indicates the proportion of trajectories that violate the position or velocity constraints. Adapted from [76].

	Method	Mean |velocity|	Max |velocity|	Max |position|	Violation rate
$f_D = 240\,\mathrm{Hz}$	Optimal baseline	0.92	1.00	1.00	0.0 %
	Proposed mapping	0.91	1.00	1.00	0.0 %
	Soft penalties	0.51	0.90	1.25	8.0 %
	Hard penalties	0.71	1.56	1.43	52.7 %
$f_D = 20\,\mathrm{Hz}$	Optimal baseline	0.92	1.00	1.00	0.0 %
	Proposed mapping	0.88	1.00	1.00	0.0 %
	Soft penalties	0.44	1.14	0.93	1.2 %
	Hard penalties	0.58	1.69	1.14	41.5 %

With $\dot{\varphi}_{d_{t+1,i}}$ being the resulting velocity setpoint of the i-th joint at time step $t+1$, the reward function is specified as follows:

$$R(s_t, a_t) = \frac{1}{7} \sum_{i=1}^{7} \left| \dot{\varphi}_{d_{t+1,\,i}} \right| \tag{3.15}$$

For each experiment, a decision frequency f_D is specified, which determines the time between decision steps $\Delta\mathrm{t}$:

$$\Delta\mathrm{t} = \frac{1}{f_D} \tag{3.16}$$

As visualized in Figure 1.3, actions are generated by a fully connected neural network. The training of the neural network is performed in a simulation environment using a reference implementation of the algorithm PPO [148] provided by the Python library RLlib [96]. To simplify the interaction with RLlib, the implementation of the desired learning task is based on a common interface called *Gym* [23], which was originally introduced by OpenAI. In order to generate training data in parallel, the Ray framework [117] for distributed computing is used.

Figure 3.6 shows exemplary trajectories for the *velocity maximization task* using a decision frequency of $240\,\mathrm{Hz}$. As visualized in (a), an optimal trajectory oscillates between the minimum and maximum position. The trajectory shown in (b) is generated by a neural network trained with the proposed action mapping. It can be seen that the trajectory closely resembles the optimal baseline. For the purpose of comparison, additional neural networks are trained without the proposed action mapping. To this end, the range of feasible setpoints for a_{t+1} is set to $[\mathrm{a}_{\min}, \mathrm{a}_{\max}]$ rather than $[\mathrm{a}_{t+1_{\min}}, \mathrm{a}_{t+1_{\max}}]$. In order to avoid exceeding the position and velocity limits, the reward function is adjusted. When using hard penalties, the reward is set to zero if any of the joints violates a position or velocity limit. With soft penalties, the reward is gradually reduced if a joint velocity or position exceeds $50\,\%$ of the corresponding maximum value. As shown in (c) and (d), the

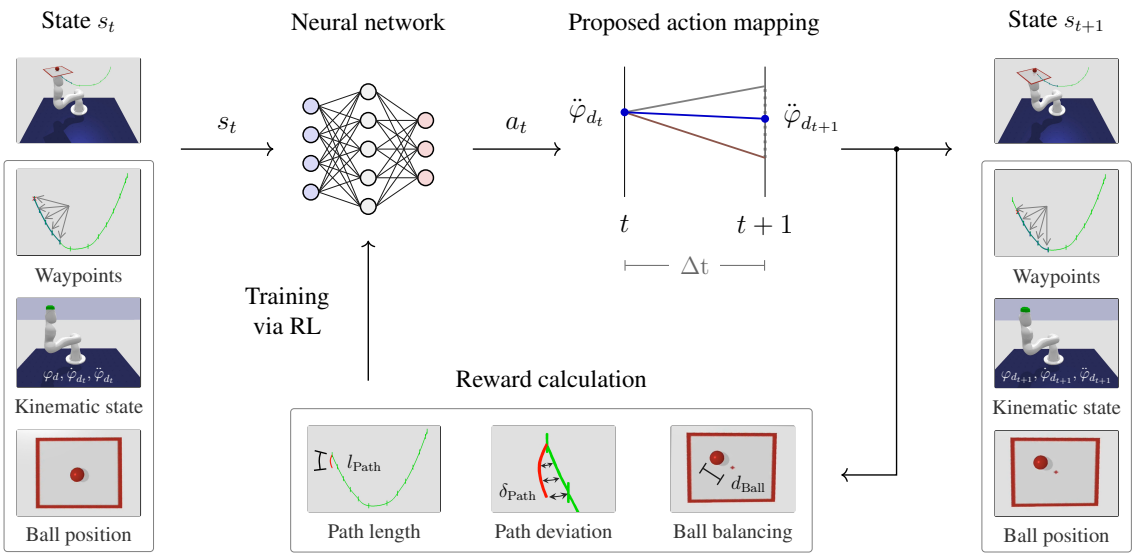


Figure 3.7.: The tracking of a reference path with an industrial robot shown for one exemplary time step t. The figure illustrates the components of each state, the trajectory generation and the relevant metrics for the reward calculation when tracking a reference path shown in green with the additional objective of balancing a ball on a plate. The figure is adapted from [78].

resulting movements are still similar to the optimal baseline. However, since the trajectories keep a certain distance from the kinematic limits, the resulting task performance is lower. Table 3.1 shows a quantitative analysis of the results obtained for $f_D = 240\,\text{Hz}$ and $f_D = 20\,\text{Hz}$. In both cases, the performance achieved with the proposed action mapping is close to the optimal baseline. Moreover, the position and velocity limits are not violated. When using penalties, the resulting task performance is lower and the kinematic constraints are not strictly respected.

In summary, the trajectories generated with the proposed action mapping complied with the kinematic constraints and led to an average absolute joint velocity close to the optimal baseline. Contrary to that, the use of penalties did not strictly prevent violations of the kinematic limits and had a negative impact on the resulting task performance.

3.5.2. Tracking of reference paths

In this section, the proposed action mapping is used to track reference paths defined in joint space. For this purpose, the problem of quickly following a reference path is formulated as an optimization problem with two objectives: On the one hand, the progress along the path should be maximized, on the other hand, the deviation to the reference path should be minimized. As an additional constraint, the generated trajectories must satisfy the kinematic limits of the robot joints. When using the proposed action mapping, compliance with the kinematic limits is already ensured. For that reason, the resulting learning problem can be considered as unconstrained. However, in contrast to the velocity maximization task, a temporal coordination between the robot joints is required in this scenario. Thus, the RL agent must learn to coordinate all joints collectively. In the literature, the problem of finding a time-optimal trajectory to traverse a reference path is known

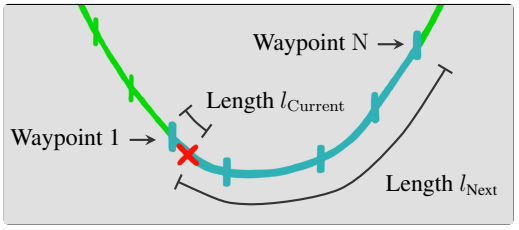


Figure 3.8.: The parameters included in the state to describe the next part of the reference path. Adapted from [78].

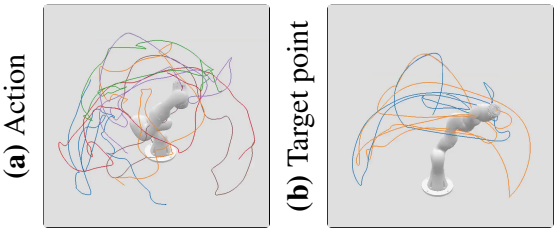
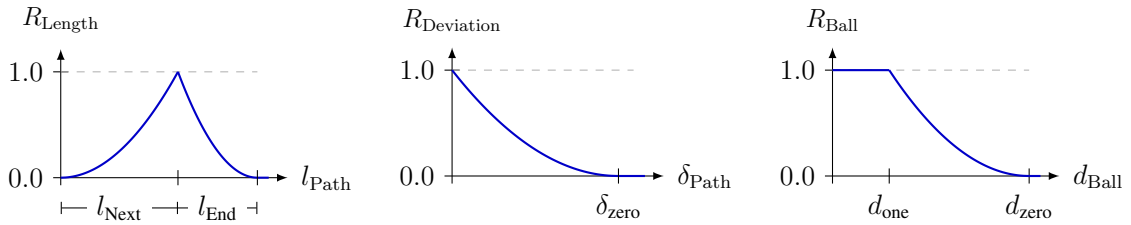


Figure 3.9.: Exemplary reference paths for an industrial robot sampled from the specified datasets. Adapted from [79].

as *time-optimal path parameterization* (TOPP). In contrast to TOPP, the trajectories produced with the presented approach are neither time-optimal nor do they perfectly match the specified reference paths. However, since the trajectory generation is performed in real time, sensory feedback can be taken into account to address additional optimization objectives. In addition, the reference path can be adjusted during motion execution. Using a physics simulator, the presented method is evaluated for a 7-DOF industrial robot and for the humanoid robots ARMAR-4 [11] and ARMAR-6 [12]. For the industrial robot and for the bipedal robot ARMAR-4, experiments with additional optimization objectives are conducted. More specifically, the industrial robot is trained to additionally balance a ball on a plate and the bipedal robot is trained to avoid falling over. An overview of the trajectory generation for the ball balancing task with an industrial robot is presented in Figure 3.7. The desired reference path is shown in green. Note that while the reference path is defined in joint space, the figures in this section illustrate the reference path in Cartesian space to simplify the visualization.

State space To formalize the learning problem, a Markov decision process $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ is defined. As for the velocity maximization task, a state $s_t \in \mathcal{S}$ contains the kinematic state of each robot joint $(\varphi_{d_t}, \dot{\varphi}_{d_t}, \ddot{\varphi}_{d_t})$. In addition, information on the upcoming part of the reference path is provided. More specifically, a reference path is represented by a cubic spline specified by a set of waypoints defined in joint space. Figure 3.8 visualizes how the reference path is described in the state. While the upcoming part of the reference path is shown in light blue, other parts are shown in green. The red cross serves as an indicator of the current progress along the reference path. To specify the upcoming part of the reference path, a state includes the waypoint preceding the red cross and $\mathrm{N}-1$ subsequent waypoints. In addition, the path lengths l_{Current} and l_{Next} are part of the state. To evaluate the proposed method, reference paths with different path characteristics are used. One path of the dataset used for ball balancing is visualized in Figure 3.7. Figure 3.9 shows paths from two additional datasets labeled as *action dataset* and *target point dataset*. The paths of the *action dataset* are generated by selecting random actions using the proposed action mapping. In contrast, the paths from the *target point dataset* result from movements between randomly selected target points in Cartesian space. If additional task objectives are taken into account, the state contains further sensor data. For the ball balancing task, the state includes the current position and the initial position of the ball relative to the plate. When keeping ARMAR-4 in balance, the state additionally incorporates the position and the orientation of the robot's pelvis.

Action space Since the proposed action mapping is used, an action $a_t \in \mathcal{A}$ is composed of one scalar $\in$ [-1,1] per controlled robot joint. In the experiments with the indus-



(a) Path length reward **(b)** Path deviation reward **(c)** Ball balancing reward

Figure 3.10.: The reward of the path tracking task is a weighted sum of the reward components R_{Length} and $R_{\text{Deviation}}$. When considering additional objectives such as ball balancing, further components like R_{Ball} are added. The figure is adapted from [78].

trial robot KUKA iiwa, all seven joints are actuated. With ARMAR-6, both arms and the torso are controlled, resulting in a total of 17 actuated joints. For ARMAR-4, two different configurations are evaluated. Without the additional objective to balance the robot, only the upper body is controlled, leading to 18 actuated joints. With the balancing objective, the upper body and the legs are actuated, resulting in a total of 30 controlled joints.

Reward function The reward function R is a weighted sum of a path length reward R_{Length} and a path deviation reward $R_{\text{Deviation}}$:

$$R = \alpha \cdot R_{\text{Length}} + \beta \cdot R_{\text{Deviation}}, \tag{3.17}$$

where α and β are weighing factors. In order to consider additional task objectives like ball balancing, further reward components like R_{Ball} can be added to the reward function:

$$R = \alpha \cdot R_{\text{Length}} + \beta \cdot R_{\text{Deviation}} + \gamma \cdot R_{\text{Ball}} \tag{3.18}$$

As shown in Figure 3.10, the computation of R_{Length}, $R_{\text{Deviation}}$, and R_{Ball} is based on quadratic functions. The path length reward R_{Length} at time step t depends on l_{Path}, the length of the path resulting from executing action a_t. In the reward calculation box in Figure 3.7, the reference path is shown in green and the path resulting from action a_t is shown in red. As indicated, l_{Path} is the length of the red path. Since the reference path should be traversed quickly, the path length reward is higher if l_{Path} increases. However, this only applies as long as l_{Path} is smaller than l_{Next}, the length of the reference path included in the state. At the end of a reference path, l_{Next} is zero and the decreasing reward encourages the robot to come to a stop. How sharply the reward decreases depends on l_{End}, a predefined parameter that remains constant during the training process. The path deviation reward $R_{\text{Deviation}}$ ensures that the generated path stays close to the reference path. As shown in Figure 3.10b, $R_{\text{Deviation}}$ depends on δ_{Path}, the average Euclidean distance between waypoints sampled along the generated path and waypoints sampled along the reference path. The higher δ_{Path}, the lower the path deviation reward. If δ_{Path} exceeds a threshold δ_{zero}, the path deviation reward remains at zero. The ball balancing reward R_{Ball} at time step t depends on d_{Ball}, the distance between the ball position at $t+1$ and the initial ball position on the plate. As long as d_{Ball} is smaller than a threshold d_{one}, the reward is one. If d_{Ball} exceeds d_{zero}, the reward is zero. Between d_{one} and d_{zero}, the reward gradually decreases. For the balancing task with ARMAR-4, two additional reward components are defined: The first component encourages the robot to keep an upright posture. To this

Table 3.2.: Performance metrics of neural networks trained to track reference paths without additional objectives. The evaluation is based on 1200 reference paths per experiment using a separate test version of the datasets used for training. Adapted from [78].

Experiment	Trajectory duration [s]	Joint position deviation [rad]			Cart. position deviation [cm]		
		mean	max	final	mean	max	final
KUKA iiwa							
• Action dataset	4.28	0.11	0.19	0.09	3.3	7.5	2.7
• Target point dataset	4.99	0.12	0.21	0.12	3.7	8.1	3.8
• Ball balancing dataset	2.44	0.04	0.08	0.03	1.4	3.0	1.4
ARMAR-6							
• Action dataset	4.98	0.14	0.20	0.16	5.5	13.6	6.2
ARMAR-4 (fixed base)							
• Action dataset	5.09	0.14	0.20	0.14	3.3	7.8	3.5
• Target point dataset	5.48	0.14	0.21	0.15	3.6	8.8	3.8

end, the angle between the pelvis of the robot and the vertical z-axis is determined. Small angles are rewarded as they indicate an upright posture. If the legs are used to stabilize the robot, a second component is added, rewarding the robot for keeping a small distance to its initial position with respect to the ground.

Termination During training, each episode ends after a predefined period of time. However, an episode is terminated earlier if the path deviation δ_{Path} exceeds a threshold value, if the ball falls off the plate, or if the bipedal robot ARMAR-4 falls to the ground. Since early termination leads to a lower sum of rewards, the learning algorithm tries to avoid the occurrence of these undesirable events.

Path tracking without additional objectives Table 3.2 shows various performance metrics for neural networks trained to track reference paths without additional task objectives. The experiments were carried out in the physics simulator PyBullet [30] using a decision frequency f_D of $10\,\mathrm{Hz}$. As indicated in Table 3.2, different datasets were used for the training of the neural networks. To prevent overfitting, the reference paths used for the evaluation were taken from a separate test version of the corresponding datasets. When tracking reference paths, a low trajectory duration and a low average joint position deviation are desired. For a better assessment of the results, Table 3.2 additionally shows the corresponding position deviation in Cartesian space. For the industrial robot KUKA iiwa, the reference point used for the conversion from joint space to Cartesian space was the center of the last link. For the humanoid robots, the conversion was conducted with respect to the tip of each hand. When training the KUKA iiwa robot using the *ball balancing dataset*, $\mathrm{N} = 5$ waypoints were included in the state. For all other datasets, N was set to 9. The average Cartesian deviation between the generated paths and the reference paths from the *ball balancing dataset* was $1.4\,\mathrm{cm}$. Using the *action dataset* and the *target point dataset* shown in Figure 3.9, an average Cartesian deviation between $3\,\mathrm{cm}$ and $4\,\mathrm{cm}$ was obtained. In the experiments with ARMAR-4, only the upper body was controlled and the base of the robot was fixed to prevent it from falling over.

Table 3.3.: Performance metrics of the TOPP algorithm TOPP-RA [131]. Adapted from [78].

Experiment	Trajectory duration [s]	Relative trajectory duration [%]	Joint position deviation [rad] max
KUKA iiwa			
• Action dataset	3.36	78.5	< 0.001
• Target point dataset	3.79	76.0	< 0.001
• Ball balancing dataset	1.91	78.3	< 0.001
ARMAR-4 (fixed base)			
• Action dataset	3.31	65.0	< 0.001
• Target point dataset	3.68	67.2	< 0.001

Table 3.4.: Training results for the task objective of maintaining balance with ARMAR-4 while tracking paths from the target point dataset. Adapted from [78].

Experiment	Trajectory duration [s]	Balancing error [%]	Cart. position deviation [cm] mean	max
ARMAR-4 with fixed legs				
• Without balancing objective	5.28	26.1	3.6	8.8
• With balancing objective	5.63	5.3	4.1	9.6
ARMAR-4 with controlled legs				
• With balancing objective	5.59	0.8	4.2	9.8

The *action dataset* and the *target point dataset* for ARMAR-4 were generated according to the same scheme as used for the industrial robot KUKA iiwa. Similarly to the KUKA iiwa, an average Cartesian deviation between $3\,\mathrm{cm}$ and $4\,\mathrm{cm}$ was achieved.

Comparison with TOPP-RA For benchmarking purposes, the performance metrics of the TOPP algorithm TOPP-RA [131] are provided in Table 3.3. The second column shows the relative duration of the trajectories generated with TOPP-RA compared to those produced by the neural networks. It can be seen that TOPP-RA achieves a faster traversal of the reference paths. In addition, the tracking of the reference paths is very accurate. However, in contrast to the proposed action mapping, TOPP-RA does not consider jerk constraints. Moreover, TOPP-RA is an offline method, which means that the reference path must be fully known in advance and that sensory feedback cannot be considered to achieve additional task objectives.

Path tracking with a bipedal robot An example of an additional task objective is balancing a bipedal robot. Table 3.4 shows the impact of the additional balancing objective when training neural networks for ARMAR-4. In the first two experiments, the joints of the legs are fixed. As indicated by the balancing error, the robot falls over in $26.1\,\%$ of all episodes when training a network without the balancing objective. When an upright posture is rewarded, the balancing error reduces to $5.3\,\%$. On the other hand, the average trajectory duration increases slightly. In a third experiment, the neural network can additionally control the legs to stabilize the robot. As a result, the balancing error decreases to $0.8\,\%$. Figure 3.11 illustrates the effect of the additional balancing objective for one exemplary episode.

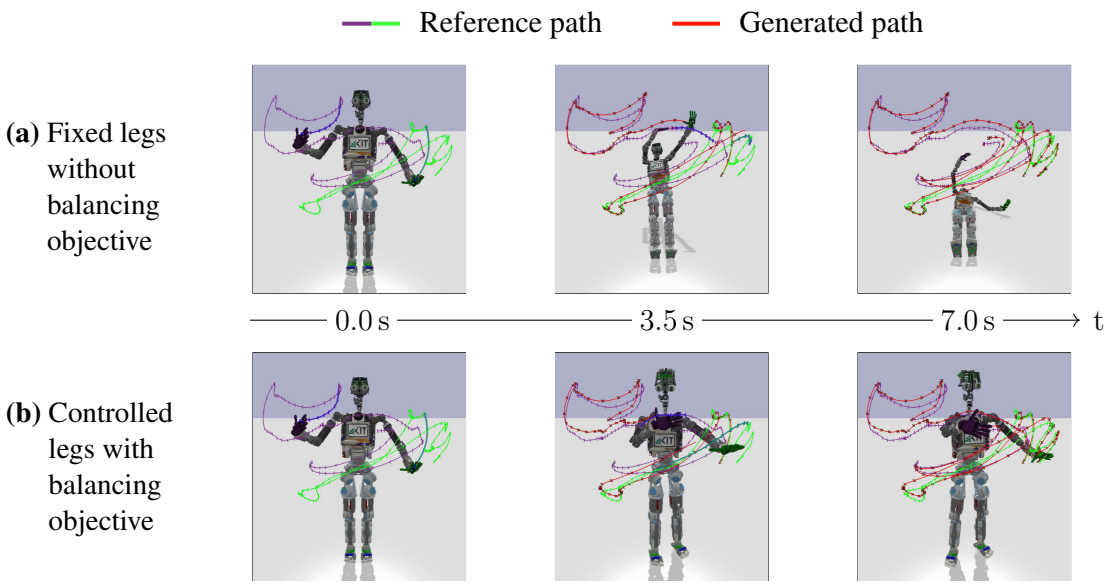


Figure 3.11.: When tracking reference paths with the bipedal robot ARMAR-4, an additional balancing objective can be included in the reward function to prevent the robot from falling over. The figure is adapted from [78].

Table 3.5.: Training results for balancing a ball on a plate with the industrial robot KUKA iiwa using paths from the ball balancing dataset. Adapted from [78].

Experiment	Trajectory duration [s]	Balancing error [%]	Cart. position deviation [cm] mean	max
KUKA iiwa				
• Without balancing objective	2.24	100.0	1.4	3.0
• With balancing objective	2.99	0.3	2.3	4.9

Path tracking while balancing a ball As visualized in Figure 3.7, another additional task objective is to balance a ball on a plate attached to an industrial robot. Table 3.5 shows the training results obtained with and without including the balancing reward R_{Ball} in the reward function. Without the additional balancing reward, the ball falls off the plate in every episode used for the evaluation. When including the balancing reward in the reward function, the balancing only fails in $0.3\,\%$ of all episodes. In return, around $30\,\%$ more time is required to traverse the reference paths and the tracking accuracy decreases.

Sim-to-real transfer To demonstrate that trajectories can be generated in real time, a real KUKA iiwa robot is used to perform a sim-to-real transfer for the ball balancing task. Figure 3.12 shows the resulting balancing performance when tracking a reference path with and without the additional balancing objective. The position of the ball is detected by a resistive touchpad placed on top of a red plate. To prevent the ball from falling to the ground, the plate is surrounded by a blue frame. When using a neural network trained without the balancing objective, the ball quickly touches the frame. In contrast, a network trained with the additional balancing objective manages to keep the ball close to its initial position on the plate. Figure 3.13 shows how the data processing for a decision step $t + 1$ is executed when generating trajectories in real time. Ideally, all calculations would be performed at time step $t + 1$.

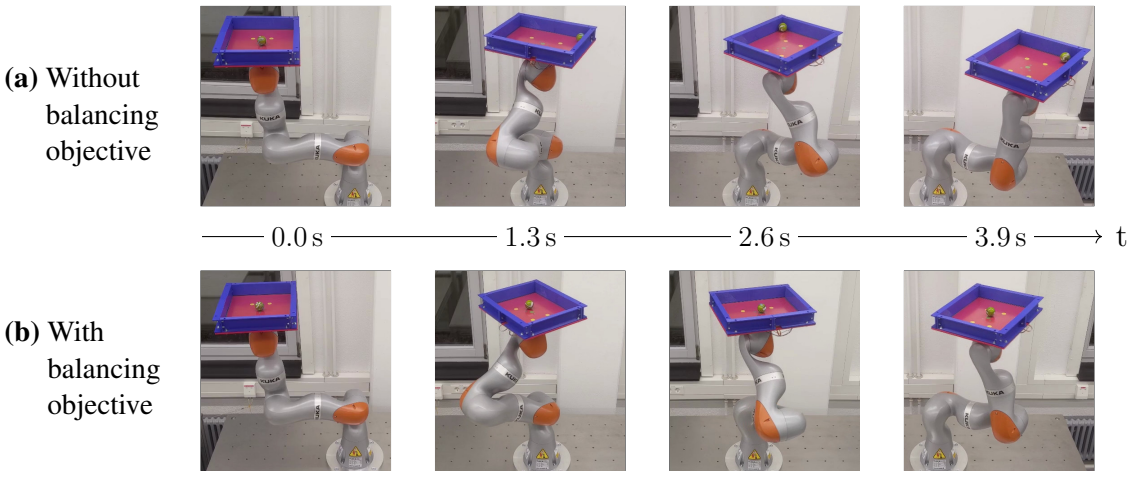


Figure 3.12.: A sim-to-real transfer for the path tracking task using a real KUKA iiwa robot and a resistive touchpad to detect the position of the ball on the red plate. Without the additional objective to balance the ball (a), the reference path is traversed faster but the ball quickly leaves its initial position on the plate. If the objective of balancing the ball is considered in the reward function (b), the ball does not touch the blue frame of the plate. The figure is adapted from [78].

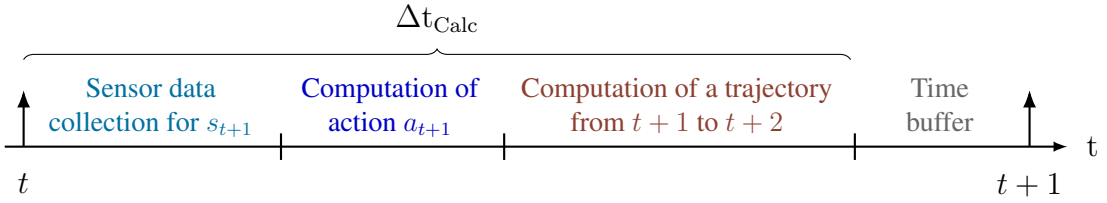
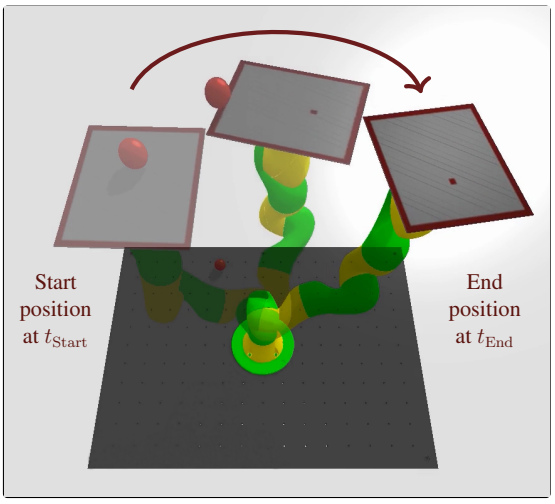


Figure 3.13.: The temporal order of the data processing required for decision step $t+1$ when generating trajectories in real time. To achieve real-time capability, Δt_{Calc} must always be shorter than the time between t and $t+1$. The figure is adapted from [76].

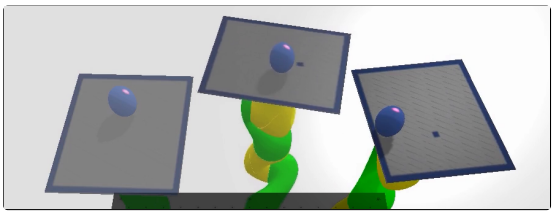
However, as computing power is limited, the data processing must be carried out in the time between t and $t+1$. This also means that sensor data for s_{t+1} must be read out slightly before $t+1$. To ensure that the trajectory can be continued at $t+1$, the time of the data processing Δt_{Calc} must not exceed the time between decision steps Δt. The computational effort can be reduced by selecting a lower decision frequency f_D. However, a lower decision frequency also leads to a less granular control of the generated trajectory.

Table 3.6.: Analysis of the computational effort to simulate path tracking experiments with different robots. Using an Intel i7-8700K as *central processing unit* (CPU), 1200 episodes are simulated per experiment. For each episode, the proportion between the computation time and the resulting trajectory duration is calculated. The highest proportion of all episodes is specified in the table. Adapted from [78].

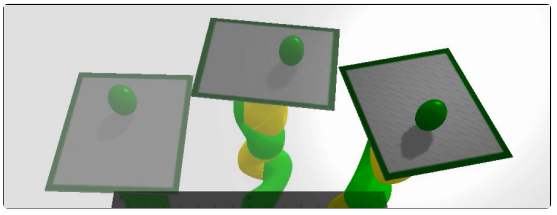
	KUKA iiwa	ARMAR-6	ARMAR-4 with controlled legs
$\frac{\text{Computation time}}{\text{Trajectory duration}}$	7.50 %	10.59 %	34.97 %



(a) Reference trajectory



(b) Adjusted trajectory without sensor feedback



(c) Adjusted trajectory with sensor feedback

Figure 3.14.: In this example, a reference trajectory is adjusted to balance a ball on a plate. As shown in (a), the ball falls off the plate when executing the reference trajectory. If the reference trajectory is adjusted without access to the current ball position (b), the ball stays on the plate but starts to move towards the edge. In contrast, a neural network having access to the ball position (c) manages to keep the ball close to its initial position on the plate. The figure is adapted from [82].

Computational effort Based on the selected decision frequency of 10 Hz, an analysis of the computational effort is provided in Table 3.6. To this end, various path tracking experiments are simulated and the proportion between the computation time and the duration of the generated trajectories is calculated. As a result, the time required for the simulations is considerably shorter than the duration of the simulated trajectories.

In conclusion, the proposed action mapping was successfully used to track reference paths. Suitable actions were generated based on neural networks trained via model-free RL. The selected reward function encouraged a fast and precise traversal of the reference paths. In addition to these objectives, experiments with further balancing requirements were conducted. In particular, a bipedal robot was trained to avoid falling over and an industrial robot was trained to balance a ball on a plate. A successful sim-to-real transfer of the ball balancing task showed that trajectories can be generated in real time.

3.5.3. Adjusting reference trajectories

In a next step, the proposed action mapping is evaluated in the context of reference trajectories. Compared to reference paths, reference trajectories additionally specify the desired timing of a movement. When provided with a reference trajectory, the learning goal is to generate a slightly adjusted trajectory that satisfies additional task constraints. For the following evaluation, the additional objective is to balance a ball on a plate. Figure 3.14 visualizes the execution of a reference trajectory and two adjusted trajectories. When executing the reference trajectory (a), the ball falls off the plate. The adjusted trajectory shown in (b) is generated by a neural network that receives no feedback about the move-

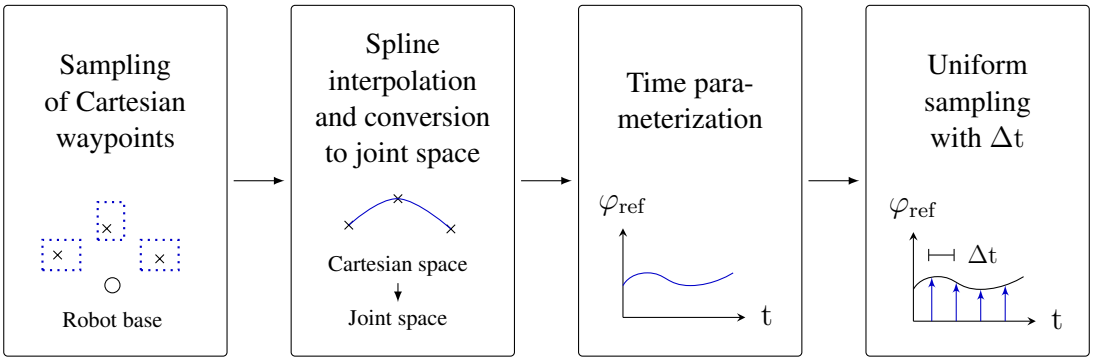


Figure 3.15.: The procedure to generate reference trajectories for the ball balancing task. The figure is adapted from [82].

ments of the ball. In this case, the ball does not fall to the ground but moves towards the edge of the plate. The network used to produce the trajectory shown in (c) has access to the current ball position. Due to this sensor feedback, the ball can be kept close to its initial position on the plate. This example demonstrates the importance of sensor feedback, which can only be incorporated when computing trajectories during motion execution.

Generation of reference trajectories The procedure to generate reference trajectories for the ball-on-plate task is illustrated in Figure 3.15. First, three random Cartesian waypoints are selected from predefined areas around the robot base. Next, a spline interpolation is performed to calculate a smooth Cartesian path connecting the waypoints. While the path indicates the desired Cartesian positions of the plate, the orientations are selected so that the top side of the plate is aligned horizontally. Using an inverse kinematics solver provided by PyBullet [30], the Cartesian positions and orientations are converted to joint space. Next, a time parameterization is computed using a TOPP algorithm proposed by Kunz and Stilman [89]. To ensure a certain degree of flexibility when adjusting the resulting reference trajectory, the velocity and acceleration limits for the time parameterization are set below the maximum values of the robot. Finally, position values φ_{ref} are sampled from the reference trajectory at uniform time intervals Δt. The time interval Δt corresponds to the duration between the decision steps when adjusting the reference trajectory. Thus, a reference position φ_{ref_t} can be provided for every time step t. The computed reference positions are either assigned to a training dataset, which is used for the training of neural networks, or to a test dataset, which is used for evaluation.

Markov decision process The Markov decision process $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ that is used to adjust reference trajectories is similar to the one defined for tracking reference paths. A state $s_t \in \mathcal{S}$ contains the kinematic state of each robot joint $(\varphi_{d_t}, \dot{\varphi}_{d_t}, \ddot{\varphi}_{d_t})$, the current ball position, and the initial ball position with respect to the plate. To provide information on the reference trajectory, s_t additionally includes $\varphi_{\text{ref}_{t+1}}$, the reference position at $t+1$. Using the proposed action mapping, an action $a_t \in \mathcal{A}$ consists of one scalar $\in$ [-1,1] per robot joint. Since the 7-DOF robot KUKA iiwa is used for the experiments, each action is a seven-dimensional vector. The reward function R is based on two main components R_{Ball} and R_{Ref}.

Table 3.7.: Training results when adjusting reference trajectories to balance a ball on a plate. If the ball deviates more than $6\,\mathrm{cm}$ from its initial position, the execution of a trajectory is aborted. The first column indicates the proportion of trajectories without such a balancing error. In the second column, the part of a trajectory traversed before a balancing error is specified. The third column shows the average distance between the ball and its initial position on the plate. In simulation, the evaluation is based on 10 000 trajectories from the test dataset. For the experiment with a real robot, 50 trajectories with 5 different initial ball positions were carried out. Adapted from [82].

Experiment	Trajectories without balancing error [%]	Trajectory part before error [%]	Average distance to initial ball position [cm]
Simulated KUKA iiwa			
• Reference trajectories	0.3	22.1	–
• Without sensor feedback			
- during evaluation	34.3	73.0	2.4
- during training and evaluation	61.7	91.7	1.6
• With sensor feedback	98.6	99.7	1.2
Real KUKA iiwa			
• With sensor feedback	82.0	96.1	1.7

While R_{Ball} encourages the robot to balance the ball, R_{Ref} makes sure that the position setpoint $\varphi_{\mathrm{d}_{t+1}}$ resulting from action a_t is close to the reference position $\varphi_{\mathrm{ref}_{t+1}}$. The definition of R_{Ball} is the same as that used for the path tracking task shown in Figure 3.10 (c). For the reference reward R_{Ref}, the deviation $\Delta\varphi_{t+1}$ between $\varphi_{\mathrm{ref}_{t+1}}$ and $\varphi_{\mathrm{d}_{t+1}}$ is computed as follows:

$$\Delta\varphi_{t+1} = \varphi_{\mathrm{ref}_{t+1}} - \varphi_{\mathrm{d}_{t+1}} \tag{3.19}$$

Next, the highest deviation of all joints is determined and denoted as $\Delta\varphi_{t+1,\,\mathrm{max}}$:

$$\Delta\varphi_{t+1,\,\mathrm{max}} = \max_{i\in\{1,\ldots,7\}} |\Delta\varphi_{t+1,\,i}| \tag{3.20}$$

The function for determining R_{Ref} has the same form as the function for R_{Ball}, depicted in Figure 3.10 (c). However, the distances d_{Ball}, d_{one}, and d_{zero} are substituted by the deviations $\Delta\varphi_{t+1,\,\mathrm{max}}$, $\Delta\varphi_{\mathrm{one}}$, and $\Delta\varphi_{\mathrm{zero}}$, respectively. During training, an episode is terminated if d_{Ball} exceeds d_{zero} or if $\Delta\varphi_{t+1,\,\mathrm{max}}$ exceeds $\Delta\varphi_{\mathrm{zero}}$.

Evaluation of the balancing performance Table 3.7 shows a quantitative analysis of the balancing performance for various scenarios. For all experiments, a decision frequency f_D of $20\,\mathrm{Hz}$ was used. The distances d_{one} and d_{zero} were set to $2\,\mathrm{cm}$ and $6\,\mathrm{cm}$, while the deviations $\Delta\varphi_{\mathrm{one}}$ and $\Delta\varphi_{\mathrm{zero}}$ were chosen to be $2°$ and $8°$. The first experiments were conducted in a simulation environment using 10 000 reference trajectories from the test dataset. When executing the reference trajectories without adjustments, the balancing of the ball failed in $99.7\,\%$ of all cases. On average, the ball distance d_{Ball} exceeded d_{zero} after executing the first $22\,\%$ of a reference trajectory.

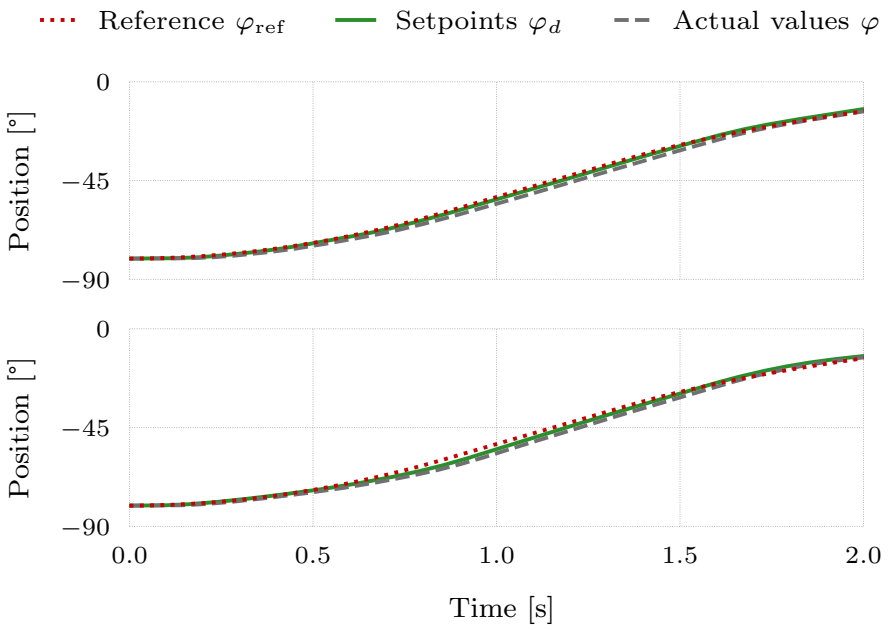


Figure 3.16.: The position of Joint 1 during the first two seconds of an exemplary trajectory executed in simulation and with a real KUKA iiwa robot. In both cases, the same neural network is used to generate the trajectory. It can be seen that the position setpoints φ_d lie close to the reference positions φ_{ref}. In addition, the time delay between the setpoints φ_d and the actual values φ is similar for the simulated robot and the real robot. Around $\mathrm{t} = 1.0\,\mathrm{s}$, the deviation between the reference trajectory and the generated trajectory is slightly higher when using the real robot. This discrepancy can be attributed to differences in the movement of the simulated ball and the real ball. The figure is adapted from [82].

In contrast, a neural network trained to adjust the reference trajectories was able to balance the ball in $98.6\,\%$ of all cases if the current ball position was provided as sensor feedback. Two additional experiments were conducted to evaluate the impact of having access to the current ball position. In the first experiment, the network used to generate the adjusted trajectories was trained with access to the ball position. However, during evaluation, the ball position was not updated. As a result, the rate of successful trajectories dropped from $98.6\,\%$ to $34.3\,\%$. Thus, it can be concluded that the current ball position is an important input signal for the neural network. The second experiment was conducted using a network that was trained without access to sensor feedback. In this case, the rate of successful trajectories increased to $61.7\,\%$, but still remained far below the rate achieved when using sensor feedback. To analyze the impact of a sim-to-real transfer, a network trained in simulation was used to control a real KUKA iiwa robot. Using the same experimental setup as shown in Figure 3.12, a total of 50 trajectories were executed. The current ball position was detected by a resistive touchpad placed on top of the plate. For the quantitative analysis, five different initial ball positions were selected. More precisely, the ball was placed either in the center of the plate or at one of the four yellow spots shown in Figure 3.18. Under these testing conditions, the ball was successfully balanced in $82.0\,\%$ of all cases. Therefore, it can be concluded that a network trained in simulation can also be used to control a real robot. However, compared to the results achieved in simulation, the sim-to-real transfer led to a certain decrease in performance.

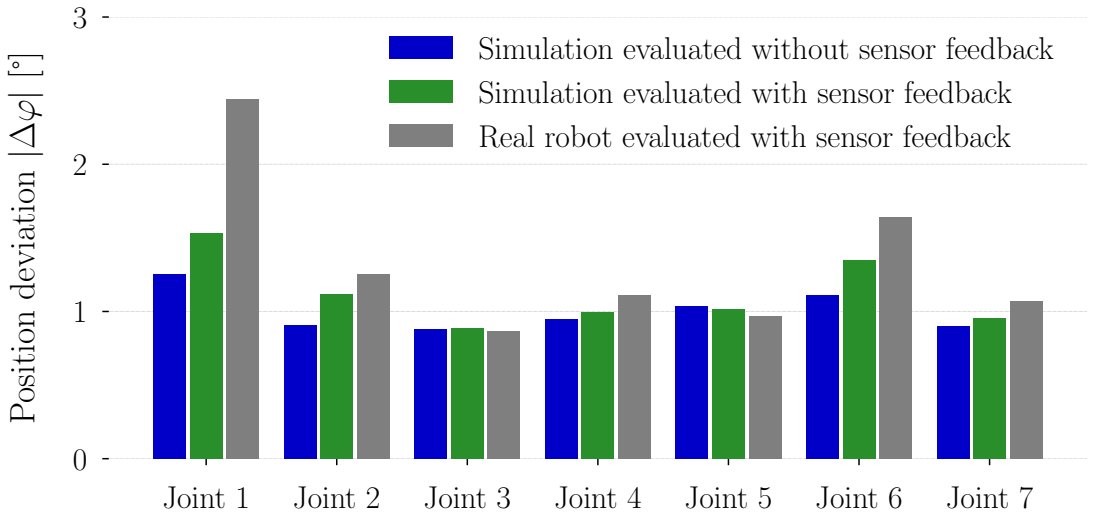
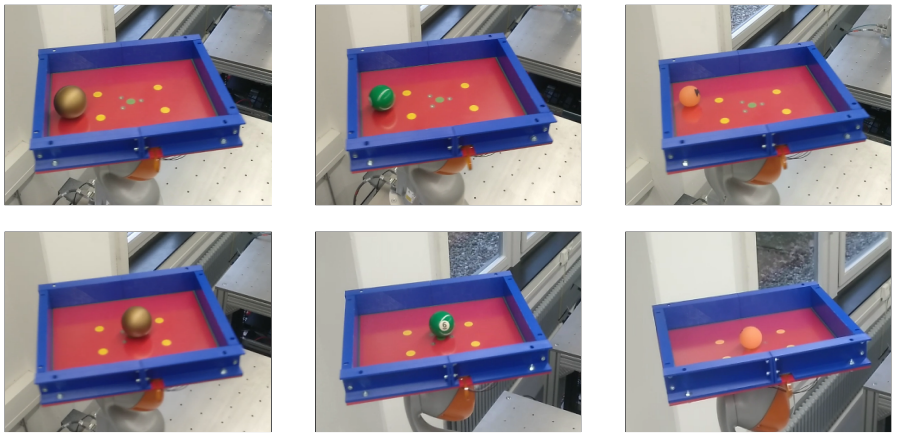


Figure 3.17.: The average absolute position deviation between a reference trajectory and the corresponding adjusted trajectory evaluated based on 50 episodes. All experiments were conducted using the same neural network. During training, the network had access to the current ball position. The figure is adapted from [82].

Comparison between simulated and real trajectories To further analyze the differences between the simulation and the real robot, Figure 3.16 shows the first two seconds of an adjusted trajectory for one exemplary joint. It can be seen that the time delay between the position setpoints φ_d and the actual values φ is similar for the simulation and the real robot. While in both scenarios, the position setpoints φ_d closely align with the reference positions φ_{ref}, a slightly higher deviation can be observed when using the real robot. A quantitative analysis of the deviation between the position setpoints φ_d and the reference positions φ_{ref} is shown in Figure 3.17. During the training phase, the reference reward R_{Ref} starts to decrease if the position deviation $\Delta\varphi_{t+1,\,\max}$ exceeds $\Delta\varphi_{\text{one}} = 2°$. As a result, the average position deviation in simulation remains below $2°$ for all joints. However, when using a real robot, the deviation increases for most of the joints. This observation can be attributed to slight differences in the movement of the real ball compared to the simulated ball.

Robustness of the balancing policy The real-world experiments analyzed so far were carried out using a glass marble depicted in Figure 3.12. To investigate the robustness of the balancing policy, further real-world experiments were conducted. As shown in Figure 3.18, a neural network with access to the current ball position was able to balance different balls varying in terms of their size, mass, and material. During the training phase of the neural network, the learning of a robust policy was encouraged by selecting several parameters of the simulated ball in a randomized manner.



(a) Reference trajectory

(b) Adjusted trajectory

Figure 3.18.: The position of three different balls one second after starting the execution of a reference trajectory (a) and the corresponding adjusted trajectory (b). Although the balls vary in terms of their size, mass, and material, a neural network having access to the current ball position is able to keep all balls close to their initial position at the center of the plate.

In summary, the evaluation showed that neural networks can be effectively trained to adjust reference trajectories. A quantitative analysis conducted for a ball-on-plate task demonstrated the importance of sensory feedback during motion execution. By performing experiments with a real KUKA robot, the impact of a sim-to-real transfer was investigated. It was shown that a neural network trained in simulation could also be used to balance a ball with a real robot, albeit with a slight decrease in performance. Additional real-world experiments highlighted the robustness of the balancing policy across variations in the size, mass, and material of the ball selected for evaluation.

3.6. Summary

This chapter introduced a method for learning robot trajectories under consideration of kinematic joint constraints. More specifically, the actions of a reinforcement learning agent were translated into trajectories that adhered to predefined position, velocity, acceleration, and jerk limits. To this end, an upper and a lower trajectory were introduced. Both trajectories were assumed to comply with the kinematic constraints. As a result, an action of an RL agent could be used to specify an intermediate trajectory that also adhered to the kinematic constraints. The jerk between decision steps was assumed to be constant. Therefore, the problem of finding an upper and a lower trajectory was simplified to determining a range of feasible accelerations for the upcoming discrete time step. By employing concepts related to the generation of time-optimal trajectories, analytical equations to compute the desired range of accelerations could be derived.

The action mapping presented was evaluated based on three different robot tasks learned using model-free RL. First, an industrial robot was trained to generate movements that maximize the average joint velocity over time. As desired, the generated trajectories fully utilized the kinematic capabilities of the robot joints without exceeding them.

The evaluation also highlighted the advantages of the proposed method over penalties, which had a negative impact on the learning performance and did not strictly prevent violations of the kinematic limits.

Subsequently, the evaluation was extended to tasks requiring a temporal coordination between the robot joints. In particular, the proposed method was successfully employed to track reference paths with an industrial robot and two humanoid robots. As the trajectories were generated online, it was possible to consider additional objectives that required access to sensory feedback. For example, a bipedal robot could be trained to track reference paths without falling over. In this case, a total of 30 robot joints were controlled simultaneously, demonstrating that the proposed method can be applied to complex robotic systems. Similarly, an industrial robot was trained to additionally balance a ball on a plate. For this task, a neural network trained in simulation was successfully transferred to a real KUKA robot, showing that trajectories can be generated in real time.

In addition to tracking reference paths, the action mapping was also used to adjust reference trajectories. The evaluation for a ball balancing task further emphasized the importance of sensory feedback. To investigate the impact of a sim-to-real transfer, a quantitative analysis was conducted. While the simulation led to a success rate of $98.6\,\%$, the real robot managed to balance the ball in $82.0\,\%$ of all cases. As a result, the impact of the sim-to-real transfer on the task performance was limited.

4. Avoiding safety violations based on background simulations

The previous chapter introduced an approach for learning robot trajectories subject to kinematic joint constraints. In this chapter, the approach is extended by integrating further safety constraints into the learning process. While the main focus is on collision avoidance, the presented concept can also be applied to other safety constraints, such as torque limits. As in the previous chapter, the learning of robot trajectories is based on a *Markov decision process* (MDP). Actions for the MDP are generated by a neural network. The action mapping presented in section 3.3 is used to ensure compliance with the kinematic joint constraints. In order to consider further safety constraints, a background simulation and a backup policy are introduced. While the background simulation is performed to check if an action from the neural network can be safely executed, the backup policy provides an alternative action if the safety check fails.

In general, the backup policy can either be specified manually or learned from data. First, a scenario is analyzed in which the backup policy is predefined. More precisely, the backup policy involves calculating a braking trajectory that leads all robot joints to a complete stop. This particular scenario was initially described in [77]. Based on experiments with a humanoid robot and up to three industrial robots, it is shown that collisions can be completely avoided if the robots are firmly connected to the ground and surrounded by stationary obstacles only.

Next, the more general case of learning a backup policy is examined. In particular, an industrial robot is trained to actively avoid moving obstacles via model-free RL. The method was initially presented in [83] and is evaluated using three different environments, including a human-robot scenario, in which the human moves stochastically. As a result, the proposed method effectively reduces the occurrence of collisions. While the computational effort of background simulations is relatively high, a significantly faster data-based extension is introduced in chapter 5.

4.1. Problem description

Based on the framework of an MDP $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, robot trajectories are learned using model-free RL. As in chapter 3, all trajectories must comply with the kinematic joint constraints given by the equations (3.1) - (3.4). During and after the training process of a task policy π_T, additional safety constraints should be taken into account by utilizing a backup policy π_B. The backup policy π_B should be deterministic and task-independent. In particular, it should not depend on the current task policy π_T.

At each decision step t, the task policy is invoked to determine an action a_t^T. However, action a_t^T is only executed if it passes a safety check. The safety check is based on a

background simulation conducted in a physics simulator. If the safety check fails, the backup policy is used to provide an alternative action a_t^B:

$$a_t = \begin{cases} a_t^T & \text{if } a_t^T \text{ passes a safety check} \\ a_t^B & \text{otherwise} \end{cases} \tag{4.1}$$

Executing a_t^B instead of a_t^T does not lead to safety violations provided that certain conditions are met. Further details on the conditions are given in section 4.4. The backup policy π_B can be specified manually or learned from data. Section 4.5 introduces the concept of utilizing braking trajectories as a backup policy. Braking trajectories are particularly suited for environments without moving obstacles and for robots that cannot lose their balance. As a safety constraint, self-collisions and collisions with static obstacles should be avoided. For this purpose, the minimum distance between two robot links d_{self} and the minimum distance between a robot link and a static obstacle d_{static} must exceed predefined safety distances:

$$d_{\text{self}} > d_{\text{safety}_{\text{self}}} \tag{4.2}$$

$$d_{\text{static}} > d_{\text{safety}_{\text{static}}} \tag{4.3}$$

In addition, the torque τ of each robot joint can be limited:

$$\tau_{\min} \leq \tau \leq \tau_{\max} \tag{4.4}$$

Section 4.6 introduces the more general case of learning a backup policy via model-free RL. In the presented example, the backup policy is trained to avoid self-collisions, collisions with static obstacles, and collisions with moving obstacles. To this end, the minimum distance between a robot link and a moving obstacle is denoted as d_{moving} and the desired safety distance is denoted as $d_{\text{safety}_{\text{moving}}}$. Building on these definitions, the following additional safety constraint is specified:

$$d_{\text{moving}} > d_{\text{safety}_{\text{moving}}} \tag{4.5}$$

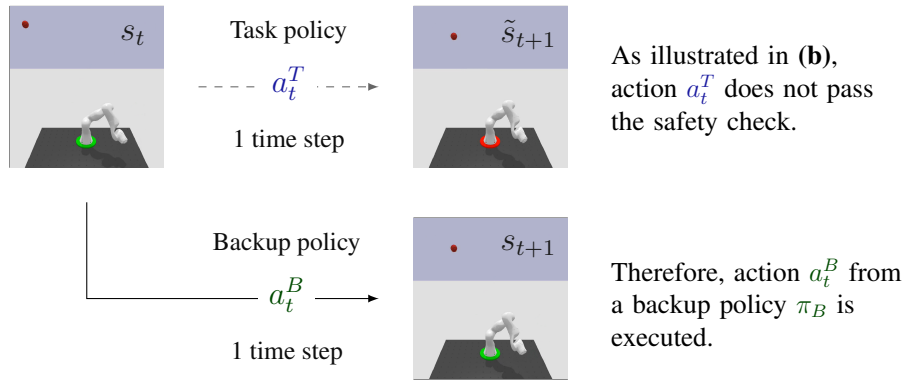
A discussion on incorporating further safety constraints can be found in chapter 6.

4.2. Relation to previous studies

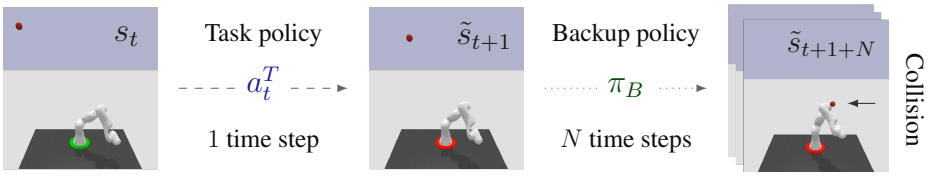
As outlined in section 2.2.2.2, the general concept of using backup policies to prevent safety violations has been employed in various application domains. For instance, Hans et al. [54] used a backup policy to avoid undesirable side effects when learning to control a gas turbine. Similarly, Garcia and Fernández [43] employed backup policies to learn tasks like pole balancing and car parking. In the field of robotics, Yang et al. [186] utilized a backup policy to stabilize a quadruped robot, while Thananjeyan et al. [167] made use of backup policies to avoid safety violations when learning two-dimensional navigation and manipulation tasks.

The presented approach differs from previous studies in three key aspects:

- First, backup policies are used to learn safely executable trajectories for industrial robots and humanoid robots in joint space. Especially in multi-robot scenarios, this involves controlling a large number of joints simultaneously.



(a) In order to avoid safety violations, an action a_t^B from a backup policy π_B is executed if an action a_t^T from a task policy π_T does not pass a safety check.



(b) The safety check is based on a background simulation conducted in a physics simulator. In this example, the safety check fails as the ball collides with the robot.

Figure 4.1.: The concept of avoiding safety violations based on a backup policy and a background simulation shown for one exemplary time step t. In this setting, the robot must avoid a collision with a red ball that is thrown in its direction. The figure is adapted from [83].

- Second, the proposed approach ensures that the kinematic constraints of the robot joints are respected, regardless of whether the task policy or the backup policy is executed. For this reason, it is possible to learn fast robot movements without exceeding the kinematic limits of the robot joints.
- Third, a background simulation is performed to determine whether the task policy or the backup policy is executed. As a result, conditions can be derived under which safety violations can be entirely prevented.

In this chapter, it is shown that braking trajectories can be used to learn collision-free trajectories in environments without moving obstacles. This particular result was inspired by general safety criteria for robot motions outlined in [40] and the idea of utilizing braking trajectories as a fallback strategy proposed in [141].

4.3. Basic principle

Using model-free RL, trajectories for a robot task are learned based on the framework of an MDP $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$. Specifically, a task policy π_T is trained to generate actions that maximize the expected sum of future rewards. The task policy is represented by a neural network. At the beginning of a training process, the neural network typically generates random actions. In order to avoid potential safety violations when following the task

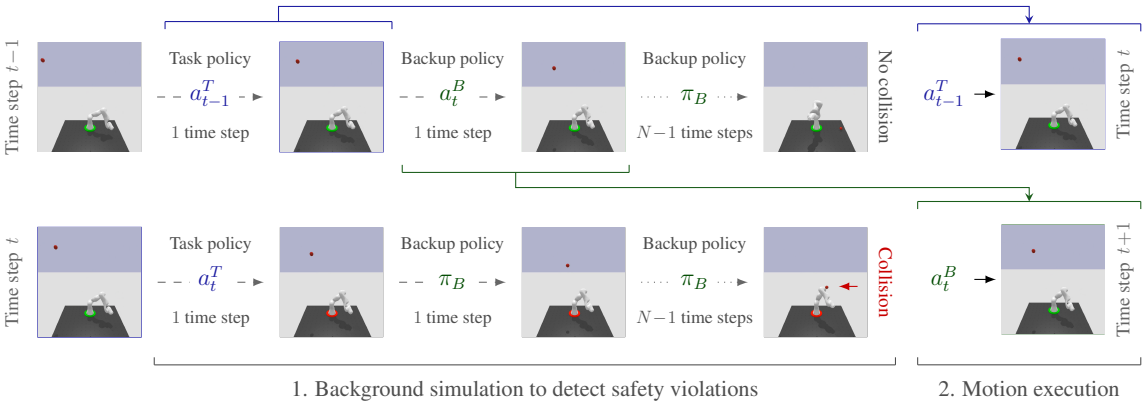


Figure 4.2.: In this example, the safety check for action a_t^T at time step t fails as a collision is detected during the background simulation. However, the action a_t^B from the backup policy can be safely executed as it was checked for collisions during the background simulation conducted at time step $t-1$. The figure is adapted from [83].

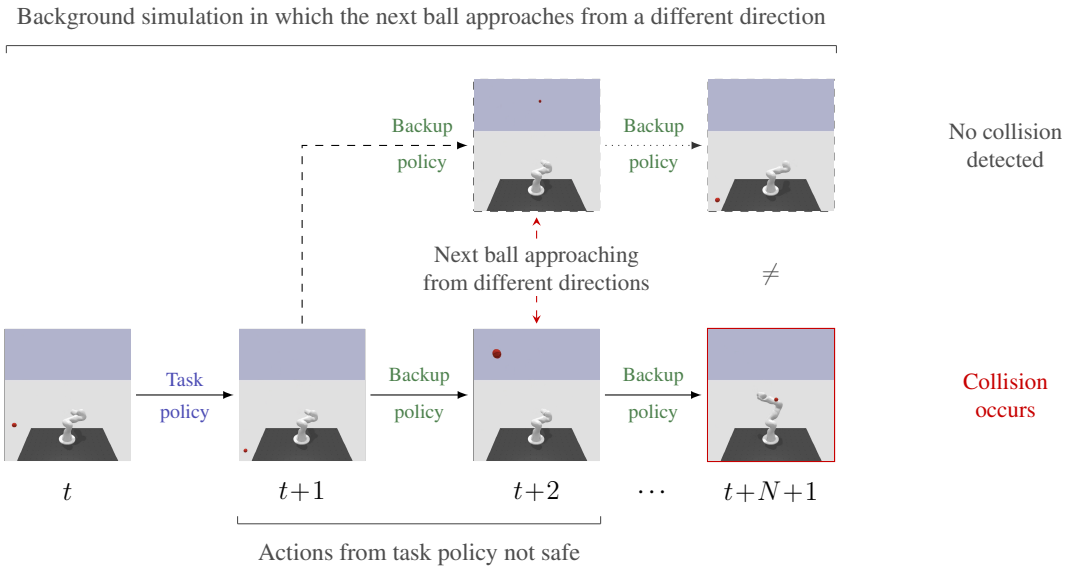
policy, a backup policy π_B and the concept of background simulations are introduced. Figure 4.1 illustrates the basic principle of the approach for one exemplary time step t. Based on the current state s_t, the task policy π_T generates an action a_t^T. As shown in Figure 4.1 (a), action a_t^T is only executed if it passes a safety check based on a background simulation. Otherwise, it is replaced by an action a_t^B from the backup policy π_B. The implementation of the background simulation is visualized in Figure 4.1 (b). Starting from state s_t, the execution of action a_t^T is simulated. Subsequently, up to N further time steps are simulated using actions generated by the backup policy. If no safety violation is detected, the safety check is considered as passed. In this particular example, the robot collides with the red ball during the background simulation. Thus, the safety check is considered failed and the backup action a_t^B is executed.

In cases like this, it is assumed that the backup action can be executed safely. This assumption is based on the idea that the backup action has already been checked for safety violations during a background simulation conducted at an earlier time step. Figure 4.2 illustrates this idea for the previously discussed example. In this example, a collision is detected during the background simulation at time step t, so that the backup action a_t^B is executed. However, the figure additionally shows that the execution of a_t^B was already simulated at time step $t-1$. Since no safety violation was detected during this background simulation, executing a_t^B is considered safe. In the following section, conditions are derived under which the assumption that the backup action can be safely executed holds true.

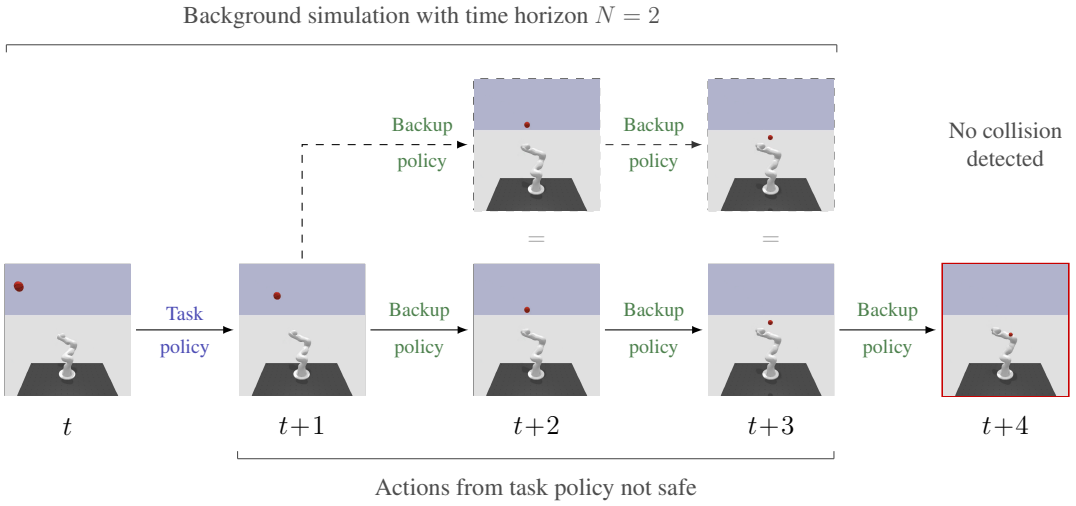
4.4. Safety conditions

The presented strategy for avoiding safety violations is based on two main assumptions:

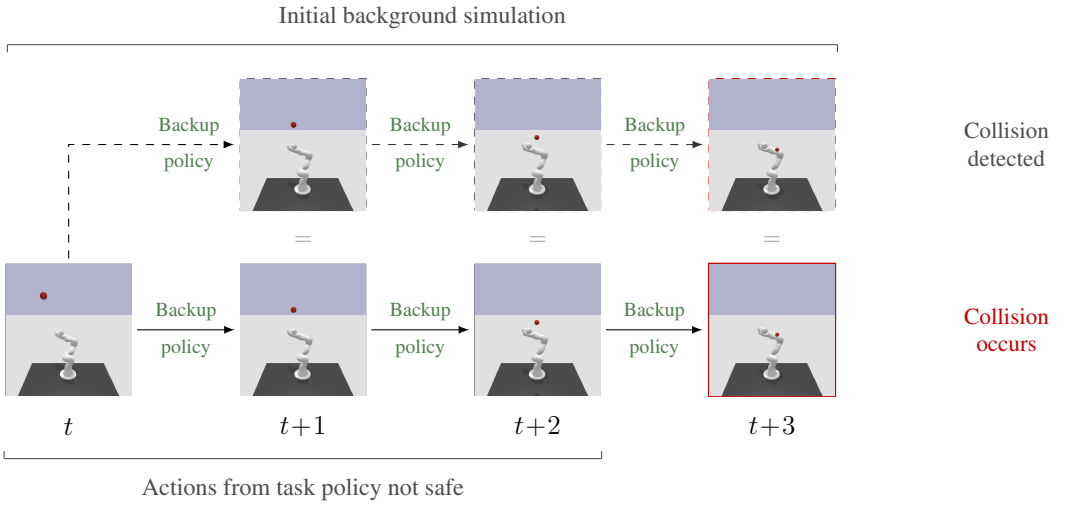
- First, potentially unsafe actions of the task policy are detected. Specifically, an action of the task policy is considered unsafe if a subsequent execution of the backup policy leads to a safety violation.
- Second, if an action of the task policy is considered unsafe, an alternative action from the backup policy can be executed safely.



(a) The background simulation assumes a different temporal development of the environment.



(b) The time horizon of the background simulation is too short.



(c) The initial state of the environment is not safe.

Figure 4.3.: Failure causes when using background simulations to avoid safety violations. The figure is adapted from [83].

In this section, conditions are derived under which the above assumptions can be met. The conditions are divided into three categories, which are described in detail below. For each of these categories, Figure 4.3 shows how non-compliance can result in a safety violation.

Predictability of the environment As an important prerequisite, the background simulation must accurately reflect the actual temporal development of the environment. This requirement encompasses two main aspects:

- First, the physics simulation must be accurate. This involves accurate models of the robot and the environment as well as a proper consideration of all relevant physical effects. In practice, the requirement can be relaxed by introducing safety margins. For example, the background simulation may already indicate a collision if the distance between two objects falls below a predefined safety distance. In this way, inaccuracies in the physics simulation can be tolerated to a certain extent.
- Second, future developments must be predictable. In particular, all parts of the environment that are relevant for adhering to the safety constraints must exhibit deterministic behavior.

Figure 4.3 shows an environment in which a robot is supposed to avoid collisions with a ball thrown in its direction. Every time the ball misses the robot, a new ball is thrown from a random direction. Since the direction of the new ball is not known in advance, the future development of the environment cannot be exactly predicted. If a random ball direction is selected during the background simulation, an action of the task policy might be incorrectly classified as safe, which might result in a collision. This particular failure cause is illustrated in Figure 4.3 (a). One approach to deal with the non-deterministic nature of the environment is to conduct a large number of background simulations assuming different ball directions. This enables a probabilistic assessment of the risk associated with a particular action of the task policy. However, as the computational effort required for the background simulations is high, a more efficient data-based approach for stochastic environments is presented in chapter 5.

Time horizon of the background simulation A background simulation conducted at time step t involves the transition from t to $t+1$, followed by N additional state transitions where the backup policy is used to determine actions. The time horizon of the background simulation is defined as the time span $N \cdot \Delta\mathrm{t}$, where $\Delta\mathrm{t}$ is the time interval between two subsequent states. For the sake of simplicity, the constant time interval $\Delta\mathrm{t}$ can be omitted when referring to the time horizon. On the one hand, the time horizon should be long enough to detect potential safety violations. On the other hand, the computational effort of the background simulation increases with the selected time horizon.

Fraichard [40] introduced general safety criteria that must be met to ensure motion safety. Specifically, the time horizon under consideration should either be infinite or at least encompass the duration needed to reach a safe goal state. While infinite time horizons cannot be realized when using a physics simulator, the backup policy can guide the robot to a safe goal state under certain conditions. Assuming that no further safety violations can occur once the robot is at a standstill, section 4.5 introduces the use of braking trajectories as a backup policy. In this particular case, the time horizon corresponds to the time required to bring the robot to a complete stop.

Depending on the environment, however, safety violations might occur even if the robot joints do not move. One example is the environment shown in Figure 4.3, where balls are thrown in the direction of the robot. If the backup policy does not lead to a safe goal state, a constant time horizon N can be selected. Assuming that no safety violation is detected during a background simulation, it is considered safe to use the backup policy for the following N decision steps. When using the backup policy more than N times in a row, safety violations might occur as the time horizon of the background simulation is exceeded. Therefore, a background simulation without any safety violation must take place at least every $N+1$ time steps. This corresponds to the condition that an action from the task network must be selected at least every $N+1$ time steps. Figure 4.3 (b) illustrates a scenario in which the time horizon is set to $N = 2$. Due to the short time horizon, potential collisions might be detected too late to avoid them. In the specific example, a collision occurs after using the backup policy more than two times in a row.

Initialization of the environment As an additional condition, the environment must be initialized in a safe state. A state is considered safe if no safety violation is detected during an initial background simulation. In contrast to the regular background simulation, the initial background simulation only covers the execution of the backup policy for the selected time horizon. Figure 4.3 (c) shows an example, in which a collision occurs as the environment is initialized in an unsafe state. This type of failure can be prevented by initializing the environment in a state where an initial background simulation does not detect any safety violation.

4.5. Using braking trajectories as a backup policy

In this section, the backup policy π_B is supposed to provide actions that slow down the robot joints and finally bring the robot to a complete stop. The use of braking trajectories is motivated by the idea that a standstill of the robot can be regarded as a safe goal state under certain conditions. In this thesis, a stationary state of the robot joints is assumed to be safe if there are no moving obstacles in the environment and if a stable base prevents the robot from falling over.

With these assumptions in place, the safety conditions outlined in section 4.4 are revisited. As one of the requirements, the robot must be initialized in a safe state. Since a robot is usually started from standstill, it is considered safe to select any initial robot position that does not immediately lead to a safety violation. Specifying the time horizon of the background simulation is also straightforward, as it corresponds to the time required to slow down all robot joints. Another important aspect is the predictability of the environment. In the scenario under consideration, the calculation of braking trajectories solely depends on the kinematic state of the robot joints. As a result, it is relatively easy to determine the robot trajectory to be performed during the background simulation. In addition, the detection of potential safety violations is simplified by the absence of moving obstacles.

This thesis investigates the use of braking trajectories to prevent self-collisions and collisions with stationary obstacles. In addition, torque limits can be taken into account. Figure 4.4 shows a scenario in which self-collisions need to be avoided when performing

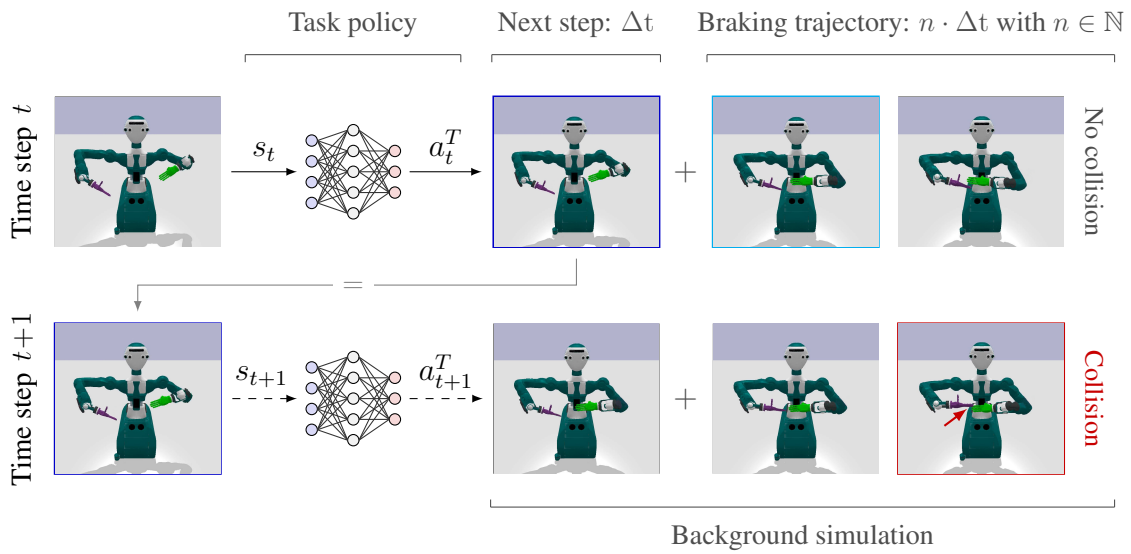


Figure 4.4.: Two exemplary time steps with the humanoid robot ARMAR-6 when using braking trajectories to avoid self-collisions. At time step t, action a_t^T is executed because no collision is detected during a background simulation. In contrast, a collision is detected at time step $t+1$. The resulting braking trajectory leads to the state marked by a light blue frame, which is regarded as safe. The figure is adapted from [77].

movements with the humanoid robot ARMAR-6. As visualized, the task policy is represented by a neural network that is trained via reinforcement learning. At time step t, no collision is detected during a background simulation. Consequently, an action provided by the task policy is executed. In contrast, the hands of the robot collide during the background simulation conducted at time step $t+1$. Therefore, the backup policy is used to slow down the robot joints. Using the backup policy is considered safe, as the resulting braking trajectory was checked for collisions during the background simulation at time step t. In the following, it is explained how the braking trajectories are computed and how the safety checks are conducted. Subsequently, the proposed approach is evaluated by learning a variety of reaching tasks with multiple robots.

4.5.1. Computation of braking trajectories

At each time step t, the backup policy π_B can be invoked to generate an action a_t^B that slows down the robot joints. The resulting braking trajectory must bring the robot to a full stop after a number of time steps and must not violate the kinematic limits of the robot joints. In theory, these conditions can be met by different braking trajectories. However, since the computational effort of the background simulation depends on the duration of the braking process, a time-optimized braking trajectory is preferred. To this end, additional acceleration and jerk limits for the braking process are introduced:

$$\mathrm{a}_{\mathrm{min,\,brake}} \leq \ddot{\varphi}_d \leq \mathrm{a}_{\mathrm{max,\,brake}} \tag{4.6}$$

$$\mathrm{j}_{\mathrm{min,\,brake}} \leq \dddot{\varphi}_d \leq \mathrm{j}_{\mathrm{max,\,brake}} \tag{4.7}$$

These limits must be less than or equal to the kinematic constraints defined in equation (3.3) and (3.4). The desired braking trajectories should reach the kinematic target state ($\mathrm{v_T} = 0$, $\mathrm{a_T} = 0$) at a discrete time step. To this end, a suitable braking acceleration

$\mathrm{a}_{t+1_{\mathrm{brake}}}$ is calculated at each time step t. In section 3.4, the computation of $\mathrm{a}_{t+1_{\mathrm{max,\,v}}}$ was explained. When setting a_{t+1} to $\mathrm{a}_{t+1_{\mathrm{max,\,v}}}$, the velocity limit $\mathrm{v}_{\mathrm{max}}$ is eventually reached at an acceleration of zero. Assuming that the initial velocity v_t is negative, the braking acceleration $\mathrm{a}_{t+1_{\mathrm{brake}}}$ can be computed in a similar manner. Specifically, the velocity limit $\mathrm{v}_{\mathrm{max}}$ is set to zero, while the considered acceleration and jerk limits are taken from equation (4.6) and (4.7). If the initial velocity is positive, $\mathrm{a}_{t+1_{\mathrm{brake}}}$ is computed analogously by assuming the velocity limit $\mathrm{v}_{\mathrm{min}}$ to be zero. To ensure that the resulting braking trajectory does not violate the position or velocity limits defined in equation (3.1) and (3.2), $\mathrm{a}_{t+1_{\mathrm{brake}}}$ is clipped if it exceeds the range of valid accelerations $[\mathrm{a}_{t+1_{\mathrm{min}}}, \mathrm{a}_{t+1_{\mathrm{max}}}]$. Based on $\mathrm{a}_{t+1_{\mathrm{brake}}}$, the backup action a_t^B can be calculated as follows:

$$a_t^B = 2 \cdot \frac{\mathrm{a}_{t+1_{\mathrm{brake}}} - \mathrm{a}_{t+1_{\mathrm{min}}}}{\mathrm{a}_{t+1_{\mathrm{max}}} - \mathrm{a}_{t+1_{\mathrm{min}}}} - 1 \tag{4.8}$$

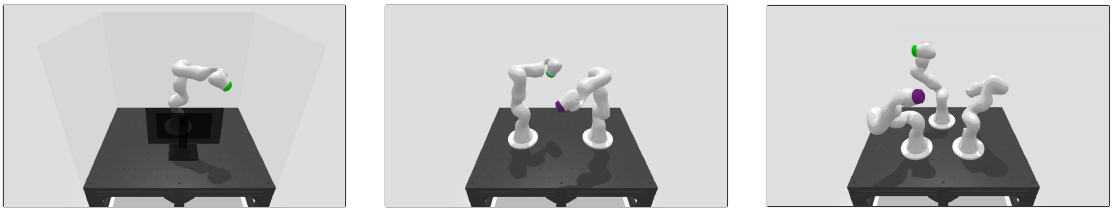
Knowing a_t^B, a braking trajectory from t to $t+1$ can be computed. By repeating the procedure described above, the braking trajectory is extended until all robot joints are brought to a complete stop.

4.5.2. Detection of safety violations

To detect potential safety violations, a background simulation is conducted. In this thesis, the background simulation is performed by the physics simulator PyBullet [30]. The simulation is carried out in discrete time steps using a predefined frequency f_S. Typically, the simulation frequency f_S is significantly higher than the decision frequency of the Markov decision process f_D. After each simulation step, the physics simulator provides access to a wide range of data that can be used to check whether safety constraints are violated. For example, the torque applied to each robot joint can be read out. If one of the torque values exceeds the permitted operating range, the background simulation is aborted and a torque limit violation is reported. Similarly, it is possible to detect whether a collision occurred during the last simulation step. In addition, PyBullet offers a function to determine the minimum distance between two objects. This way, it is possible to report a safety violation if the minimum distance between certain objects is smaller than a specified safety distance. To this end, relevant object pairs are defined in advance. More precisely, a link-link pair is defined to avoid collisions between two robot links. Similarly, an obstacle-link pair is specified to ensure a minimum distance between an obstacle and a robot link. To reduce the computational effort, the frequency of the distance calculations f_C can be set lower than the simulation frequency f_S. In addition, the minimum distances can be checked with respect to the position setpoints rather than the actual values. While both measures lead to a certain degree of inaccuracy, collisions are still avoided if the safety distance is specified appropriately.

4.5.3. Evaluation

The use of braking trajectories as a backup policy is evaluated by learning reaching tasks in four different environments. This includes an environment with the humanoid robot ARMAR-6 depicted in Figure 4.4 and three environments with the industrial robot KUKA iiwa shown in Figure 4.5. Further characteristics for each environment are listed in Table 4.1. The environment with a single industrial robot shown in Figure 4.5 (a) is used to



(a) One industrial robot **(b)** Two industrial robots **(c)** Three industrial robots

Figure 4.5.: Three environments with the industrial robot KUKA iiwa used to evaluate the avoidance of safety violations by simulating braking trajectories.

Table 4.1.: Properties of the environments used for evaluation. Adapted from [77].

Environment	DOF	Obstacles	Obstacle-link pairs	Link-link pairs
• One industrial robot	7	6	36	0
• Two industrial robots	14	1	12	72
• Three industrial robots	21	1	18	216
• Humanoid ARMAR-6	17	1	2	151

evaluate the impact of static obstacles. Specifically, the robot has to avoid collisions with a table, a monitor, and four virtual walls that serve as workspace barriers. In the environments with two and three industrial robots shown in Figure 4.5 (b) and (c), the focus is on avoiding collisions between the robots. The only static obstacle that needs to be considered is the table. Since the robots are controlled as a single unit, collisions between the robots can be treated as self-collisions. Similarly, self-collisions are the primary focus of the environment featuring the humanoid robot ARMAR-6, shown in Figure 4.4. In addition, potential collisions between the robot hands and the floor are taken into account. For each environment, the number of obstacle-link pairs and link-link pairs defined to avoid collisions is given in Table 4.1. In order to save computing time, only object pairs that can collide are taken into consideration.

Task description As part of the evaluation, several reaching tasks are learned using model-free RL. Specifically, the robots are trained to reach as many randomly positioned target points as possible within a certain period of time. In the environments with industrial robots, the target points must be touched by the last robot link highlighted in Figure 4.5. When using ARMAR-6, the tips of the robot hands fulfill this purpose. The evaluation includes the following three options to assign target points to the robot arms:

- **Single target point:** At any point in time, there is a single target point. Each of the robot arms can be used to reach the target point.
- **Alternating target points:** At any point in time, there is a single target point that is assigned to one specific robot arm. The assignment of the target points to the robot arms follows a fixed order.
- **Simultaneous target points:** At any point in time, there is one target point per robot arm. Each of the target points is assigned to one specific robot arm.

Table 4.2.: Performance metrics of random agents when avoiding collisions by performing background simulations. The safety distances $d_{\text{safety}_{\text{self}}}$ and $d_{\text{safety}_{\text{static}}}$ were set to $1.0\,\text{cm}$. To obtain the results, 900 episodes with a duration of 8 seconds were simulated. Adapted from [77].

Environment	Target points per second	Closest distance	Action adjustment rate
One industrial robot			
• single target point	0.008	0.93 cm	12.9 %
Two industrial robots			
• alternating target points	0.005	0.93 cm	8.1 %
• simultaneous target points	0.011	0.91 cm	8.1 %
Three industrial robots			
• alternating target points	0.005	0.91 cm	18.6 %
• simultaneous target points	0.013	0.82 cm	18.4 %
ARMAR-6			
• single target point	0.008	0.60 cm	5.6 %
• alternating target points	0.003	0.46 cm	5.5 %
• simultaneous target points	0.006	0.79 cm	5.4 %

State space In order to learn the reaching tasks via model-free RL, a Markov decision process $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ is defined. A state $s_t \in \mathcal{S}$ includes the current kinematic state of each robot joint (φ_{d_t}, $\dot{\varphi}_{d_t}$, $\ddot{\varphi}_{d_t}$). In addition, information about the target points is provided. Specifically, the state contains the Cartesian position of each target point, as well as the positional deviation between the target points and the robot arms to which the target points are assigned. In environments without moving obstacles, the state does not include additional information about the obstacles. Instead, the presence of static obstacles is learned through interaction with the environment.

Action space Analogous to the other evaluation tasks described so far, trajectories are generated using the action mapping presented in chapter 3. Consequently, an action $a_t \in \mathcal{A}$ consists of one scalar ∈ [-1,1] per robot joint. The number of controlled robot joints per environment is specified in Table 4.1.

Reward function The reward function R is a sum of target point rewards R_{T_k} computed for each active target point k, with K being the total number of active target points:

$$R = \sum_{k=1}^{K} R_{T_k} \tag{4.9}$$

At each time step t, the target point rewards R_{T_k} are computed as follows:

$$R_{T_k} = \frac{d_{k_t} - d_{k_{t+1}}}{d_{k_{\text{init}}}}, \tag{4.10}$$

where d_{k_t} and $d_{k_{t+1}}$ are the distances between a target point k and its associated robot arm at time step t and t+1, respectively. If a target point is assigned to multiple robot arms, the distances are calculated with respect to the robot arm that is the closest to the target point. The distance $d_{k_{\text{init}}}$ refers to the initial distance at the time the target point was selected.

Table 4.3.: Performance metrics of agents trained to reach target points using RL. Collisions were avoided by performing background simulations. The safety distances $d_{\text{safety}_{\text{self}}}$ and $d_{\text{safety}_{\text{static}}}$ were set to $1.0\,\text{cm}$. To obtain the results, 900 episodes with a duration of 8 seconds were simulated. Adapted from [77].

Environment	Target points per second				Closest distance	Action adjustment rate
	All arms	Arm 1	Arm 2	Arm 3		
One industrial robot						
• single target point	1.11	1.11	–	–	0.88 cm	1.6 %
Two industrial robots						
• alternating target points	1.05	0.52	0.53	–	0.65 cm	2.3 %
• simultaneous target points	1.31	0.26	1.05	–	0.62 cm	3.1 %
Three industrial robots						
• alternating target points	0.63	0.21	0.21	0.21	0.71 cm	5.1 %
• simultaneous target points	0.72	0.56	0.03	0.13	0.72 cm	17.2 %
ARMAR-6						
• single target point	1.63	0.79	0.84	–	0.95 cm	1.7 %
• alternating target points	1.55	0.78	0.77	–	0.62 cm	1.7 %
• simultaneous target points	2.20	0.98	1.22	–	0.21 cm	7.0 %

Initialization When initializing the environment, the robot is placed at a random collision-free position. The initial velocity and acceleration of the robot joints is set to zero. To specify the position of a target point, a random collision-free robot position is determined. Subsequently, the target point is positioned in a way that the robot at the randomly selected position would touch it. This procedure ensures that the target point is located within the operating range of the robot. Once a target point is reached, a new one is selected.

Collision avoidance Table 4.2 shows the performance of random agents when using braking trajectories to avoid collisions. For the experiments, a decision frequency f_D of $10\,\text{Hz}$ is selected. The safety distances $d_{\text{safety}_{\text{self}}}$ and $d_{\text{safety}_{\text{static}}}$ are set to $1.0\,\text{cm}$. During the background simulation, collisions are checked at a rate of $100\,\text{Hz}$. Whether a target point is reached and whether a collision occurs is determined based on the generated trajectory setpoints. Table 4.2 shows that random agents hardly ever reach a target point. The closest distance to a collision, however, is always greater than zero. Consequently, self-collisions and collisions with static obstacles are completely prevented. It can also be seen that the closest distances are slightly smaller than the specified safety distances. This is due to the fact that the closest distances are determined with respect to the actual positions rather than the position setpoints. Moreover, the closest distance is computed at a frequency of $240\,\text{Hz}$, while the collision checks during the background simulation are conducted at a rate of $100\,\text{Hz}$. The action adjustment rate in the third column indicates the percentage of actions that are replaced by actions from the backup policy. The rate depends on the environment and the current task policy and can be regarded as an indicator of the degree of difficulty in complying with the safety constraints. Since the actions of the backup policy do not directly contribute to the completion of the desired task, a low action adjustment rate is preferred to limit the potential impact on the task performance.

Table 4.4.: Ablation studies conducted to investigate the impact of collision avoidance based on simulated braking trajectories. The results were obtained by simulating 900 episodes, each with a duration of 8 seconds. Adapted from [77].

Configuration	Target points	Episodes with collisions	Action adjustment rate
ARMAR-6 with simultaneous target points			
• Random agent without considering collisions	$0.01\,\mathrm{s}^{-1}$	$75.0\,\%$	–
• Trained and evaluated without considering collisions, using actual values	$2.32\,\mathrm{s}^{-1}$	$64.8\,\%$	–
• Trained without considering collisions, evaluated considering collisions, using actual values	$1.86\,\mathrm{s}^{-1}$	$0.0\,\%$	$23.0\,\%$
• Trained and evaluated considering collisions, using actual values	$2.01\,\mathrm{s}^{-1}$	$0.0\,\%$	$9.1\,\%$
• Trained and evaluated considering collisions, using setpoints	$2.20\,\mathrm{s}^{-1}$	$0.0\,\%$	$7.0\,\%$

Table 4.3 indicates the performance of agents trained for the specified reaching tasks. Compared to the random agents shown in Table 4.2, the number of target points reached per second increased substantially in all experiments. As for the random agents, the closest distance to a collision was always greater than zero, showing that no collision occurred during the evaluation. To investigate the impact of the backup policy on the task performance, several ablation studies were conducted. In a first experiment with the humanoid robot ARMAR-6, a random agent was employed without using a backup policy to account for collisions. As a result, collisions occurred in most episodes. Next, an agent was trained without using a backup policy. Contrary to the experiments shown in Table 4.3, the rewards were computed using the actual robot positions rather than the position setpoints. This was done to discourage the learning of trajectories that simply pass through the obstacles. While the agent learned to reach the target points, collisions occurred in around $65\,\%$ of all episodes. The occurrence of collisions could be avoided by using a backup policy based on braking trajectories during evaluation. However, in this case, the task performance reduced from 2.32 target points per second to 1.86 target points per second and the proportion of adjusted actions was relatively high. When using the backup policy during training and evaluation, the action adjustment rate was considerable lower and a better task performance could be achieved. Nevertheless, the adjustment of actions led to a slight decrease in performance compared to the agent trained without using a backup policy. However, since the generated trajectory setpoints are collision-free, it is possible to compute the reward based on the trajectory setpoints rather than the actual values. In this case, the decrease in performance could be reduced as the delay caused by the trajectory controller was avoided.

Compliance with torque limits While the previous part of the evaluation focused on collision avoidance, Table 4.5 shows the results of experiments in which braking trajecto-

Table 4.5.: Experiments conducted to evaluate the compliance with torque limits when using a backup policy based on braking trajectories. The results were obtained by simulating 900 episodes with a duration of 8 seconds. Adapted from [77].

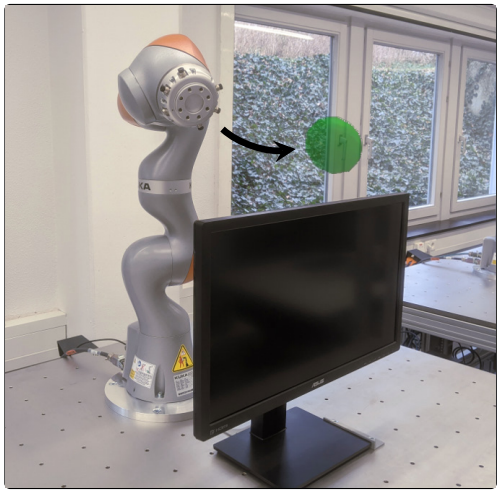
Configuration	Target points per second	Episodes with collisions	Episodes with torque violations
One industrial robot, maximum torque reduced by $40\,\%$			
• Trained and evaluated considering collisions only	1.11	$0.0\,\%$	$52.3\,\%$
• Trained considering collisions, evaluated considering collisions and torque limits	1.03	$0.0\,\%$	$0.0\,\%$
Two industrial robots, alternating target points, maximum torque reduced by $40\,\%$			
• Trained and evaluated considering collisions only	1.05	$0.0\,\%$	$84.3\,\%$
• Trained considering collisions, evaluated considering collisions and torque limits	0.77	$0.0\,\%$	$0.0\,\%$

Table 4.6.: Analysis of the computational effort when using a backup policy based on braking trajectories. The table shows the highest occurring ratio between the computation time and the trajectory duration when simulating 100 episodes with a duration of 8 seconds using an Intel i9-9900K CPU. Adapted from [77].

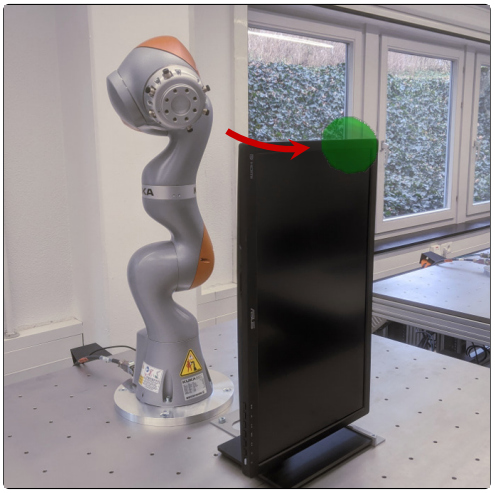
Configuration	One industrial robot	ARMAR-6 with a single target point
• Considering neither collisions nor torque limits	$7.1\,\%$	$8.1\,\%$
• Considering collisions only	$12.0\,\%$	$30.4\,\%$
• Considering torque limits only	$16.4\,\%$	$32.9\,\%$
• Considering collisions and torque limits	$19.9\,\%$	$52.9\,\%$

ries are used to avoid torque limit violations. To increase the number of potential torque limit violations, the maximum torque of the industrial robots was reduced by $40\,\%$ compared to the limits specified by the robot manufacturer. Under these conditions, torque limit violations occurred in $52\,\%$ of all episodes when using a single robot and in $84\,\%$ of all episodes when using two robots. In both cases, no more torque limit violations occurred when detecting potential violations during a background simulation.

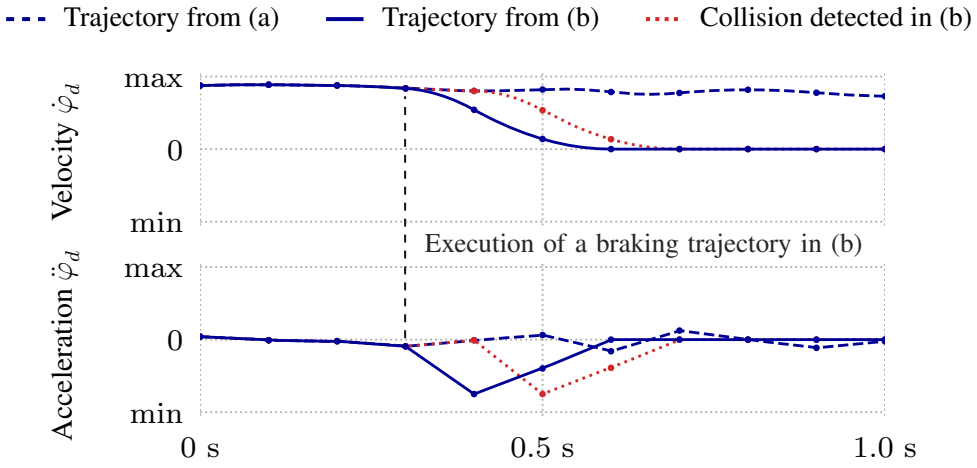
Computational effort The computational effort of the presented method is analyzed in Table 4.6. To this end, 100 episodes were simulated using different configurations. The table specifies the highest occurring ratio between the computation time and the trajectory duration for each configuration. In all experiments, the computation time was significantly shorter than the duration of the generated trajectories. When using a single industrial robot, the ratio is around $7\,\%$ if no background simulations are carried out.



(a) If the monitor is positioned as during training, the target point is reached.



(b) When rotating the monitor, a braking trajectory is executed to avoid a collision.

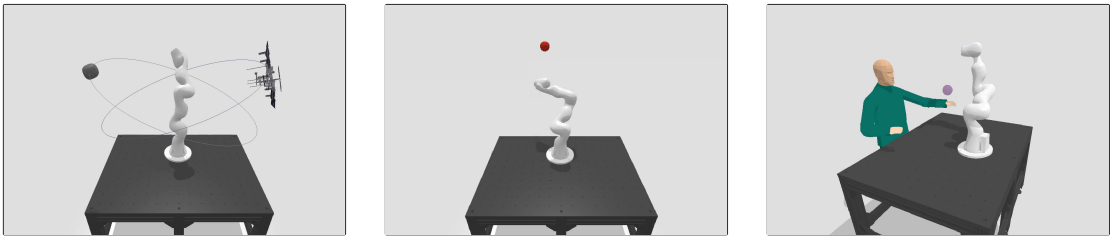


(c) The trajectories from (a) and (b) shown for one exemplary joint. In (b), a braking trajectory is executed because a collision is detected during a background simulation of the red trajectory.

Figure 4.6.: A sim-to-real transfer is conducted to demonstrate that trajectories can be generated in real time and to show that safety violations can be avoided even if obstacles are modified after the training phase. Adapted from [77].

Using background simulations to avoid collisions results in a ratio of $12\,\%$, while a ratio of $16\,\%$ is obtained when avoiding torque limit violations. If both collisions and torque limits are taken into account, the ratio is around $20\,\%$. For the humanoid robot ARMAR-6, the ratio ranges from $8\,\%$ if no safety restrictions are considered to $53\,\%$ when considering both collisions and torque limits. The computational effort for the safety checks largely depends on the time horizon taken into account during the background simulations. Since braking trajectories quickly lead to a safe goal state, a short time horizon can be selected. As a result, a backup policy based on braking trajectories is well suited for real-time applications.

Sim-to-real transfer For the environment with a single industrial robot, the real-time capability is demonstrated by performing a sim-to-real transfer. With the presented



(a) Space **(b)** Ball **(c)** Human

Figure 4.7.: Environments with moving obstacles in which a learned backup policy is used to avoid collisions. The figure is adapted from [83].

method, background simulations are conducted during and after the training phase. Provided that the simulation environment is updated, obstacles can be modified after training without causing safety violations. This feature is demonstrated by an experiment visualized in Figure 4.6. During training, the monitor in front of the robot is positioned as shown in Figure 4.6 (a). In this configuration, the target point, which is represented by a green sphere, can be reached. When rotating the monitor as shown in Figure 4.6 (b), the selected target point can no longer be reached. Since the task policy is unaware of the modified obstacle, the robot still tries to reach the target point. However, as can be seen in Figure 4.6 (c), a braking trajectory is executed so that a collision with the rotated monitor is prevented.

In summary, a backup policy based on braking trajectories was successfully used to prevent collisions and torque limit violations. Specifically, the evaluation of the presented method was conducted by learning various reaching tasks with an industrial robot and a humanoid robot using model-free RL. As a prerequisite, the environments were free of moving obstacles so that a standstill of the robots could be regarded as a safe goal state. Under this assumption, the safety conditions for background simulations outlined in section 4.4 could be satisfied. In order to analyze the impact of the backup policy on the task performance, several ablation studies were conducted. As a result, the use of actions from the backup policy led to a certain decrease in performance. However, when using the backup policy not only during evaluation but also during the training phase, the impact on the task performance was limited. An analysis of the computational effort showed that the presented method is well suited for real-time applications. For an environment with a KUKA iiwa robot, the real-time capability was demonstrated by performing a sim-to-real transfer.

4.6. Learning a backup policy via model-free RL

In the previous section, a backup policy based on braking trajectories was examined. As a prerequisite, it was assumed that safety violations cannot occur if all robot joints are at a standstill. While this assumption is reasonable for many environments, there are also scenarios in which safety violations can occur even if the robot joints are fully stopped. This section focuses on the particular example of environments with moving obstacles. It is assumed that the movements of the obstacles are not influenced by the presence of the robot. In order to avoid a collision, the robot might need to perform an evasive movement in certain situations. One way to achieve this behavior is to use a

backup policy that actively avoids collisions with moving obstacles. As will be shown in the following part of this section, it is possible to learn such a backup policy using model-free RL. In the context of this work, the use of model-free RL offers the advantage that the action mapping proposed in chapter 3 can be used for both the task policy and the backup policy. Therefore, compliance with the kinematic joint limits is ensured no matter which policy is executed. As an additional advantage, model-free RL can be used to learn policies in stochastic environments, which is important if obstacles can move in a non-deterministic way.

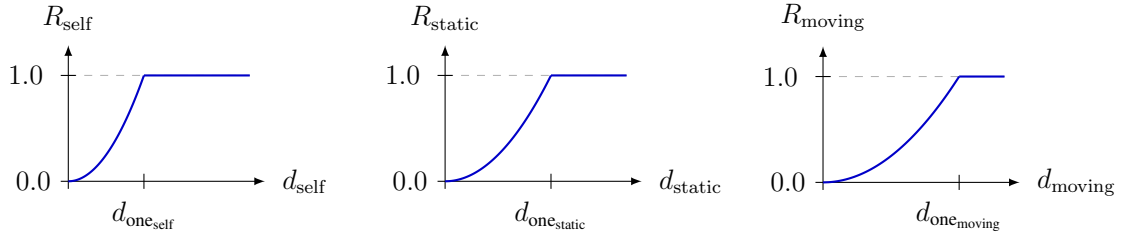
Figure 4.7 shows three environments with moving obstacles, which are used as examples for the study presented below:

- **Space:** In the space environment, the robot is orbited by a miniature model of the International Space Station (ISS) and a model of an asteroid. The orbits and orbital periods of both obstacles are constant so that their future position can be accurately predicted. Thus, the space environment serves as an example of a deterministic environment.
- **Ball:** The ball environment was already introduced to explain the basic principle of the presented method. Specifically, a ball is thrown towards the robot from a random direction. Since the ball is affected by gravity, it follows a parabolic trajectory. Once the ball misses the robot, a new ball is thrown. The initial position of the ball is chosen at random, making the environment stochastic. To determine the release angle, a random robot position is selected. The angle is then adjusted so that the ball would hit the robot if it were located at the randomly selected position. This procedure ensures that bringing the robot to a standstill is not a viable safety strategy in the ball environment.
- **Human:** The human environment serves as an example to examine how humans and robots can operate in close proximity. To this end, a geometric model of a human is positioned next to the table on which the robot is mounted. For reasons of simplification, only the arms of the human are moved. Specifically, the human is controlled by a neural network trained to reach target points using the method introduced in section 4.5. Since the target points are selected at random, the human moves in a stochastic manner.

The remaining part of this section is structured as follows: First, the training of the backup policy is explained. Subsequently, the learning performance is evaluated based on the three environments introduced above. Finally, the backup policies are used to avoid collisions when executing a task policy. More precisely, the occurrence of safety violations is evaluated for the particular case of using a random agent as task policy.

4.6.1. Training of the backup policy

In the following, the problem of learning a backup policy is formalized as a Markov decision process $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$. Subsequently, it is explained how the environment is initialized during the training phase. The initialization of the robot is particularly important, as the backup policy is supposed to generate well-performing evasive movements across a wide range of motion states.



(a) Self-collision reward **(b)** Static obstacle reward **(c)** Moving obstacle reward

Figure 4.8.: The reward components used for the training of the backup policy. The figure is adapted from [83].

State space The backup policy is designed to be task-independent and trained prior to a specific task policy. Consequently, the state of the backup policy does not contain any task-specific information, e.g., the position of a target point when learning a reaching task. Instead, a state $s_t \in \mathcal{S}$ of the backup policy is composed of two parts: The kinematic state of each robot joint (φ_{d_t}, $\dot{\varphi}_{d_t}$, $\ddot{\varphi}_{d_t}$) and an environment-specific part used to indicate the current state of the moving obstacles. Further details on the environment-specific part are given below:

- **Space:** The environment-specific part indicates the position of the asteroid and the space station along their orbital path.
- **Ball:** The environment-specific part indicates the current position and velocity of the ball.
- **Human:** The human in the human environment is controlled by a neural network trained to reach target points. To this end, the joints of the human are modeled similar to the joints of the humanoid robot ARMAR-6 in section 4.5. The state of the neural network controlling the human includes the kinematic state of its joints ($\varphi^H_{d_t}$, $\dot{\varphi}^H_{d_t}$, $\ddot{\varphi}^H_{d_t}$) and the position of the current target point. For the training of the backup policy, the environment-specific part contains ($\varphi^H_{d_t}$, $\dot{\varphi}^H_{d_t}$, $\ddot{\varphi}^H_{d_t}$) only.

Action space Using the action mapping proposed in chapter 3, an action $a_t \in \mathcal{A}$ consists of one scalar $\in$ [-1,1] per robot joint being controlled.

Reward function The backup policy is trained to avoid self-collisions, collisions with static obstacles, and collisions with moving obstacles. At time step t, an action a_t specifies a trajectory from t to $t+1$. To assess the impact of an action a_t on the occurrence of collisions, the resulting robot position at $t+1$ is used to determine the minimum distance to a self-collision d_{self}, to a collisions with a static obstacle d_{static}, and to a collision with a moving obstacle d_{moving}. Based on these distances, the reward components R_{self}, R_{static}, and R_{moving} are computed as shown in Figure 4.8. The calculation of the reward components differs only with regard to the predefined threshold values $d_{\text{one}_{\text{self}}}$, $d_{\text{one}_{\text{static}}}$, and $d_{\text{one}_{\text{moving}}}$. In addition to these components, a termination bonus R_B is introduced. The bonus is granted if an episode terminates without a collision occurring. Considering all four components, the reward is calculated as follows:

$$R = \alpha \cdot R_{\text{self}} + \beta \cdot R_{\text{static}} + \gamma \cdot R_{\text{moving}} + R_B, \tag{4.11}$$

with α, β, and γ being weighting factors.

Table 4.7.: Performance metrics of backup policies trained to avoid collisions. Adapted from [83].

Environment	**A:** Training settings		**B:** Evaluation settings	
	Episodes without a collision		Time until collision	
	Random agent	Trained agent	Random agent	Trained agent
• Space	38 %	91 %	2.6 s	> 40 000 s
• Ball	48 %	91 %	3.3 s	85.4 s
• Human	51 %	94 %	3.4 s	> 40 000 s

Termination An episode is either terminated after a certain number of decision steps or earlier if a collision is detected. The selected number of decision steps should be large enough to allow the robot to dodge the obstacles in the environment. Due to the termination bonus, early termination is discouraged as part of the learning process.

Initialization To learn well-functioning backup motions for a wide range of kinematic states, the environment is initialized in a random state. Specifically, the robot is not only initialized at a random position, but also at a random velocity and acceleration. As a first step, random joint positions are chosen until a collision-free robot position is found. Next, a random velocity and acceleration is selected for each robot joint. However, not every random kinematic state is feasible with respect to the kinematic joint limits (3.1) - (3.4). For that reason, the method presented in chapter 3 is used to compute a range of feasible accelerations $[\mathrm{a}_{t+1_{\min}}, \mathrm{a}_{t+1_{\max}}]$ for the selected random kinematic state $(\mathrm{p_t}, \mathrm{v_t}, \mathrm{a_t})$. If the computed range is empty, the kinematic state is regarded as invalid. In this case, new velocities and accelerations are randomly chosen until a valid initial kinematic state is found.

4.6.2. Evaluation

The evaluation presented below is composed of two parts: First, the performance of policies trained to avoid collisions is analyzed. Second, the occurrence of collisions is evaluated for a random agent when employing the previously trained policies as backup policies. In both cases, the evaluation is conducted based on the environments with moving obstacles shown in Figure 4.7.

Training performance of the backup policy As described in section 4.6.1, the backup policy is trained to produce well-performing evasive movements for a wide range of environmental states. During the training, the robot is initialized in random kinematic states. For the experiments presented below, the decision frequency f_D is set to $10\,\mathrm{Hz}$. An episode is terminated after 20 decision steps or sooner if a collision is detected. Based on these settings, the left part of Table 4.7 shows the proportion of episodes that are completed without collisions. The data clearly indicates that a trained agent causes significantly fewer collisions than a random agent. While collisions may still occur when using the trained policy, it is important to note that the robot can be initialized in kinematic states where collision avoidance is not possible. The right part of Table 4.7 shows the average time to a collision when the robot is initialized in states where executing the

Table 4.8.: The average time to a collision for a random agent when using the previously trained backup policy and a background simulation with a time horizon of N. Adapted from [83].

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					
• Deterministic	2.7 s	6.8 s	170.4 s	>2000 s	>2000 s
Ball					
• Stochastic	3.6 s	10.4 s	64.1 s	160.3 s	116.3 s
• Deterministic	3.8 s	10.9 s	68.7 s	164.7 s	181.6 s
Human					
• Stochastic	3.7 s	11.1 s	62.9 s	72.3 s	64.5 s
• Deterministic	3.7 s	11.3 s	700.6 s	>2000 s	>2000 s

trained policy does not result in a collision within the first two seconds. In the space environment and in the human environment, the policy can navigate the robot to an area without obstacles during this time. Therefore, collisions are extremely rare. In the ball environment, the robot can be hit by a ball anywhere in the working area. As a result, a collision occurs on average after $85.4\,\mathrm{s}$ compared to $3.3\,\mathrm{s}$ when using a random agent.

Collision avoidance for random agents In a next step, the performance of the backup policies is evaluated for the case that the task policy is represented by a random agent. This case reflects the situation typically encountered at the beginning of a training process. Table 4.8 shows the average time to a collision depending on the selected time horizon of the background simulation. The initial states of all experiments are selected in a way that the backup policy can be safely executed during the first three seconds of each episode. As a consequence, collisions caused by an unsafe initialization do not occur. In the space environment, obstacles move in a deterministic manner. Contrary to that, the ball environment and the human environment behave stochastically. In these environments, the background simulation is carried out for one potential future development of the environment, which, however, does not necessarily reflect the actual development of the environment. For reasons of comparison, additional experiments are conducted based on the assumption that the obstacles move in a predictable way. When selecting a time horizon of $N = 0$, the background simulation only covers the period from the current time step t to the next time step $t + 1$, as resulting from action a_t^T of the task policy. In all environments, a collision occurs on average within less than four seconds when selecting this time horizon. It can be concluded that collisions are detected too late to perform an evasive movement when considering the next time interval only. As shown in Table 4.8, the average time to a collision increases significantly if the time horizon of the background simulation is extended. This applies in particular if the obstacles in the environment move deterministically. When selecting a time horizon of $N = 20$, the average time to a collision in the space environment increased by a factor of more than 500. In stochastic environments, however, collisions do not necessarily decrease if the time horizon is extended, as the background simulation deviates more and more from the actual temporal development of the environment.

Table 4.9.: The proportion of collisions that do not involve moving obstacles when using a random agent and background simulations with a time horizon of N. Adapted from [83].

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					
• Deterministic	57 %	32 %	0 %	0 %	0 %
Ball					
• Stochastic	78 %	43 %	6 %	4 %	9 %
• Deterministic	82 %	43 %	0 %	0 %	0 %
Human					
• Stochastic	76 %	53 %	13 %	13 %	11 %
• Deterministic	74 %	67 %	0 %	0 %	0 %

Nevertheless, the time to a collision increased by a factor of around 20 in the stochastic human environment and by a factor of around 40 in the stochastic ball environment when selecting a time horizon of $N = 20$ compared to a time horizon of $N = 0$.

Analyzing the causes of collisions For the experimental settings described above, Table 4.9 shows the relation between the time horizon of the background simulation and the proportion of collisions that do not involve moving obstacles. When selecting a time horizon of $N = 0$, most of the collisions do not involve moving obstacles. Instead, self-collisions and collisions with the table on which the robot is mounted are prevalent. However, as the time horizon increases, moving obstacles become the primary cause of collisions. With regard to the deterministic environments, self-collisions and collisions with the table are almost negligible if a time horizon of $N = 5$ is chosen. In the stochastic environments, however, these types of collisions still occur even if a longer time horizon is selected, as the background simulation does not necessarily reflect the actual temporal development of the environment.

Computational effort The computational effort required to perform the background simulations is analyzed in Table 4.10. Specifically, the table shows the average ratio between the computation time and the duration of the robot trajectory generated within that time for various time horizons N. This ratio is an important indicator, as it affects whether trajectories can be generated in real time. Moreover, the computational effort involved in generating the robot trajectories influences the time required to train a task policy. Using an Intel i9-9900K as CPU, the ratio exceeded $100\,\%$ for a time horizon greater than or equal to $N = 5$, meaning that trajectories cannot be generated in real time. With respect to collision avoidance, the best results for stochastic environments were obtained for a time horizon of $N = 20$. Using this time horizon, the ratio exceeded $200\,\%$ for the ball environment and $400\,\%$ for the human environment. The high computational effort involved in performing background simulation motivates the use of faster data-based risk estimators, as elaborated in chapter 5.

Table 4.10.: The average ratio between the computation time and the duration of the generated trajectory when using background simulations with a time horizon of N and an Intel i9-9900K as CPU. Adapted from [83].

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	38 %	48 %	119 %	235 %	303 %
• Ball	27 %	42 %	119 %	239 %	290 %
• Human	57 %	89 %	205 %	474 %	673 %

In summary, the use of a backup policy trained via model-free RL proved to be effective in avoiding collisions with moving obstacles. When selecting random actions a^T, the time to a collision could be increased by a factor of 20 in the human environment, 40 in the ball environment, and more than 500 in the space environment. However, compared to self-collisions and collisions with static obstacles, the background simulations had to cover a larger time horizon in order to effectively detect collisions with moving obstacles. As the resulting computational effort for the background simulations was found to be high, faster data-based risk estimators are introduced in the following chapter.

4.7. Summary

This chapter presented the concept of using background simulations to avoid safety violations when learning robot trajectories via reinforcement learning. In addition to the normal task policy, a backup policy was introduced. The background simulations were conducted to determine whether actions of the task policy can be carried out safely. If a safety violation was detected during a background simulation, the action from the task policy was replaced by an action from the backup policy. Using the action mapping proposed in chapter 3, it was ensured that the resulting trajectories adhered to kinematic joint constraints. For the presented strategy, safety conditions were established under which safety violations can be entirely prevented. In particular, it was shown that the environment must be predictable, the time horizon of the background simulation must be sufficiently long, and the initial state of the environment must be chosen appropriately.

To avoid safety violations in environments without moving obstacles, a backup policy based on braking trajectories was proposed. As a prerequisite, it was assumed that no further safety violations can occur if all robot joints are fully stopped. For this particular case, it was shown that the established safety conditions can be satisfied. The method was evaluated by successfully learning reaching tasks without causing self-collisions, collisions with static obstacles, and torque limit violations. As the duration of the braking trajectories was relatively short, the computational effort for the background simulations was moderate. Consequently, it was possible to generate trajectories in real time, which was demonstrated by performing a sim-to-real transfer with a KUKA iiwa robot.

By using a backup policy trained to actively avoid collisions, the presented concept could be extended to environments with moving obstacles. Similar to the task policy, the backup policy was trained using model-free RL, however, without having access to task-specific information. The evaluation was carried out for three different environments, with two of them behaving stochastically. Using a random agent as task policy, the occurrence of collisions could be significantly reduced in all environments. Compared to self-collisions and collisions with static obstacles, the time horizon of the background simulations had to be increased to account for moving obstacles. However, in stochastic environments, the validity of the background simulations was found to decrease when longer time horizons were taken into account. In addition, the computational effort for the background simulations was found to be high. Both of these findings motivate the use of data-based risk estimators, which are introduced in the following chapter.

5. Using data-based risk estimators to speed-up risk assessments

This chapter examines the use of data-based estimators to assess the risk of safety violations when learning robot trajectories. In the previous chapter, it was shown that safety violations can be effectively detected by simulating the execution of a backup policy in a physics simulator. Based on results initially published in [83], this chapter analyzes how data from previous executions of the backup policy can be utilized for this purpose. Compared to performing a background simulation in a physics simulator, the use of data-based risk estimators offers several advantages:

- First, the computing power required to generate trajectories in real time is significantly lower.
- Second, stochastic environments can be addressed in a straightforward manner.
- Third, data-based risk estimators can also be used in scenarios where no physics simulator is available.

As a drawback, a sufficient amount of data must be available to train the estimator. In addition, risky actions can be falsely classified as harmless, as the risk estimation is not always correct. The following study examines the effectiveness of risk estimators for collision avoidance in environments with moving obstacles. Specifically, four different options to estimate the risk of collisions are analyzed and compared. In all cases, the risk estimation is based on neural networks trained via supervised learning.

The evaluation is conducted using three different environments with moving obstacles introduced in the previous chapter. First, the occurrence of collisions is analyzed for a random agent. Subsequently, the risk estimators are employed during the training of a reaching task and a basketball task. For the reaching task, the performance is compared with an alternative safety strategy using *quadratic programs* (QPs). Finally, a sim-to-real transfer with a KUKA robot is conducted, showing that the presented approach is fast enough to enable real-time execution.

5.1. Problem description

The problem of learning robot trajectories is formalized as a Markov decision process $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$. Specifically, a task policy π_T is trained to maximize the expected sum of future rewards using model-free RL. As an additional constraint, the occurrence of safety violations should be avoided during and after the training process of the task policy. For that purpose, an action a_t^T generated by the task policy at state s_t should only be executed if the estimated risk $\bar{Q}_{\pi_B}(s_t, a_t^T)$ does not exceed a predefined threshold value $\bar{q}_{Th}$. In

the scenario under consideration, $\bar{Q}_{\pi_B}(s_t, a_t^T)$ indicates the likelihood of a safety violation when executing action a_t^T, assuming that the following N actions are generated by a backup policy π_B. The backup policy is trained to avoid the occurrence of safety violations using model-free RL. During the training process of the task policy π_T, the backup policy π_B is not updated. Assuming that $\bar{Q}_{\pi_B}(s_t, a_t^T)$ exceeds the risk threshold $\bar{q}_{Th}$, an action a_t^B from the backup policy π_B is executed:

$$a_t = \begin{cases} a_t^T & \text{if } \bar{Q}_{\pi_B}(s_t, a_t^T) \leq \bar{q}_{Th} \\ a_t^B & \text{otherwise} \end{cases} \tag{5.1}$$

The problem examined in this study is the approximation of the risk function $\bar{Q}_{\pi_B}(s, a)$ based on data from previous executions of the backup policy. Specifically, neural networks are used to estimate the risk of collisions in environments with moving obstacles. The accuracy of the risk estimation influences both the compliance with safety constraints and the task performance. If risky actions are falsely classified as harmless, safety violations are likely to occur. Falsely classifying harmless actions as risky, on the other hand, can reduce the task performance as the backup policy does not contribute to the fulfillment of the desired task.

5.2. Relation to previous studies

The basic idea of avoiding safety violations by utilizing a backup policy and a risk function has been examined in the context of different application scenarios. An overview of previous research is provided in section 2.2.2.2. Most notably, the presented approach differs from related studies in three main aspects:

- **Risk estimation:** The decision whether to use the task policy or the backup policy is made based on a risk function $\bar{Q}_{\pi_B}(s, a)$, which is approximated using neural networks. Compared to using task-specific criteria [186], a data-based approach offers greater flexibility with regard to the desired application scenario. The subscript $_{\pi_B}$ in $\bar{Q}_{\pi_B}(s, a)$ indicates that the risk of an action a is estimated under the assumption that the following actions are generated by the backup policy. As an alternative, it is also possible to estimate a risk function $\bar{Q}_{\pi}(s, a)$, where π is the policy resulting from the interaction between the task policy and the backup policy [167]. However, in this case, the risk function depends on the task policy and needs to be updated if the task policy is adjusted.
- **Backup policy:** In this study, the backup policy π_B is trained in advance to the task policy using model-free RL. In comparison to backup policies relying on model-based controllers [54, 43], no explicit model of the system dynamics is required. As an additional advantage, the action space and the decision frequency of the backup policy can be easily aligned with the task policy. Compared to deriving the backup policy from a risk function $\bar{Q}_{\pi}(s, a)$ [167], the backup policy in the presented approach does not depend on the task policy. As a result, it can be utilized to learn different task policies.
- **Application scenario:** The presented approach is applied for learning collision-free goal-directed robot trajectories in environments with moving obstacles. As the

action mapping from chapter 3 is utilized, fast trajectories can be generated without violating kinematic joint constraints. In contrast to previous work, various methods to approximate the risk function $\bar{Q}_{\pi_B}(s, a)$ are analyzed and compared. Moreover, the applicability to stochastic environments is systematically investigated.

5.3. Data-based risk estimators

In this chapter, data-based risk estimators are employed to decide whether to use the task policy π_T or the backup policy π_B. Specifically, a risk function $\bar{Q}_{\pi_B}(s, a)$ is approximated using neural networks. This section outlines the generation of training data for the risk estimation and the training of the neural networks based on supervised learning.

5.3.1. Generation of training data

The training dataset for the risk estimation is generated using a physics simulator. In theory, it is also possible to gather training data through real-world experiments. However, as safety violations are inevitable when collecting training data, the experiments must be carried out in a specially protected environment.

Each data point of the training dataset consist of a tuple (s_t, a_t, s_{t+1}, $\bar{q}$). In order to generate a data point, the environment is initialized in a random state s_t, using the same initialization procedure as described in section 4.6.1. Next, an action a_t is randomly chosen and executed. Starting from state s_{t+1}, the backup policy π_B is used to generate actions for the following N time steps. Hence, a total of $N + 1$ actions are executed for each data point. If a collision is detected during the execution of these $N + 1$ actions, the risk signal $\bar{q}$ is set to 1, otherwise to 0.

To generate the training dataset, the procedure described above is repeated for a large number of states s_t and actions a_t. Typically, a risk signal of 1 occurs less frequently than a risk signal of 0. To compensate for this imbalance, data points with a risk signal of 1 are selected more frequently during the training process.

5.3.2. Training via supervised learning

Using the data points (s_t, a_t, s_{t+1}, $\bar{q}$) from the training dataset, fully connected neural network are trained for a binary classification task. Specifically, two different ways to estimate the risk of a collision $\hat{\bar{q}}$ are analyzed and compared. In case of a so-called *action-based* risk network, the risk $\hat{\bar{q}}$ is predicted using the state s_t and the action a_t as input signals. Contrary to that, a *state-based* risk network uses the state s_{t+1} as input signal for the prediction. In both cases, the risk signals $\bar{q}$ from the dataset serve as ground truth values and the training is performed using a cross-entropy loss function.

For the binary classification, two different types of errors can be distinguished:

- **False positives**: The risk is classified as 1, but the risk signal in the dataset is 0.
- **False negatives**: The risk is classified as 0, but the risk signal in the dataset is 1.

When using a risk estimator to train a task policy, *false positives* reduce the task performance, while *false negatives* increase the probability of safety violations. The ratio of *false positives* and *false negatives* depends on the selected risk threshold $\bar{q}_{Th}$. During the training of the risk estimator, the ratio can be additionally influenced by using a weighted loss function. In the scenario under consideration, *false negatives* are generally considered more harmful, as safety violations need to be avoided. For that reason, a higher weight is assigned to data points with a risk signal $\bar{q}$ of 1.0

When utilizing data-based risk estimators, accounting for stochastic environments is straightforward. While the initial risk signals $\bar{q}$ may vary depending on the further development of the environment, the predicted risk $\hat{\bar{q}}$ indicates the probability of a safety violation.

5.4. Risk-aware action generation

The risk-aware action generation presented in this chapter is based on a task policy π_T, which is represented by a task network, and a backup policy π_B, which is represented by a backup network. Both policies rely on the mathematical framework of a Markov decision process $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ and are employed in environments with moving obstacles. The training of the backup policy is carried out as described in section 4.6.1. In order to avoid collisions during the training of the task policy, the backup policy is utilized if the estimated risk $\hat{\bar{q}}$ exceeds a risk threshold $\bar{q}_{Th}$. Figure 5.1 provides an overview of the steps involved in this procedure.

The first step is to generate an action a_t^T with the task network. As shown in Figure 5.1, the state of the task policy is composed of three parts $s_{t_{Ki}}$, $s_{t_{Mo}}$, and $s_{t_{Ta}}$, where $s_{t_{Ki}}$ indicates the kinematic state of the robot joints, $s_{t_{Mo}}$ describes the state of the moving obstacles, and $s_{t_{Ta}}$ provides additional task-specific information. The action a_t^T is composed of two parts $a_{t_{Ki}}^T$ and $a_{t_{Ta}}^T$. While the kinematic part $a_{t_{Ki}}^T$ is used to control the robot joints, an optional task-specific part $a_{t_{Ta}}^T$ can be used to encode additional control signals. In contrast to the task policy, the backup policy does not make use of the task-specific components $s_{t_{Ta}}$ and $a_{t_{Ta}}^T$.

The second step is to estimate a risk value $\hat{\bar{q}}$ using the action-based risk network or the state-based risk network described in the previous section. Specifically, four different methods are analyzed in this study:

- **(A) Action-based risk estimation:** The risk estimation is performed by the action-based risk network. As shown in Figure 5.1, the input signal of the risk network is composed of $s_{t_{Ki}}$, $s_{t_{Mo}}$, and $a_{t_{Ki}}^T$. The task-specific components $s_{t_{Ta}}$ and $a_{t_{Ta}}^T$ are not used for the risk estimation. From a conceptual perspective, the use of both state and action components is consistent with the desired risk function $\bar{Q}_{\pi_B}(s, a)$.

For the remaining three methods, the state-based risk network is used. The state-based risk network is trained using the state components $s_{t+1_{Ki}}$ and $s_{t+1_{Mo}}$. Compared to the action-based risk network, the dimensionality of the learning problem is reduced as the input signal does not contain the action component $a_{t_{Ki}}^T$. However, $s_{t+1_{Ki}}$ and $s_{t+1_{Mo}}$ are not directly accessible, which requires further approximations.

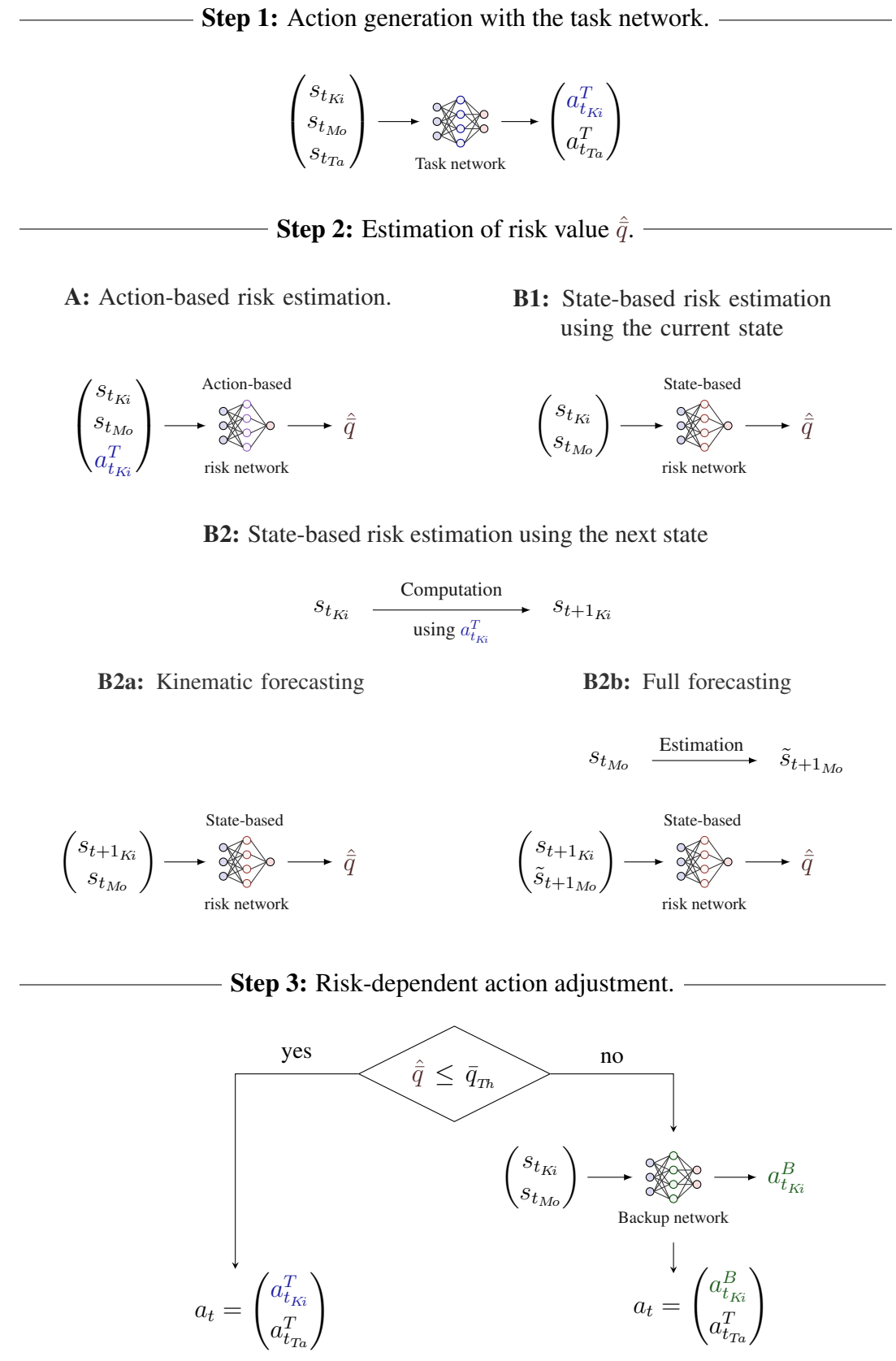
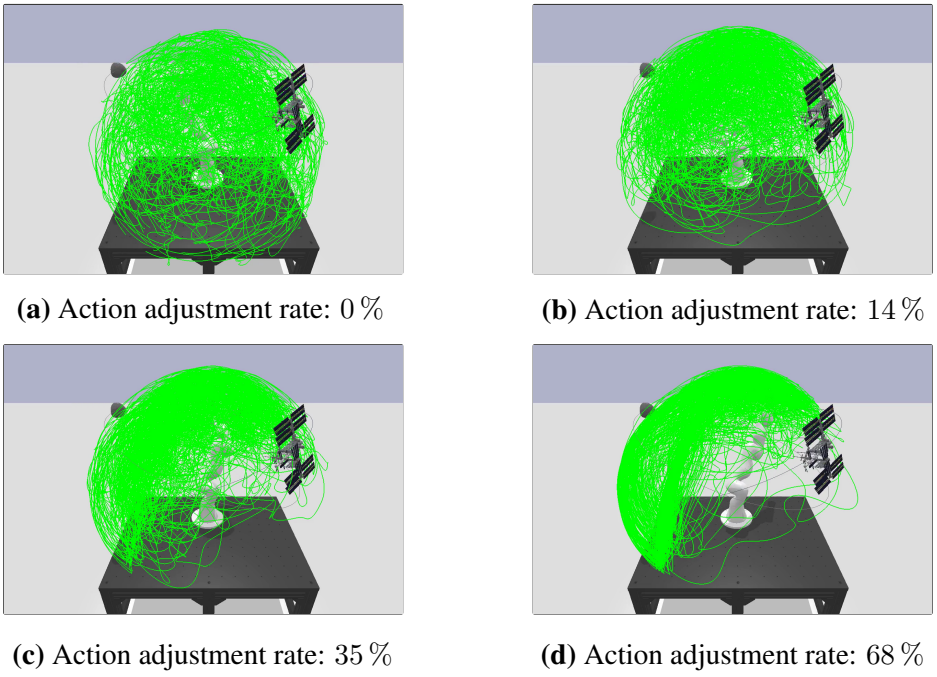


Figure 5.1.: The steps involved in the risk-aware generation of actions based on a task network, a backup network and a risk network. Specifically, the figure visualizes four different methods (A, B1, B2a, B2b) to estimate a risk value $\hat{\bar{q}}$ using either an action-based risk network or a state-based risk network. The figure is adapted from [83].



(a) Action adjustment rate: 0 %

(b) Action adjustment rate: 14 %

(c) Action adjustment rate: 35 %

(d) Action adjustment rate: 68 %

Figure 5.2.: The impact of the action adjustment rate on the exploration, illustrated for the space environment. Using an action-based risk network and a random agent as task policy, the green path visualizes the position of the last robot link when executing a trajectory with a duration of 1000 seconds.

The three state-based methods described below differ with respect to the representations used for $s_{t+1_{Ki}}$ and $s_{t+1_{Mo}}$:

- **(B1) State-based risk estimation using the current state:** The risk estimation is performed based on the state components $s_{t_{Ki}}$ and $s_{t_{Mo}}$. The impact of the action component $a^T_{t_{Ki}}$ is ignored.

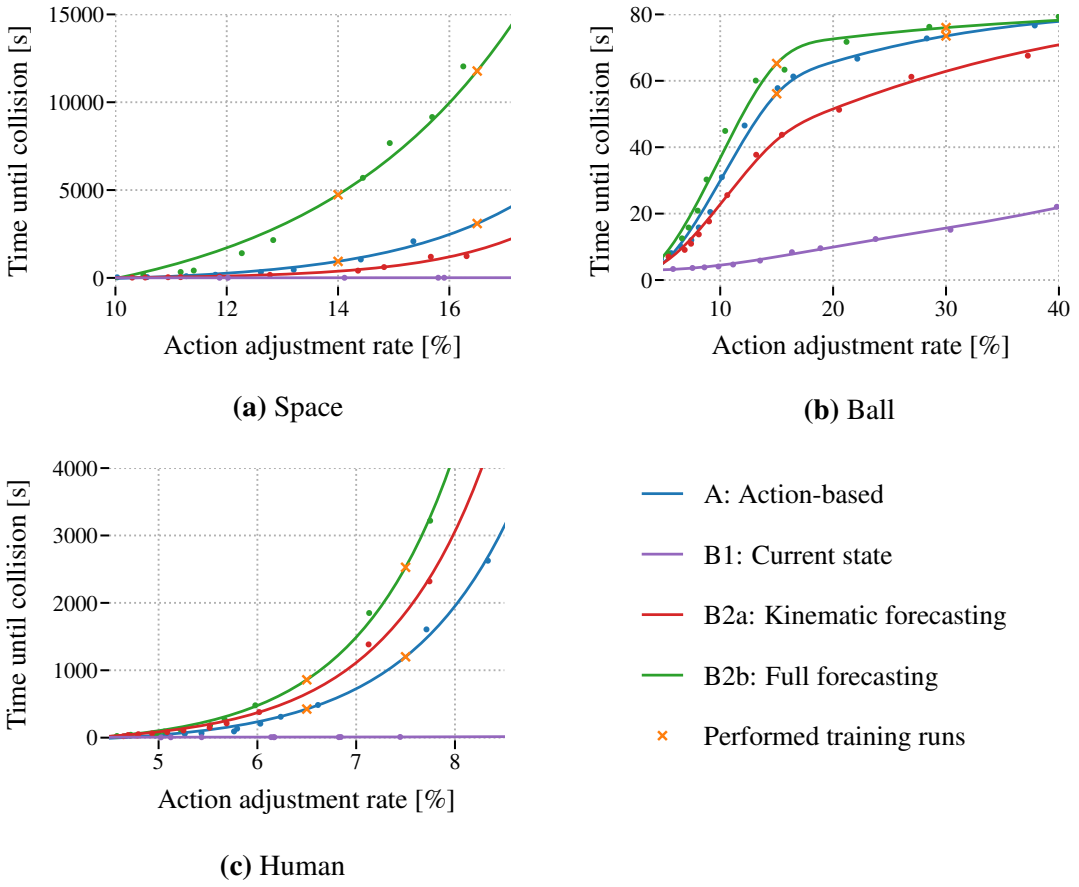
- **(B2a) State-based risk estimation with kinematic forecasting:** The action component $a^T_{t_{Ki}}$ is used to compute $s_{t+1_{Ki}}$, the next kinematic state of the robot joints. The risk estimation is conducted based on $s_{t+1_{Ki}}$ and $s_{t_{Mo}}$.

- **(B2b) State-based risk estimation with full forecasting:** As with (B2a), the kinematic state $s_{t+1_{Ki}}$ is computed based on $a^T_{t_{Ki}}$. In addition, the next state of the moving obstacles $\tilde{s}_{t+1_{Mo}}$ is estimated. Subsequently, $s_{t+1_{Ki}}$ and $\tilde{s}_{t+1_{Mo}}$ are used for the risk estimation.

As a final step, the action component $a^T_{t_{Ki}}$ is replaced by $a^B_{t_{Ki}}$ if the estimated risk $\hat{\bar{q}}$ exceeds a risk threshold $\bar{q}_{Th}$:

$$a_{t_{Ki}} = \begin{cases} a^T_{t_{Ki}} & \text{if } \hat{\bar{q}} \leq \bar{q}_{Th} \\ a^B_{t_{Ki}} & \text{otherwise} \end{cases} \tag{5.2}$$

In this scenario, $a^B_{t_{Ki}}$ denotes the action of the backup policy computed using $s_{t_{Ki}}$ and $s_{t_{Mo}}$. The task-specific action component $a^T_{t_{Ta}}$ is not influenced by the estimated risk value $\hat{\bar{q}}$.



(a) Space

(b) Ball

(c) Human

Figure 5.3.: The average time to a collision as a function of the action adjustment rate when using a random agent as task policy shown for different methods to estimate the risk value $\hat{\bar{q}}$. The figure is adapted from [83].

5.5. Evaluation

The evaluation of the data-based risk estimation is carried out for the *space*, *ball*, and *human* environment introduced in the previous chapter. In a first step, the backup policies from section 4.6 are used to generate training data for the risk estimation. Subsequently, the data is used to train an action-based risk network and a state-based risk network for each environment. The following evaluation examines various aspects related to the risk estimation using risk networks. This includes the impact of the selected risk threshold $\bar{q}_{Th}$, the occurrence of collisions for random and trained agents, and the computational effort associated with the risk estimation. In addition, a comparison with a QP-based method is carried out and a sim-to-real transfer with a KUKA robot is conducted.

Impact of the selected risk threshold The risk threshold $\bar{q}_{Th}$ affects the proportion of actions from the task policy that are adjusted using the backup policy. In the following, this proportion is denoted as *action adjustment rate*. When selecting a risk threshold of 1.0, all actions from the task policy are considered safe, resulting in an action adjustment rate of $0\,\%$. With a risk threshold of 0.0, all actions are adjusted so that the action adjustment rate is $100\,\%$. When choosing a risk threshold between 0.0 and 1.0, the resulting action adjustment rate additionally depends on the current task policy.

Table 5.1.: Results obtained when training a task policy for a reaching task using either the action-based risk estimation (A) or the state-based risk estimation with full forecasting (B2b). Adapted from [83].

	Random agent	Trained agent		
Initial adjustment rate	Target points per second	Target points per second	Adjustment rate	Time until collision
Space				
• Adjustment rate 14.0 %	0.004 / 0.004	1.01 / 0.94	9.1 % / 8.8 %	423 s / 887 s
• Adjustment rate 16.5 %	0.003 / 0.002	0.93 / 0.90	10.0 % / 8.8 %	428 s / 1647 s
Ball				
• Adjustment rate 15.0 %	0.004 / 0.004	0.70 / 0.71	18.2 % / 17.7 %	68.5 s / 68.7 s
• Adjustment rate 30.0 %	0.003 / 0.003	0.52 / 0.51	29.4 % / 31.1 %	89.3 s / 83.0 s
Human				
• Adjustment rate 6.5 %	0.004 / 0.003	0.79 / 0.70	10.0 % / 12.7 %	96 s / 114 s
• Adjustment rate 7.5 %	0.003 / 0.003	0.49 / 0.63	16.3 % / 16.8 %	191 s / 139 s

Figure 5.2 illustrates how the action adjustment rate affects the exploration of the space environment when utilizing a random agent as task policy. To this end, trajectories with a duration of 1000 seconds are generated using four different risk thresholds. The green lines in Figure 5.2 visualize the areas of the environment that are explored when executing the trajectories. As can be seen in (a), the environment is explored evenly if no actions are adjusted. However, an even exploration also leads to collisions with the table. If 14 % of the actions are adjusted (b), the green path no longer intersects with the table, but the environment is still explored relatively evenly. With an action adjustment rate of 35 % (c), the lower right corner is no longer reached. If 68 % of the actions are adjusted (d), the area being explored is even more restricted.

In order to avoid collisions without excessively restricting the task policy, the risk threshold must be selected carefully. For the specific example, the operating point shown in (b) represents a reasonable compromise between avoiding collisions and exploring the environment.

Collision avoidance for random agents In a next step, the collision avoidance is analyzed quantitatively. For that purpose, experiments with different risk thresholds $\bar{q}_{Th}$ are performed using a random agent as task policy. Figure 5.3 shows the resulting relation between the action adjustment rate and the average time to a collision for the *space*, *ball*, and *human* environment. The evaluation is conducted for each of the risk estimation methods described in section 5.4. For the space environment and the human environment, the data points of the individual experiments are fitted with an exponential function. In case of the ball environment, the average time to a collision approaches a limit value and the resulting curves resemble a logistic function. This outcome is consistent with the results obtained by exclusively using the backup policy shown in Table 4.7 B. The risk estimation method *full forecasting (B2b)* requires an estimation of $\tilde{s}_{t+1_{Mo}}$, the next state of the moving obstacles. In contrast, the other three methods do not require any additional information about the moving obstacles. For the evaluation, it is assumed that

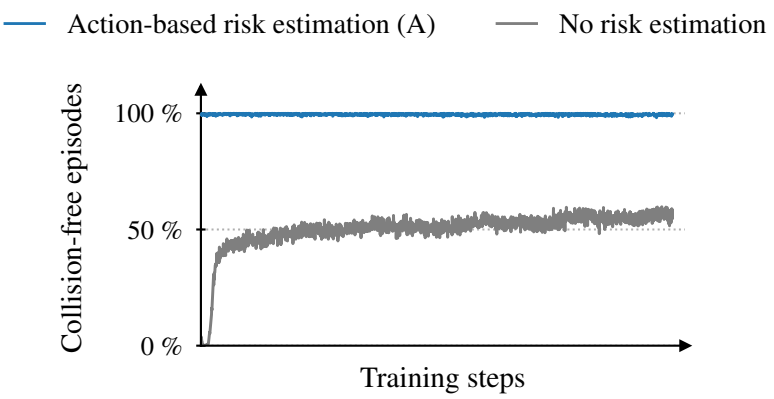


Figure 5.4.: The percentage of collision-free episodes with a duration of eight seconds when learning a reaching task in the space environment using the action-based risk estimation (A) compared to using no risk estimation. The figure is adapted from [83].

$\tilde{s}_{t+1_{Mo}}$ can be accurately forecasted. Under this assumption, the highest time to a collision is obtained using *full forecasting (B2b)*. In the space environment and the ball environment, the *action-based risk estimation (A)* leads to the second best results. For the human environment, the second best results are obtained by *kinematic forecasting (B2a)*. The results also show that the state-based risk estimation using the *current state (B1)* performs significantly worse than the other methods.

Collision avoidance for trained agents Next, the collision avoidance is analyzed when learning a task policy using model-free RL. Table 5.1 shows the training results for a reaching task performed by the industrial robot KUKA iiwa using the *action-based risk estimation (A)* and the *state-based risk estimation with full forecasting (B2b)*. The experiments are conducted using a fixed risk threshold $\bar{q}_{Th}$, chosen to establish a specific action adjustment rate at the beginning of the training process. In Figure 5.3, the selected operating points for the training runs are indicated by an orange cross. The action adjustment rate does not only depend on the selected risk threshold, but also on the current task policy. Consequently, the rate can vary during the training process. For the training runs shown in Table 5.1, the following tendency can be identified: When selecting a higher initial action adjustment rate, the average time to a collision increases while the number of target points reached per second decreases. This result is plausible as the backup policy is trained to avoid collisions but does not contribute to the fulfillment of the reaching task. When comparing the action-based risk estimation (A) with the state-based risk estimation (B2b), similar outcomes are observed in the ball and the human environments. In the space environment, however, fewer collisions were recorded when using the state-based risk estimation (B2b).

By using the presented data-based risk estimators, the occurrence of collisions can be reduced significantly throughout the entire training process. This effect is illustrated in Figure 5.4. Specifically, the figure shows the percentage of collision-free episodes when learning a reaching task in the space environment. During the training phase, an episode is terminated after eight seconds or earlier if a collision occurs. When training a task network without action adjustments, the proportion of collision-free episodes slowly increases over the course of training. Conversely, when employing the *action-based risk estimation (A)*, almost all episodes are collision-free right from the start of the training.

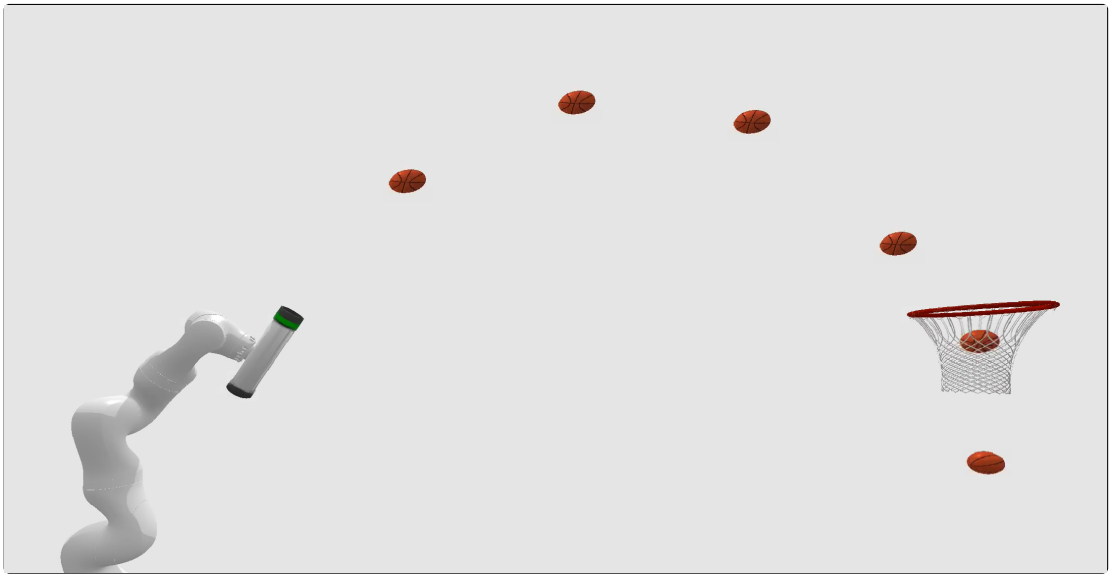


Figure 5.5.: A visualization of the basketball task, in which the robot is trained to place balls in moving basketball hoops.

Table 5.2.: Results obtained when training a task policy for a basketball task using the action-based risk estimation (A). Adapted from [83].

	Random agent		Trained agent	
Environment	Baskets scored per second	Time until collision	Baskets 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

In addition to the reaching task, the presented method is used to learn a task policy for a basketball task illustrated in Figure 5.5. During the training phase, the robot is rewarded for placing a basketball in a basketball hoop. The basketball hoop is moving around the robot on a horizontal circular path, with the radius and the height of the path being randomly selected. Once a ball is placed inside the hoop, a new one is generated at a different location. The angular velocity of each basketball hoop is constant but selected at random. As part of the task-specific state component $s_{t_{Ta}}$, the current position and the angular velocity of the basketball hoop are provided. The task-specific action component $a^T_{t_{Ta}}$ is used to determine the velocity at which a ball should leave a tube connected to the last link of the robot. In contrast to the reaching task, the orientation of the end effector plays an important role in the successful execution of the basketball task. Table 5.2 shows training results for the basketball task obtained using the *action-based risk estimation (A)*. During the course of training, the number of baskets scored per second increases significantly in all environments. While the occurrence of collisions is reduced for both random and trained agents, the average time to a collision is higher for the trained agents. In the space and in the human environment, collisions hardly ever occur when employing the trained agent.

Table 5.3.: Comparison of the action-based risk estimation (A) with a QP-based method for a reaching task in a simplified version of the space environment. Adapted from [83].

Method	Random agent		Trained agent	
	Action adjustment rate	Time until collision	Target points per second	Time until collision
• Action-based (A)	7.1 %	855.2 s	0.82	4325.2 s
• QP-based	12.3 %	35.6 s	0.70	2216.6 s

Table 5.4.: The average ratio between the computation time and the duration of the generated trajectory for different data-based risk estimation methods. The analysis was carried out using an Intel i9-9900K as CPU. Adapted from [83].

Environment	Action-based	State-based		
	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 %

Comparison with a QP-based method The presented *action-based risk estimation (A)* is compared to an alternative method for collision avoidance introduced by Pham et al. [132]. Using the alternative method, unsafe actions are adjusted by solving a mathematical optimization problem formulated as a *quadratic program* (QP). Further details on the method are provided in section 2.2.2.2. From a conceptual perspective, an important difference between the presented concept and the QP-based method is the time horizon taken into consideration when adjusting actions. The optimization problem in the QP-based method focuses on determining a safe action for the current decision step without ensuring recursive feasibility at the subsequent decision steps. As a result, conflicting constraints may arise at a later point in time, preventing the optimizer from finding a safely executable action. In contrast, the presented action generation with a risk network and a backup policy aims to ensure the existence of a safely executable action at the subsequent decision steps. To reduce the occurrence of conflicting constraints, the comparison is carried out using a simplified version of the space environment. In particular, self-collisions and collisions with the table on which the robot is mounted are not taken into consideration. For the QP-based method, the end effector of the robot and the moving obstacles are approximated by spheres.

Table 5.3 shows the results of the comparison conducted for a reaching task. Especially in the case of the random agent, the average time to a collision was significantly higher when the action-based risk estimation was used. In addition, a slightly better task performance was observed with the action-based risk estimation.

Computational effort The computational effort to generate trajectories with the presented risk estimation methods is analyzed in Table 5.4. In all experiments, the average ratio between the computation time and the duration of the generated trajectory was less than 20 %. The moderate computing power requirements of the data-based risk estimators

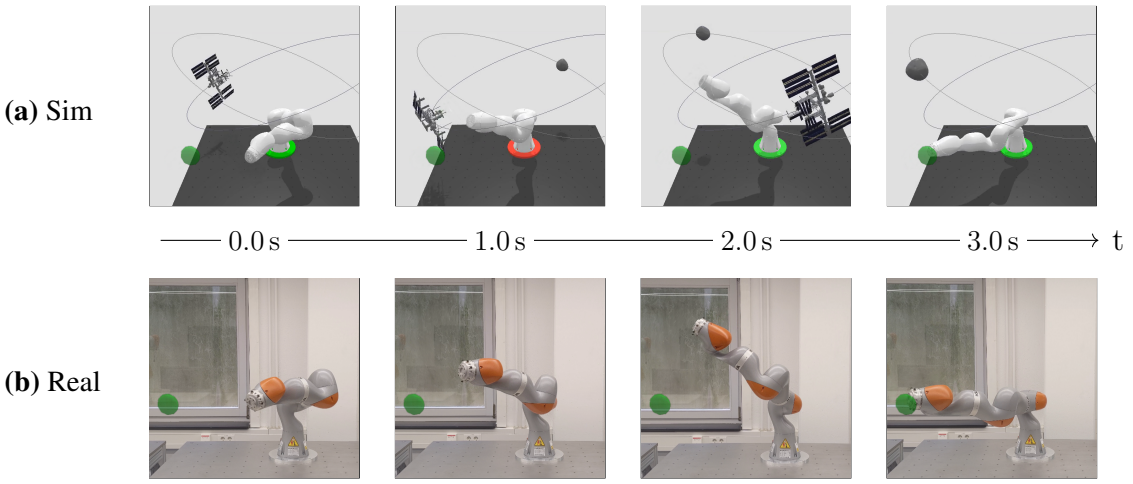


Figure 5.6.: A sim-to-real transfer is conducted for a reaching task in the space environment. At $\mathrm{t} = 1.0\,\mathrm{s}$, the backup policy is executed to prevent a collision with the space station.

are a significant advantage compared to conducting background simulations. In particular, the use of risk networks reduces the time required to train a task policy in a simulator and facilitates the generation of trajectories in real time. The latter is an important prerequisite for a successful sim-to-real transfer.

Sim-to-real transfer Using a real KUKA iiwa robot, a successful sim-to-real transfer is conducted for a reaching task in the space environment. To avoid collisions, the *action-based risk estimation (A)* is utilized. For practical reasons, the moving obstacles are not physically implemented in the real-world setup. Instead, a simulation is conducted to validate that the generated trajectories are collision-free. Figure 5.6 shows an exemplary scenario, in which the task policy tries to move the robot towards a target point illustrated in green. At $\mathrm{t} = 1.0\,\mathrm{s}$, the direct path to the target point is obstructed by a space station that orbits around the robot. For this reason, the backup policy is used, which leads the robot away from the orbit of the space station. At $\mathrm{t} = 2.0\,\mathrm{s}$, the path to the target point is no longer blocked. As a consequence, the execution of the task policy is resumed so that the target point is reached at $\mathrm{t} = 3.0\,\mathrm{s}$.

5.6. Summary

This chapter introduced the use of data-based risk estimators for learning collision-free robot trajectories in environments with moving obstacles. To formalize the learning problem, a Markov decision process was defined. The actions for the Markov decision process were generated using either a task policy or a backup policy. While both policies were trained using model-free RL, their optimization objectives varied. Specifically, the backup policy was trained to produce evasive movements. The objective of the task policy was determined by the respective learning task being pursued. To decide whether to use the task policy or the backup policy, data from prior executions of the backup policy were utilized.

Specifically, the study investigated the use of neural networks to estimate the risk associated with an action. Compared to conducting background simulations in a physics simulator, data-based risk estimators provided the benefit of requiring significantly less computational power. In addition, stochastic environments could be addressed straightforwardly. However, the classification of an action as either *risky* or *harmless* could also be incorrect. Specifically, risky actions classified as harmless could lead to collisions, while harmless actions classified as risky could reduce the learning performance of the task policy. The ratio between these two error types could be influenced by adjusting a so-called risk threshold used for the classification.

The performance of the data-based risk estimators was evaluated in three different environments with moving obstacles. First, it was shown how the selected risk threshold influences the occurrence of collisions when using a random agent as task policy. Subsequently, data-based risk estimators were successfully employed to learn task policies for a reaching task and a basketball task. In contrast to training a task policy without action adjustments, the presented method managed to reduce collisions throughout the entire training process. Compared to a QP-based method, the use of a data-based risk estimator proved to be more effective in preventing collisions. An analysis of the computational effort confirmed that the risk estimation based on neural networks requires only a moderate amount of computing power. By performing a sim-to-real transfer with a KUKA robot, it was shown that trajectories can be generated in real time.

6. Discussion

This chapter delves deeper into the safety techniques presented, emphasizing three important aspects for discussion. First, suggestions for improvement are put forward, for example, to reduce the impact on the learning process when adjusting risky actions. Next, possible applications of the proposed safety techniques are examined. Finally, the focus is placed on key challenges in model-free reinforcement learning and potential strategies to overcome them.

6.1. Opportunities for enhancement

Reducing the impact of action adjustments The safety techniques presented in chapter 4 and chapter 5 make use of a backup policy to adjust risky actions from a task policy. While this approach is effective in avoiding safety violations, it can also reduce the final learning performance, as the actions from the backup policy do not contribute to the fulfillment of the desired task. Therefore, it is desirable to keep the rate of adjusted actions low. In the following, potential strategies to reduce the impact of the backup policy are discussed:

- **Repeated safety checks:** Assuming that an action from the task policy is considered risky, this strategy attempts to find a new action that is close to the original action and passes the safety check. The new action can, for instance, be generated by sampling another action from the task policy or by adding noise to the original one. Ideally, the new action passes the safety check and contributes to the fulfillment of the task. If the safety check fails again, the procedure can be repeated or an action from the backup policy can be executed. However, when using data-based risk estimators, repeated safety checks can increase the probability of classification errors. If background simulations are conducted, the computational effort is a limiting factor.
- **Penalties:** Another strategy to discourage action adjustments is to reduce the reward if the backup policy is used. However, when adding penalties to the reward function, care must be taken to avoid learning an overly risk-averse task policy.
- **Improved backup policy:** Action adjustments can also be reduced by increasing the effectiveness of the backup policy in producing safe backup trajectories. When using a backup policy trained via model-free RL, the performance can be improved by tuning the hyperparameters involved in the training process. The backup policy based on braking trajectories presented in section 4.5 could potentially be enhanced by performing an evasive movement before slowing down the robot joints. Similar as explained in section 4.6, the evasive movement could be generated by a policy trained using model-free RL. However, when performing safety checks based on background simulations, the computational effort increases if an additional evasive movement has to be taken into account.

Safety guarantees in environments with moving obstacles In this thesis, backup policies trained via model-free RL are utilized to maintain safety in environments with moving obstacles. While the presented safety techniques are effective in reducing collisions, they do not provide strict safety guarantees if moving obstacles are present. Under certain conditions, however, it is possible to enhance the methods so that collisions can be completely avoided. In particular, the movement of the obstacles must follow a deterministic pattern and there must be areas in the environment that are free of moving obstacles. In this case, the trajectories generated by the backup policy can be extended by a braking trajectory. When performing background simulations to detect unsafe actions, the safety check is only considered to be passed if the backup trajectories end in an area without moving obstacles. This way, it is possible to ensure that no further collisions can occur, so that the safety conditions introduced in section 4.4 can be satisfied.

Incorporation of further safety constraints While the primary objective of the backup policies used in this thesis is collision avoidance, it is also possible to incorporate further safety constraints. To this end, the safety checks carried out to identify unsafe actions must take additional safety-relevant properties into account. When using a backup policy based on braking trajectories, a prerequisite is that no further safety violations can occur once the robot is stopped. An example of such a constraint would be a limit on the Cartesian velocity of a robot link. Balancing constraints, on the other hand, typically do not satisfy this condition. For instance, a bipedal robot can still fall over even if all robot joints are stopped. One approach to account for balancing constraints is to utilize a backup policy that actively attempts to keep the robot in balance. Similar as in section 4.6, such a backup policy could be trained using model-free RL. When conducting background simulations, the time horizon of the simulations should be selected sufficiently long to bring the robot into a stable position. As explained in chapter 5, the computing power needed for real-time execution can be reduced by employing a data-based risk estimator.

Inclusion of additional sensor data The learning experiments in this thesis are based on the framework of a Markov decision process. Specifically, a state $s \in \mathcal{S}$ serves as the input signal for a policy π. In this work, the state of the robot is described by the kinematic setpoints of the individual robot joints (φ_d, $\dot{\varphi}_d$, $\ddot{\varphi}_d$). It is assumed that the actual values closely follow the setpoints. If this assumption does not hold true, the actual values (φ, $\dot{\varphi}$, $\ddot{\varphi}$) can be added to the state. Furthermore, it might be advantageous to additionally incorporate feedback from torque sensors, especially for tasks involving contact. While sensor data can be easily provided in a simulation environment, care must be taken to avoid widening the sim-to-real gap. In particular, it is important to consider that real sensor data may exhibit a time delay caused by filtering mechanisms and the process of data transmission. Another possible enhancement is the use of visual feedback to describe the state of the environment. Compared to environment-specific state descriptors, visual feedback offers the advantage of being more versatile, making it particularly interesting for policies that need to operate in varying environments. On the other hand, the inclusion of visual feedback may lead to a higher demand for training data and may require additional precautions to avoid performance setbacks when conducting a sim-to-real transfer.

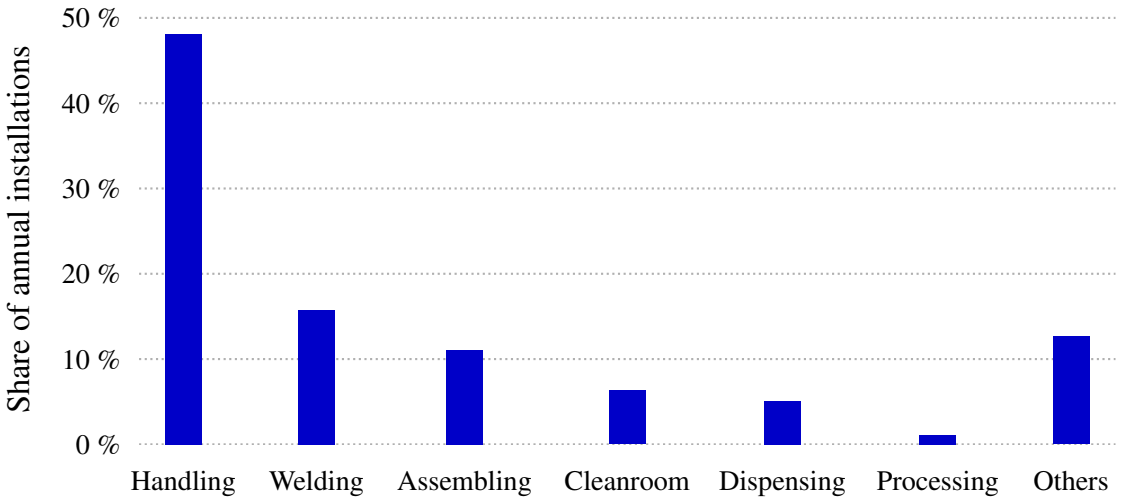


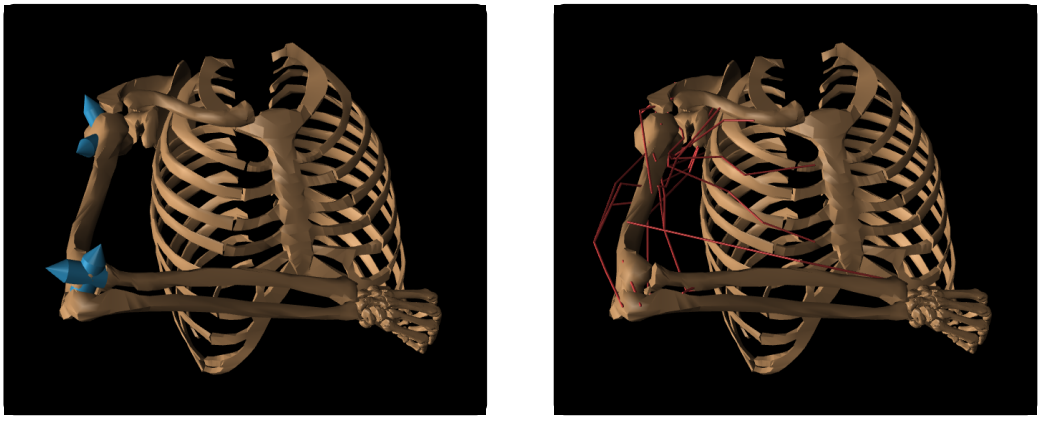
Figure 6.1.: The application areas of industrial robots installed in 2022 according to the *World Robotics Report* of the IFR Statistical Department [118].

6.2. Potential applications

Industrial robots Figure 6.1 shows common application areas of industrial robots as reported in [118]. With a share of almost $50\,\%$, handling operations are the most important application in industrial robotics. According to the classification scheme used in [118], handling operations describe assistant processes for tasks such as palletizing, placing, measuring, inspecting, or testing goods. In these scenarios, the robot is primarily used to transport objects from one location to another. Apart from collision avoidance, the speed of operation is a crucial factor in industrial environments. As the safety techniques proposed in this thesis can be used to learn fast, collision-free trajectories, industrial handling operations represent a promising field of application. The use of data-based optimization techniques might also be beneficial for dispensing processes like powder coating or spray painting [80, 163, 168]. In view of the increasing importance of collaborative robots, the collision-free interaction between humans and robots is another interesting field for future applications.

Humanoid robots In this thesis, the action mapping introduced in chapter 3 is successfully employed for learning experiments with the humanoid robots ARMAR-4 [11] and ARMAR-6 [12]. The safety techniques outlined in chapter 4 and chapter 5 are primarily intended for robots with a stable base like ARMAR-6. From an application perspective, the techniques can be used to facilitate the learning of tasks requiring collision-free coordination of the robot arms. A possible extension option for bipedal robots is the use of a backup policy that actively attempts to keep the robot in balance. Since humanoid robots are expected to operate in diverse and unstructured environments, future applications will likely benefit from visual feedback.

Biologically inspired robots Biologically inspired robots often use actuation mechanisms that differ from traditional robots. The methods presented in this thesis are mainly



(a) Modeling based on revolute joints. **(b)** Modeling based on muscles and tendons.

Figure 6.2.: Two different ways to simulate the motion of a human arm in the physics simulator MuJoCo [171] using an upper limb model presented in [144].

intended for conventional prismatic and revolute robot joints. However, alternative actuation mechanisms may be considered indirectly. Figure 6.2 shows a model of a human arm visualized in the physics simulator MuJoCo [171]. While the actual actuation is based on muscles and tendons, the motion capabilities of the arm can also be approximated using revolute joints and an additional controller that translates joint movements into suitable muscle activities. Based on such an indirect control mechanism, the techniques presented in this thesis may also be applied to enhance the safety of biologically inspired robots.

6.3. Key challenges in model-free reinforcement learning

Although model-free RL has proven to be effective in various scenarios, there are still open research problems that require further attention. Two important aspects are the definition of a suitable reward function and the amount of data required for the training process. In addition, current policies are often limited to a specific task and environment. Figure 6.3 shows how the safety techniques presented in this thesis could be integrated into a future system to train a policy that can be used for different tasks and environments.

Definition of the reward function When employing model-free RL, one difficulty is to specify the reward function for a desired learning task. Providing suitable rewards is particularly challenging if several task objectives need to be balanced against each other. In practice, the reward function is often defined by a human expert, which reduces the autonomy of the learning process. The field of *inverse reinforcement learning* [9] aims to infer the reward function based on demonstrations provided by a teacher. Looking ahead, one potential approach for learning a policy capable of performing different tasks could involve utilizing a data-based reward estimator trained on a large dataset of human demonstrations. Figure 6.3 shows how such a reward estimator could be integrated into a future system that uses either demonstrations or a textual description to specify the desired learning task.

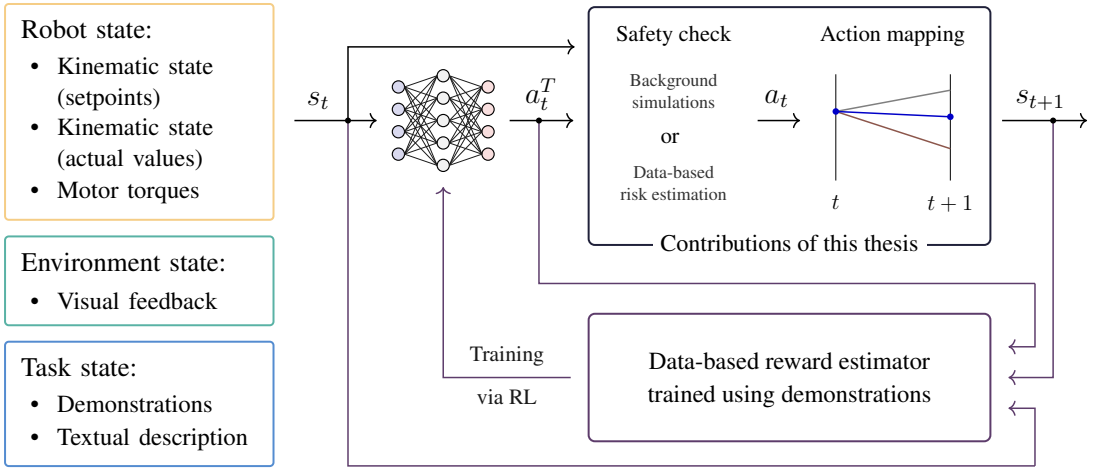


Figure 6.3.: A potential future architecture to train a generalist policy using the contributions presented in this thesis. In this architecture, a state $s \in \mathcal{S}$ additionally includes visual feedback of the environment and a specification of the desired task, e.g., based on a textual description. Rewards are assigned by a data-based estimator trained using a large amount of task demonstrations.

Amount of training data required Another challenge when utilizing model-free RL is the amount of data required for training a policy. In practice, the training process is often carried out in a simulator so that the data generation can be scaled up through parallelization. However, even when using a simulator, the tuning of hyperparameters is often restricted by limited computing resources. Another challenge when utilizing simulated data is the sim-to-real gap arising from inaccuracies in the physics simulation. The sim-to-real gap can be narrowed by improving the accuracy of the physics simulation or by employing domain randomization techniques. Alternatively, a policy trained in simulation can be fine-tuned by collecting additional data with real robots.

Learning a generalist policy Today, model-free RL is often used to train a policy for a single task in a specific environment. In the future, however, robots are expected to perform a variety of tasks in different environments. Recent studies on generalist robotic agents include approaches like Gato [138], RT-2 [194], and RoboCat [22]. Figure 6.3 shows a potential way to incorporate the safety techniques presented in this thesis into a larger system so that different tasks and environments can be addressed. While learning a policy for diverse scenarios further increases the amount of training data required, this approach holds promise in enabling future robots that are both versatile and safe in operation.

7. Conclusion

This thesis investigated the problem of generating safely executable robot trajectories using model-free *reinforcement learning* (RL). Recent advances in artificial intelligence have rendered model-free RL a highly promising tool for the development of autonomous, self-learning robots that do not rely on predefined environmental models. However, when learning robot trajectories in unknown environments, a major difficulty is to ensure that neither the robot nor its surroundings are damaged.

In model-free RL, learning problems are typically formalized using the mathematical framework of a *Markov decision process* (MDP). The objective of the learning process is to find a policy, which maps states to actions so that a sum of rewards is maximized. In order to find the policy, a training phase is conducted, in which the environment is explored based on the principle of trial and error. When utilizing model-free RL to generate robot trajectories in real time, three safety-related challenges emerge:

- First, it is usually not possible to predetermine future robot movements as they depend on unknown future states.
- Second, since random actions are selected at the beginning of the training phase, potential safety measures must be able to deal with arbitrary actions.
- Third, the computational effort of potential safety measures must be limited in order to allow real-time execution.

In this thesis, the challenges mentioned above were addressed by developing, implementing, and evaluating methods to effectively avoid safety violations during and after the training phase of a policy. While particular emphasis was placed on considering kinematic joint constraints and preventing collisions, the methods were designed to cover a broad spectrum of safety constraints.

In a first step, a comprehensive literature survey on the topic of *safe reinforcement learning* was conducted. Existing techniques were categorized into *practical approaches* and *theoretical approaches*. The methods belonging to the former category aimed at addressing safety issues from a practical perspective, typically without a thorough mathematical analysis of the problem. As a drawback, the resulting safety measures were often task-specific or required additional engineering effort. The theoretical approaches, on the other hand, were further divided into two sub-categories. In particular, a distinction was made between those approaches that adjust the optimization objective of the learning problem and those approaches that adjust actions that may pose a safety risk. Approaches that modified the optimization objective had the disadvantage of limited effectiveness at the beginning of the training phase. Conversely, approaches involving the adjustment of unsafe actions did not face this restriction. However, existing techniques were either tailored to specific applications, required a model of the system dynamics, or did not provide strict safety guarantees.

An important research problem that was not sufficiently covered by existing methods was the learning of robot trajectories subject to kinematic joint constraints. Especially when generating fast robot movements, compliance with kinematic constraints is essential to prevent damage to the robot joints. Therefore, the first research question covered in this thesis was how actions can be translated into robot trajectories that adhere to position, velocity, acceleration, and jerk limits specified for each robot joint. The basic idea was to design an action mapping ensuring that every possible action leads to a valid trajectory. To this end, the concept of an upper and a lower trajectory was introduced. Both of these trajectories were required to satisfy the kinematic constraints over an infinite time horizon. Based on this prerequisite, it was shown that every possible action can be mapped to an intermediate trajectory that also complies with the kinematic constraints. As a result, neither the above-mentioned uncertainty about future states nor the random selection of actions at the beginning of a training phase posed a safety problem. For this reason, the action mapping served as the basis for all learning experiments carried out in this thesis.

In order to consider further safety constraints, the concept of background simulations was established in a follow-up study. To this end, a backup policy was introduced as a complement to the normal task policy. The purpose of the backup policy was to replace unsafe actions generated by the task policy. To determine whether an action from the task policy was considered safe, a background simulation was carried out in a physics simulator. In order to ensure that the backup policy could always be executed safely, certain safety conditions had to be fulfilled. Most importantly, the time horizon of the background simulations had to be sufficiently long and the environment had to be predictable. For environments without moving obstacles, a backup policy based on braking trajectories was proposed. Using this backup policy, collisions and torque limit violations could be effectively prevented when learning movements for robots with a stable base. Next, the concept was extended to environments with moving obstacles. For this purpose, a backup policy was trained to actively avoid collisions using model-free RL. In this context, two additional challenges were identified: First, the computational effort for the background simulations increased significantly when moving obstacles were taken into account. Second, the predictability of the environment was limited when obstacles moved in a non-deterministic way. Both of these challenges could be addressed by introducing data-based risk estimators.

The key idea of the data-based risk estimators was to predict the outcome of a computationally intensive background simulation by leveraging previously collected data. To this end, neural networks were trained using supervised learning. Due to the low resource demands of the neural networks, the computational effort for real-time trajectory generation could be significantly reduced. In addition, it became possible to estimate the risk of safety violations in environments with stochastic behavior. The effectiveness of the data-based risk estimators was evaluated by avoiding collisions in environments with moving obstacles. More specifically, the evaluation included a human-robot interaction scenario, in which the robot had to anticipate the movements of a human and a ball environment, in which the robot had to keep moving to avoid collisions. The results showed that the use of a backup policy and a risk estimator was effective in reducing collisions throughout the entire training process of a task policy. In addition, the evaluation revealed a trade-off between exploring the environment and avoiding collisions, which could be controlled by adjusting a threshold used for the risk classification of actions.

In summary, the research carried out in this thesis led to three key contributions to the field of safe reinforcement learning:

- **Action mapping considering kinematic joint constraints**:

  This contribution outlined a method for translating actions from a Markov decision process into robot trajectories that strictly adhere to kinematic joint constraints. Specifically, constraints on the position, velocity, acceleration, and jerk of each robot joint were considered. Using the proposed action mapping, policies were successfully trained to track reference paths and to adjust reference trajectories for a ball-on-plate task. In addition, the action mapping served as the basis for the trajectory generation presented in the following two contributions.

- **Safety assessment using simulated braking trajectories**:

  In this contribution, a backup policy based on braking trajectories was introduced. The backup policy served as a safety measure for robots with a stable base in environments without moving obstacles. By conducting background simulations in a physics simulator, collisions and torque limit violations could be strictly prevented. The evaluation was carried out by successfully learning reaching tasks with up to three industrial robots and a humanoid robot.

- **Risk estimation based on learned backup behaviors**:

  As part of this contribution, backup policies were trained to actively avoid safety violations. Subsequently, the backup policies were used to generate training data for a risk estimator. Compared to using braking trajectories as a backup policy, learned backup behaviors offered the advantage of being more flexible with regard to the safety constraints being considered. In this work, the proposed risk estimation was evaluated for collision avoidance in environments with moving obstacles. As part of the evaluation, policies for a reaching task and a basketball task were successfully trained. Using data-based risk estimators, compliance with the specified safety constraints was encouraged but not strictly guaranteed.

Each of the contributions underwent a thorough and systematic evaluation process. To demonstrate that safely executable trajectories can be generated in real time, policies trained in simulation were successfully transferred to a real industrial robot.

The discussion of this work highlighted opportunities for improvement, potential application areas, and directions for future research. Finally, it was outlined how the techniques presented in this thesis can contribute to the development of autonomous robots that are both flexible and safe in operation.

Appendix

A. Safety properties of the proposed action mapping

In chapter 3, an action mapping is introduced to generate trajectories that satisfy the kinematic constraints given by the equations (3.1) - (3.4). Based on an action a_t at time step t, the action mapping produces a trajectory from time step t to time step $t+1$. In order to ensure that the kinematic constraints are always satisfied, the action mapping needs to fulfill the following two safety properties:

- The generated trajectory from t to $t+1$ must satisfy the kinematic constraints.
- There must be at least one feasible way to continue the trajectory at time step $t+1$.

In the following, it is shown that these properties can be fulfilled when providing an upper and a lower trajectory, which both fulfill the kinematic constraints (3.1) - (3.4). Contrary to t, the variable t indicates a continuous time. The course of the jerk of a one-dimensional lower and upper trajectory is denoted by $\mathrm{j_L(t)}$ and $\mathrm{j_H(t)}$, respectively. Likewise, the course of acceleration is labeled $\mathrm{a_L(t)}$ and $\mathrm{a_H(t)}$. The initial acceleration at $\mathrm{t} = 0$ is labeled $\mathrm{a_0}$. With the proposed action mapping, an intermediate trajectory is computed that lies between the lower and upper trajectory. The jerk and acceleration of the intermediate trajectory are referred to as $\mathrm{j_M(t)}$ and $\mathrm{a_M(t)}$, respectively. Each action a is composed of a scalar $\in [-1, 1]$ per robot joint. In the following, the scalar for the one-dimensional case is labeled m with $m \in [-1, 1]$. The jerk of the intermediate trajectory is computed as follows:

$$\mathrm{j_M(t)} = \mathrm{j_L(t)} + \frac{1+m}{2} \cdot (\mathrm{j_H(t)} - \mathrm{j_L(t)}) \tag{7.1}$$

With $m \in [-1, 1]$, the following relation is satisfied:

$$0 \leq \frac{1+m}{2} \leq 1 \tag{7.2}$$

As a prerequisite, $\mathrm{j_L(t)}$ and $\mathrm{j_H(t)}$ comply with the jerk constraint (3.4):

$$\mathrm{j_{min}} \leq \mathrm{j_L(t)} \leq \mathrm{j_{max}} \tag{7.3}$$

$$\mathrm{j_{min}} \leq \mathrm{j_H(t)} \leq \mathrm{j_{max}} \tag{7.4}$$

Assuming $\mathrm{j_H(t)} \geq \mathrm{j_L(t)}$, it follows from (7.1) to (7.4) that:

$$\mathrm{j_{min}} \leq \mathrm{j_L(t)} \leq \mathrm{j_M(t)} \leq \mathrm{j_H(t)} \leq \mathrm{j_{max}} \tag{7.5}$$

Similarly, the following relation applies for $\mathrm{j_L(t)} \geq \mathrm{j_H(t)}$:

$$\mathrm{j_{min}} \leq \mathrm{j_H(t)} \leq \mathrm{j_M(t)} \leq \mathrm{j_L(t)} \leq \mathrm{j_{max}} \tag{7.6}$$

As a result, the jerk of the intermediate trajectory satisfies the jerk constraints in any case:

$$\mathrm{j_{min}} \leq \mathrm{j_M(t)} \leq \mathrm{j_{max}} \tag{7.7}$$

The acceleration $\mathrm{a_M(t)}$ of the intermediate trajectory can be computed by integrating the intermediate jerk $\mathrm{j_M(t)}$:

$$\mathrm{a_M(t)} = \mathrm{a_0} + \int_0^{\mathrm{t}} \mathrm{j_M(t)}\ d\mathrm{t} \tag{7.8}$$

The acceleration of the lower and upper trajectory can be calculated analogously:

$$\mathrm{a_L(t)} = \mathrm{a_0} + \int_0^{\mathrm{t}} \mathrm{j_L(t)}\ d\mathrm{t} \tag{7.9}$$

$$\mathrm{a_H(t)} = \mathrm{a_0} + \int_0^{\mathrm{t}} \mathrm{j_H(t)}\ d\mathrm{t} \tag{7.10}$$

Using (7.1), (7.9) and (7.10), equation (7.8) can be rewritten as follows:

$$\begin{aligned}
\mathrm{a_M(t)} &= \mathrm{a_0} + \int_0^{\mathrm{t}} \mathrm{j_M(t)}\ d\mathrm{t} \\
&= \mathrm{a_0} + \int_0^{\mathrm{t}} \left(\mathrm{j_L(t)} + \frac{1+m}{2} \cdot (\mathrm{j_H(t)} - \mathrm{j_L(t)}) \right)\ d\mathrm{t} \\
&= \mathrm{a_0} + \int_0^{\mathrm{t}} \mathrm{j_L(t)}\ d\mathrm{t} + \frac{1+m}{2} \cdot \left(\int_0^{\mathrm{t}} \mathrm{j_H(t)}\ d\mathrm{t} - \int_0^{\mathrm{t}} \mathrm{j_L(t)}\ d\mathrm{t} \right) \\
&= \mathrm{a_L(t)} + \frac{1+m}{2} \cdot (\mathrm{a_H(t)} - \mathrm{a_L(t)})
\end{aligned} \tag{7.11}$$

For the intermediate velocity $\mathrm{v_M(t)}$ and the intermediate position $\mathrm{p_M(t)}$, the same reasoningreason can be applied to show that:

$$\mathrm{v_M(t)} = \mathrm{v_L(t)} + \frac{1+m}{2} \cdot (\mathrm{v_H(t)} - \mathrm{v_L(t)}) \tag{7.12}$$

$$\mathrm{p_M(t)} = \mathrm{p_L(t)} + \frac{1+m}{2} \cdot (\mathrm{p_H(t)} - \mathrm{p_L(t)}) \tag{7.13}$$

It can be seen that the equations (7.11), (7.12), and (7.13) correspond to the calculation of the intermediate jerk in (7.1). Consequently, the same reasoning used to derive that $\mathrm{j_M(t)}$ complies with the jerk constraints (7.7) can also be used to show that $\mathrm{a_M(t)}$, $\mathrm{v_M(t)}$, and $\mathrm{p_M(t)}$ satisfy the following kinematic constraints:

$$\mathrm{a_{min}} \leq \mathrm{a_M(t)} \leq \mathrm{a_{max}} \tag{7.14}$$

$$\mathrm{v_{min}} \leq \mathrm{v_M(t)} \leq \mathrm{v_{max}} \tag{7.15}$$

$$\mathrm{p_{min}} \leq \mathrm{p_M(t)} \leq \mathrm{p_{max}} \tag{7.16}$$

According to (7.7) and (7.14) - (7.16), the desired kinematic constraints are fulfilled when generating an intermediate trajectory from the discrete time step t to $t+1$. At time step $t+1$, a new upper and lower trajectory are computed. However, as shown in Figure A.1, the trajectory could also be continued by keeping the same relative distance to the initial upper and lower trajectory as during the time interval from t to $t+1$. For this reason, the second safety property, which states that there must be at least one feasible way to continue the trajectory at time step $t+1$, is also satisfied.

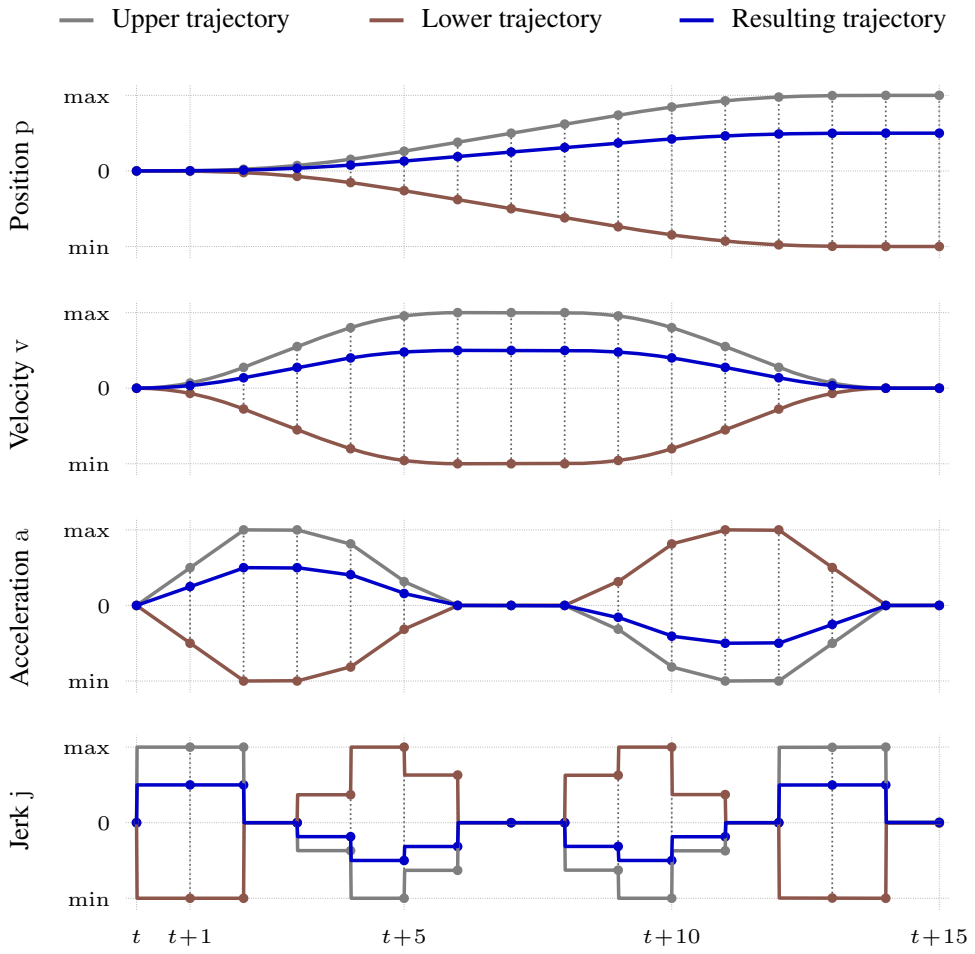


Figure A.1.: The figure shows the resulting trajectory when selecting an action $a_t = 0.5$ at time step t and keeping the specified relative distance to the initial upper and lower trajectory for all subsequent time intervals. Under normal conditions, a new upper and lower trajectory are computed at $t + 1$. The blue trajectory shows that there is at least one valid way to continue the trajectory at $t + 1$. Assuming that no valid upper and lower trajectory is found at $t + 1$, the blue trajectory could serve as a fallback for both of them.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015.

[2] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained Policy Optimization," in *International Conference on Machine Learning*. PMLR, 2017, pp. 22–31.

[3] A. K. Akametalu, J. F. Fisac, J. H. Gillula, S. Kaynama, M. N. Zeilinger, and C. J. Tomlin, "Reachability-Based Safe Learning with Gaussian Processes," in *53rd IEEE Conference on Decision and Control*. IEEE, 2014, pp. 1424–1431.

[4] E. Altman, *Constrained Markov Decision Processes*. CRC Press, 1999.

[5] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control Barrier Function Based Quadratic Programs for Safety Critical Systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3861–3876, 2016.

[6] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control Barrier Functions: Theory and Applications," in *2019 18th European Control Conference (ECC)*. IEEE, 2019, pp. 3420–3431.

[7] B. Amos and J. Z. Kolter, "OptNet: Differentiable Optimization as a Layer in Neural Networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 136–145.

[8] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki *et al.*, "Learning Dexterous In-Hand Manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[9] S. Arora and P. Doshi, "A Survey of Inverse Reinforcement Learning: Challenges, Methods and Progress," *Artificial Intelligence*, vol. 297, p. 103500, 2021.

[10] T. Asfour, P. Azad, F. Gyarfas, and R. Dillmann, "Imitation Learning of Dual-Arm Manipulation Tasks in Humanoid Robots," *International Journal of Humanoid Robotics*, vol. 5, no. 2, pp. 183–202, 2008.

[11] T. Asfour, J. Schill, H. Peters, C. Klas, J. Bücker, C. Sander *et al.*, "ARMAR-4: A 63 DOF Torque Controlled Humanoid Robot," in *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE, 2013, pp. 390–396.

[12] T. Asfour, L. Kaul, M. Wächter, S. Ottenhaus, P. Weiner, S. Rader *et al.*, "ARMAR-6: A Collaborative Humanoid Robot for Industrial Environments," in *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2018, pp. 447–454.

[13] K. J. Astrom *et al.*, "Optimal Control of Markov Processes with Incomplete State Information," *Journal of Mathematical Analysis and Applications*, vol. 10, no. 1, pp. 174–205, 1965.

[14] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin, "Hamilton-Jacobi Reachability: A Brief Overview and Recent Advances," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 2242–2253.

[15] R. Bellman, "Dynamic Programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[16] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for Hyper-Parameter Optimization," *Advances in Neural Information Processing Systems*, vol. 24, 2011.

[17] 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.

[18] L. Berscheid and T. Kröger, "Jerk-Limited Real-Time Trajectory Generation with Arbitrary Target States," *Robotics: Science and Systems XVII*, 2021.

[19] H. Bharadhwaj, A. Kumar, N. Rhinehart, S. Levine, F. Shkurti, and A. Garg, "Conservative Safety Critics for Exploration," in *9th International Conference on Learning Representations, ICLR 2021*, 2021.

[20] C. M. Bishop, "Mixture Density Networks," Aston University, Tech. Rep., 1994.

[21] S. Bohez, A. Abdolmaleki, M. Neunert, J. Buchli, N. Heess, and R. Hadsell, "Value Constrained Model-Free Continuous Control," *arXiv preprint arXiv:1902.04623*, 2019.

[22] K. Bousmalis, G. Vezzani, D. Rao, C. M. Devin, A. X. Lee, M. B. Villalonga *et al.*, "RoboCat: A Self-Improving Generalist Agent for Robotic Manipulation," *Transactions on Machine Learning Research*, 2024.

[23] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.

[24] X. Broquere, D. Sidobre, and I. Herrera-Aguilar, "Soft Motion Trajectory Planner for Service Manipulator Robot," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2008, pp. 2808–2813.

[25] 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.

[26] S. Calinon and A. Billard, "Stochastic Gesture Production and Recognition Model for a Humanoid Robot," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3. IEEE, 2004, pp. 2769–2774.

[27] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8973–8979.

[28] 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. 1, 2019, pp. 3387–3395.

[29] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2014, p. 1724.

[30] E. Coumans and Y. Bai, "PyBullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning," 2016.

[31] 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.

[32] J. Dao, K. Green, H. Duan, A. Fern, and J. Hurst, "Sim-to-Real Learning for Bipedal Locomotion Under Unsensed Dynamic Loads," in *2022 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 10 449–10 455.

[33] M. De Lasa, I. Mordatch, and A. Hertzmann, "Feature-Based Locomotion Controllers," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 4, pp. 1–10, 2010.

[34] A. Draeger, S. Engell, and H. Ranke, "Model Predictive Control Using Neural Networks," *IEEE Control Systems Magazine*, vol. 15, no. 5, pp. 61–66, 1995.

[35] B. Eysenbach, S. Gu, J. Ibarz, and S. Levine, "Leave no Trace: Learning to Reset for Safe and Autonomous Reinforcement Learning," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, Canada*, 2018.

[36] B. Faverjon and P. Tournassoud, "A Local Based Approach for Path Planning of Manipulators with a High Number of Degrees of Freedom," in *1987 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 4. IEEE, 1987, pp. 1152–1159.

[37] 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.

[38] P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs *et al.*, "Implicit Behavioral Cloning," in *Conference on Robot Learning*. PMLR, 2022, pp. 158–168.

[39] P. Florence, L. Manuelli, and R. Tedrake, "Self-Supervised Correspondence in Visuomotor Policy Learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 492–499, 2019.

[40] T. Fraichard, "A Short Paper About Motion Safety," in *2007 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2007, pp. 1140–1145.

[41] R. Frisch, "The Logarithmic Potential Method of Convex Programming," *Memorandum, University Institute of Economics, Oslo*, vol. 5, no. 6, 1955.

[42] A. Gams, A. Ude, J. Morimoto *et al.*, "Deep Encoder-Decoder Networks for Mapping Raw Images to Dynamic Movement Primitives," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 5863–5868.

[43] J. Garcia and F. Fernández, "Safe Exploration of State and Action Spaces in Reinforcement Learning," *Journal of Artificial Intelligence Research*, vol. 45, pp. 515–564, 2012.

[44] J. García and F. Fernández, "A Comprehensive Survey on Safe Reinforcement Learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[45] C. Gaskett, "Reinforcement Learning Under Circumstances Beyond Its Control," *International Conference on Computational Intelligence for Modelling, Control and Automation (CIMCA)*, 2003.

[46] A. Gasparetto and V. Zanotto, "A Technique for Time-Jerk Optimal Planning of Robot Trajectories," *Robotics and Computer-Integrated Manufacturing*, vol. 24, no. 3, pp. 415–426, 2008.

[47] P. Geibel and F. Wysotzki, "Risk-Sensitive Reinforcement Learning Applied to Control Under Constraints," *Journal of Artificial Intelligence Research*, vol. 24, pp. 81–108, 2005.

[48] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," *SIAM review*, vol. 47, no. 1, pp. 99–131, 2005.

[49] J. H. Gillulay and C. J. Tomlin, "Guaranteed Safe Online Learning of a Bounded System," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2011, pp. 2979–2984.

[50] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT press, 2016.

[51] 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.

[52] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1861–1870.

[53] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan *et al.*, "Soft Actor-Critic Algorithms and Applications," *arXiv preprint arXiv:1812.05905*, 2018.

[54] A. Hans, D. Schneegaß, A. M. Schäfer, and S. Udluft, "Safe Exploration for Reinforcement Learning," in *European Symposium on Artificial Neural Networks (ESANN)*, 2008, pp. 143–148.

[55] R. Haschke, E. Weitnauer, and H. Ritter, "On-Line Planning of Time-Optimal, Jerk-Limited Trajectories," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2008, pp. 3248–3253.

[56] M. J. Hausknecht and P. Stone, "Deep Reinforcement Learning in Parameterized Action Space," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico*, 2016.

[57] M. Heger, "Consideration of Risk in Reinforcement Learning," in *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 105–111.

[58] M. Hersch, F. Guenter, S. Calinon, and A. Billard, "Dynamical System Modulation for Robot Learning via Kinesthetic Demonstrations," *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1463–1467, 2008.

[59] A. Herzog, N. Rotella, S. Mason, F. Grimminger, S. Schaal, and L. Righetti, "Momentum Control with Hierarchical Inverse Dynamics on a Torque-Controlled Humanoid," *Autonomous Robots*, vol. 40, pp. 473–491, 2016.

[60] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot *et al.*, "Deep Q-Learning from Demonstrations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[61] L. Hewing, J. Kabzan, and M. N. Zeilinger, "Cautious Model Predictive Control Using Gaussian Process Regression," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2736–2743, 2019.

[62] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-Based Model Predictive Control: Toward Safe Learning in Control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 269–296, 2020.

[63] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[64] D. Hoeller, F. Farshidian, and M. Hutter, "Deep Value Model Predictive Control," in *Conference on Robot Learning*. PMLR, 2020, pp. 990–1004.

[65] A. Ijspeert, J. Nakanishi, and S. Schaal, "Learning Attractor Landscapes for Learning Motor Primitives," *Advances in Neural Information Processing Systems*, vol. 15, 2002.

[66] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors," *Neural Computation*, vol. 25, no. 2, pp. 328–373, 2013.

[67] S. James, A. J. Davison, and E. Johns, "Transferring End-to-End Visuomotor Control from Simulation to Real World for a Multi-Stage Task," in *Conference on Robot Learning*. PMLR, 2017, pp. 334–343.

[68] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz *et al.*, "Sim-to-Real via Sim-to-Sim: Data-Efficient Robotic Grasping via Randomized-to-Canonical Adaptation Networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 627–12 637.

[69] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang *et al.*, "Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation," in *Conference on Robot Learning*. PMLR, 2018, pp. 651–673.

[70] R. E. Kalman *et al.*, "Contributions to the Theory of Optimal Control," *Boletín de la Sociedad Matemática Mexicana*, vol. 5, no. 2, pp. 102–119, 1960.

[71] F. Kanehiro, F. Lamiraux, O. Kanoun, E. Yoshida, and J.-P. Laumond, "A Local Collision Avoidance Method for Non-strictly Convex Polyhedra," *Proceedings of Robotics: Science and Systems IV*, p. 33, 2008.

[72] M. Kaspar, J. D. M. Osorio, and J. Bock, "Sim2real Transfer for Reinforcement Learning Without Dynamics Randomization," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 4383–4388.

[73] M. Khadiv, A. Herzog, S. A. A. Moosavian, and L. Righetti, "Walking Control Based on Step Timing Adaptation," *IEEE Transactions on Robotics*, vol. 36, no. 3, pp. 629–643, 2020.

[74] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[75] O. Khatib, "A Unified Approach for Motion and Force Control of Robot Manipulators: The Operational Space Formulation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.

[76] J. 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.

[77] J. 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.

[78] J. Kiemel and T. Kröger, "Learning Time-Optimized Path Tracking With or Without Sensory Feedback," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 4024–4031.

[79] J. Kiemel and T. Kröger, "Jerk-Limited Traversal of One-Dimensional Paths and Its Application to Multi-Dimensional Path Tracking," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024.

[80] J. Kiemel, P. Yang, P. Meißner, and T. Kröger, "PaintRL: Coverage Path Planning for Industrial Spray Painting with Reinforcement Learning," in *RSS Workshop on Closing the Reality Gap in Sim2real Transfer for Robotic Manipulation*, 2019.

[81] J. Kiemel, P. Meißner, and T. Kröger, "TrueRMA: Learning Fast and Smooth Robot Trajectories with Recursive Midpoint Adaptations in Cartesian Space," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4225–4231.

[82] J. 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, pp. 5387–5394.

[83] J. Kiemel, L. Righetti, T. Kröger, and T. Asfour, "Safe Reinforcement Learning of Robot Trajectories in the Presence of Moving Obstacles," *IEEE Robotics and Automation Letters*, vol. 9, no. 12, pp. 11 353–11 360, 2024.

[84] 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.

[85] V. Konda and J. Tsitsiklis, "Actor-Critic Algorithms," *Advances in Neural Information Processing Systems*, vol. 12, 1999.

[86] T. Kröger, "Opening the Door to New Sensor-Based Robot Applications – The Reflexxes Motion Libraries," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 1–4.

[87] T. Kröger and F. M. Wahl, "Online Trajectory Generation: Basic Concepts for Instantaneous Reactions to Unforeseen Events," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 94–111, 2009.

[88] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[89] 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.

[90] M. A. Lee, Y. Zhu, K. Srinivasan, P. Shah, S. Savarese, L. Fei-Fei *et al.*, "Making Sense of Vision and Touch: Self-Supervised Learning of Multimodal Representations for Contact-Rich Tasks," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8943–8950.

[91] S. Levine and V. Koltun, "Guided Policy Search," in *International Conference on Machine Learning*. PMLR, 2013, pp. 1–9.

[92] 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.

[93] 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.

[94] W. Li and E. Todorov, "Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems," in *First International Conference on Informatics in Control, Automation and Robotics*, vol. 2. SciTePress, 2004, pp. 222–229.

[95] Z. Li, X. Cheng, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath, "Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 2811–2817.

[96] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg *et al.*, "RLlib: Abstractions for Distributed Reinforcement Learning," in *International Conference on Machine Learning*. PMLR, 2018, pp. 3053–3062.

[97] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa *et al.*, "Continuous Control with Deep Reinforcement Learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico*, 2016.

[98] 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.

[99] Y. Liu, A. Halev, and X. Liu, "Policy Learning with Constraints in Model-Free Reinforcement Learning: A Survey," in *The 30th International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[100] J. Luh, M. Walker, and R. Paul, "Resolved-Acceleration Control of Mechanical Manipulators," *IEEE Transactions on Automatic Control*, vol. 25, no. 3, pp. 468–474, 1980.

[101] K. M. Lynch and F. C. Park, *Modern Robotics*. Cambridge University Press, 2017.

[102] S. Macfarlane and E. A. Croft, "Jerk-Bounded Manipulator Trajectory Planning: Design for Real-Time Applications," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 1, pp. 42–52, 2003.

[103] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin *et al.*, "Isaac Gym: High Performance GPU Based Physics Simulation For Robot Learning," in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, vol. 1. Curran, 2021.

[104] S. Mannor and J. N. Tsitsiklis, "Mean-Variance Optimization in Markov Decision Processes," in *International Conference on Machine Learning*. PMLR, 2011, p. 177–184.

[105] N. Mansard, O. Stasse, P. Evrard, and A. Kheddar, "A Versatile Generalized Inverted Kinematics Implementation for Collaborative Working Humanoid Robots: The Stack of Tasks," in *International Conference on Advanced Robotics (ICAR)*, 2009, p. 119.

[106] H. Markowitz, "Portfolio Selection," *The Journal of Finance*, vol. 7, no. 1, pp. 77–91, 1952.

[107] C. Mastalli, W. Merkt, J. Marti-Saumell, H. Ferrolho, J. Solà, N. Mansard, and S. Vijayakumar, "A Feasibility-Driven Approach to Control-Limited DDP," *Autonomous Robots*, vol. 46, no. 8, pp. 985–1005, 2022.

[108] J. Mattmüller and D. Gisler, "Calculating a near Time-Optimal Jerk-Constrained Trajectory Along a Specified Smooth Path," *The International Journal of Advanced Manufacturing Technology*, vol. 45, pp. 1007–1016, 2009.

[109] D. Mayne, "A Second-Order Gradient Method for Determining Optimal Trajectories of Non-linear Discrete-Time Systems," *International Journal of Control*, vol. 3, no. 1, pp. 85–95, 1966.

[110] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, "Constrained Model Predictive Control: Stability and Optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[111] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin *et al.*, "Sympy: Symbolic Computing in Python," *PeerJ Computer Science*, vol. 3, p. e103, 2017.

[112] H. Miyamoto and M. Kawato, "A Tennis Serve and Upswing Learning Robot Based on Bi-directional Theory," *Neural Networks*, vol. 11, no. 7-8, pp. 1331–1344, 1998.

[113] 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.

[114] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare *et al.*, "Human-Level Control Through Deep Reinforcement Learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[115] T. M. Moerland, J. Broekens, A. Plaat, C. M. Jonker *et al.*, "Model-Based Reinforcement Learning: A Survey," *Foundations and Trends in Machine Learning*, vol. 16, no. 1, pp. 1–118, 2023.

[116] G. E. Monahan, "State of the Art – a Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms," *Management Science*, vol. 28, no. 1, pp. 1–16, 1982.

[117] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang *et al.*, "Ray: A Distributed Framework for Emerging AI Applications," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 561–577.

[118] C. Müller, "World Robotics 2023 – Industrial Robots, IFR Statistical Department, VDMA Services GmbH, Frankfurt am Main, Germany," 2023.

[119] Y. Narang, K. Storey, I. Akinola, M. Macklin, P. Reist, L. Wawrzyniak *et al.*, "Factory: Fast Contact for Robotic Assembly," in *Proceedings of Robotics: Science and Systems*, New York City, NY, USA, 2022.

[120] C. J. Ostafew, A. P. Schoellig, T. D. Barfoot, and J. Collier, "Learning-Based Nonlinear Model Predictive Control to Improve Vision-Based Mobile Robot Path Tracking," *Journal of Field Robotics*, vol. 33, no. 1, pp. 133–152, 2016.

[121] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic Movement Primitives," *Advances in Neural Information Processing Systems*, vol. 26, 2013.

[122] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Using Probabilistic Movement Primitives in Robotics," *Autonomous Robots*, vol. 42, pp. 529–551, 2018.

[123] D.-H. Park, H. Hoffmann, P. Pastor, and S. Schaal, "Movement Reproduction and Obstacle Avoidance with Dynamic Movement Primitives and Potential Fields," in *2008 IEEE-RAS 8th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2008, pp. 91–98.

[124] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and Generalization of Motor Skills by Learning from Demonstration," in *2009 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2009, pp. 763–768.

[125] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[126] R. P. C. Paul, "Modelling, Trajectory Calculation and Servoing of a Computer Controlled Arm," Ph.D. dissertation, Stanford University, 1972.

[127] 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.

[128] 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. 3803–3810.

[129] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine, "Learning Agile Robotic Locomotion Skills by Imitating Animals," *Robotics: Science and Systems (RSS)*, 2020.

[130] J. Peters and S. Schaal, "Reinforcement Learning of Motor Skills with Policy Gradients," *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008.

[131] H. Pham and Q.-C. Pham, "A New Approach to Time-Optimal Path Parameterization Based on Reachability Analysis," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 645–659, 2018.

[132] 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.

[133] S. Piche, B. Sayyar-Rodsari, D. Johnson, and M. Gerules, "Nonlinear Model Predictive Control Using Neural Networks," *IEEE Control Systems Magazine*, vol. 20, no. 3, pp. 53–62, 2000.

[134] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust Adversarial Reinforcement Learning," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2817–2826.

[135] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, "Asymmetric Actor Critic for Image-Based Robot Learning," in *14th Robotics: Science and Systems, RSS 2018*. MIT Press Journals, 2018.

[136] M. Raibert, "Manipulator Control Using the Configuration Space Method," *Industrial Robot: An International Journal*, vol. 5, no. 2, pp. 69–73, 1978.

[137] A. Ray, J. Achiam, and D. Amodei, "Benchmarking Safe Exploration in Deep Reinforcement Learning," 2019.

[138] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron *et al.*, "A Generalist Agent," *Transactions on Machine Learning Research*, 2022.

[139] S. Ross and D. Bagnell, "Efficient Reductions for Imitation Learning," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 661–668.

[140] S. Ross, G. Gordon, and D. Bagnell, "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[141] 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.

[142] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-Real Robot Learning from Pixels with Progressive Nets," in *Conference on Robot Learning*. PMLR, 2017, pp. 262–270.

[143] F. Sadeghi, A. Toshev, E. Jang, and S. Levine, "Sim2real Viewpoint Invariant Visual Servoing by Recurrent Control," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4691–4699.

[144] K. R. Saul, X. Hu, C. M. Goehler, M. E. Vidt, M. Daly, A. Velisar, and W. M. Murray, "Benchmarking of Dynamic Simulation Predictions in Two Software Platforms Using an Upper Limb Musculoskeletal Model," *Computer Methods in Biomechanics and Biomedical Engineering*, vol. 18, no. 13, pp. 1445–1458, 2015.

[145] S. Schaal, "Learning from Demonstration," *Advances in Neural Information Processing Systems*, vol. 9, 1996.

[146] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust Region Policy Optimization," in *International Conference on Machine Learning*. PMLR, 2015, pp. 1889–1897.

[147] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico*, 2016.

[148] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[149] P. L. Semënovič, B. V. Grigorevich, G. R. Valerianovich, and M. E. Frolovich, *The Mathematical Theory of Optimal Processes*. Interscience Publishers, John Wiley and Sons Inc, 1962.

[150] W. F. Sharpe, "Mutual Fund Performance," *The Journal of Business*, vol. 39, no. 1, pp. 119–138, 1966.

[151] B. Siciliano, O. Khatib, and T. Kröger, *Springer Handbook of Robotics*. Springer, 2008, vol. 200.

[152] J. Siekmann, S. Valluri, J. Dao, L. Bermillo, H. Duan, A. Fern, and J. Hurst, "Learning Memory-Based Control for Human-Scale Bipedal Locomotion," *Robotics: Science and Systems (RSS)*, 2020.

[153] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," in *International Conference on Machine Learning*. PMLR, 2014, pp. 387–395.

[154] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche *et al.*, "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[155] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez *et al.*, "A General Reinforcement Learning Algorithm That Masters Chess, Shogi, and Go Through Self-Play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[156] M. J. Sobel, "The Variance of Discounted Markov Decision Processes," *Journal of Applied Probability*, vol. 19, no. 4, pp. 794–802, 1982.

[157] K. Srinivasan, B. Eysenbach, S. Ha, J. Tan, and C. Finn, "Learning to Be Safe: Deep RL with a Safety Critic," *arXiv preprint arXiv:2010.14603*, 2020.

[158] O. Stasse, A. Escande, N. Mansard, S. Miossec, P. Evrard, and A. Kheddar, "Real-Time (Self)-Collision Avoidance Task on a HRP-2 Humanoid Robot," in *2008 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2008, pp. 3200–3205.

[159] A. Stooke, J. Achiam, and P. Abbeel, "Responsive Safety in Reinforcement Learning by PID Lagrangian Methods," in *International Conference on Machine Learning*. PMLR, 2020, pp. 9133–9143.

[160] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2018.

[161] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," *Advances in Neural Information Processing Systems*, vol. 12, 1999.

[162] A. Tamar, D. Di Castro, and S. Mannor, "Policy Gradients with Variance Related Risk Criteria," in *International Conference on Machine Learning*. PMLR, 2012, pp. 387–396.

[163] C. S. Tan, R. Mohd-Mokhtar, and M. R. Arshad, "A Comprehensive Review of Coverage Path Planning in Robotics Using Classical and Heuristic Algorithms," *IEEE Access*, vol. 9, pp. 119 310–119 342, 2021.

[164] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner *et al.*, "Sim-to-Real: Learning Agile Locomotion for Quadruped Robots," *Robotics: Science and Systems (RSS)*, 2018.

[165] Y. Tassa, N. Mansard, and E. Todorov, "Control-Limited Differential Dynamic Programming," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 1168–1175.

[166] C. Tessler, D. J. Mankowitz, and S. Mannor, "Reward Constrained Policy Optimization," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA*, 2019.

[167] B. Thananjeyan, A. Balakrishna, S. Nair, M. Luo, K. Srinivasan, M. Hwang *et al.*, "Recovery RL: Safe Reinforcement Learning with Learned Recovery Zones," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4915–4922, 2021.

[168] G. Tiboni, R. Camoriano, and T. Tommasi, "PaintNet: Unstructured Multi-Path Learning from 3D Point Clouds for Robotic Spray Painting," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 3857–3864.

[169] 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.

[170] E. Todorov and W. Li, "A Generalized Iterative LQG Method for Locally-Optimal Feedback Control of Constrained Nonlinear Stochastic Systems," in *Proceedings of the 2005 American Control Conference*. IEEE, 2005, pp. 300–306.

[171] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A Physics Engine for Model-Based Control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 5026–5033.

[172] H. Trentelman, *Linear Quadratic Optimal Control*. Encyclopedia of Systems and Control, Springer London, 2013, pp. 1–8.

[173] A. Ude, A. Gams, T. Asfour, and J. Morimoto, "Task-Specific Generalization of Discrete and Periodic Dynamic Movement Primitives," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 800–815, 2010.

[174] 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.

[175] A. Wächter and L. T. Biegler, "On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming," *Mathematical Programming*, vol. 106, pp. 25–57, 2006.

[176] Y. Wada and M. Kawato, "A Via-Point Time Optimization Algorithm for Complex Sequential Trajectory Formation," *Neural Networks*, vol. 17, no. 3, pp. 353–364, 2004.

[177] N. Wahlström, T. B. Schön, and M. P. Deisenroth, "From Pixels to Torques: Policy Learning with Deep Dynamical Models," *arXiv preprint arXiv:1502.02251*, 2015.

[178] C. J. Watkins and P. Dayan, "Q-Learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[179] C. J. Watkins, "Learning from Delayed Rewards," Ph.D. dissertation, King's College, Cambridge United Kingdom, 1989.

[180] P. Wieland and F. Allgöwer, "Constructive Safety Using Control Barrier Functions," *IFAC Proceedings Volumes*, vol. 40, no. 12, pp. 462–467, 2007.

[181] R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[182] R. Wilson, "A Simplicial Algorithm for Concave Programming," *Harvard University: Cambridge, MA, USA*, 1963.

[183] Z. Xie, P. Clary, J. Dao, P. Morais, J. Hurst, and M. Panne, "Learning Locomotion Skills for Cassie: Iterative Design and Sim-to-Real," in *Conference on Robot Learning*. PMLR, 2020, pp. 317–329.

[184] Z. Xie, X. Da, M. Van de Panne, B. Babich, and A. Garg, "Dynamics Randomization Revisited: A Case Study for Quadrupedal Locomotion," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 4955–4961.

[185] T.-Y. Yang, J. Rosca, K. Narasimhan, and P. J. Ramadge, "Projection-Based Constrained Policy Optimization," in *8th International Conference on Learning Representations, ICLR 2020*, 2020.

[186] 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.

[187] T. Yu and H. Zhu, "Hyper-Parameter Optimization: A Review of Algorithms and Applications," *arXiv preprint arXiv:2003.05689*, 2020.

[188] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel, "Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 5628–5635.

[189] W. Zhao, T. He, and C. Liu, "Model-Free Safe Control for Zero-Violation Reinforcement Learning," in *5th Annual Conference on Robot Learning*, 2021.

[190] W. Zhao, T. He, R. Chen, T. Wei, and C. Liu, "State-Wise Safe Reinforcement Learning: A Survey," in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, 2023, pp. 6814–6822.

[191] M. Zhong, M. Johnson, Y. Tassa, T. Erez, and E. Todorov, "Value Function Approximation and Model Predictive Control," in *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE, 2013, pp. 100–107.

[192] Y. Zhou, "Learning Generalization and Adaptation of Movement Primitives for Humanoid Robots," Ph.D. dissertation, Karlsruhe Institute of Technology (KIT), 2020.

[193] Y. Zhou, J. Gao, and T. Asfour, "Learning Via-Point Movement Primitives With Inter- And Extrapolation Capabilities," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 4301–4308.

[194] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia *et al.*, "RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control," in *Proceedings of the 7th Conference on Robot Learning*, vol. 229. PMLR, 2023, pp. 2165–2183.

Acronyms

AI artificial intelligence

ASB alternative safe behavior

CBF control barrier function

CMDP constrained Markov decision process

CNN convolutional neural network

CPO constrained policy optimization

CPU central processing unit

DDP differential dynamic programming

DDPG deep deterministic policy gradient

DMP dynamical movement primitive

DOF degrees of freedom

DQN deep Q-network

DRL deep reinforcement learning

GP Gaussian process

GPS guided policy search

GPU graphics processing unit

GRU gated recurrent unit

HMM hidden Markov model

HRC human–robot collaboration

IPO interior-point policy optimization

ISSA implicit safe set algorithm

KL divergence Kullback–Leibler divergence

LfD learning from demonstration

LQR linear–quadratic regulator

LSTM long short-term memory

MCTS Monte Carlo tree search

MDP Markov decision process

MPC model predictive control

MSBE mean-squared Bellman error

OSC operational space control

OTG online trajectory generation

PCPO projection-based constrained policy optimization

PID controller proportional–integral–derivative controller

POMDP partially observable Markov decision process

PPO proximal policy optimization

ProMP probabilistic movement primitive

QP quadratic program

RA reachability analysis

RL reinforcement learning

RNN recurrent neural network

SAC soft actor-critic

SQP sequential quadratic programming

TOPP time-optimal path parameterization

TPU tensor processing unit

TRPO trust region policy optimization

VMP via-points movement primitive

List of Symbols

$_d$ used to indicate controller setpoints

$\hat{}$ used to indicate estimated values

$_{\mathrm{ref}}$ used to indicate values referring to a reference trajectory

t used to indicate a discrete time step

t used to indicate a continuous time

θ trainable network parameters

φ position of a robot joint

τ torque of a revolute robot joint

F force of a prismatic robot joint

s state of a Markov decision process

$\mathcal{S}$ state space of a Markov decision process

a action of a Markov decision process

$\mathcal{A}$ action space of a Markov decision process

P state-transition probabilities of a Markov decision process

R reward function of a Markov decision process

G return of a Markov decision process

γ discount factor of a Markov decision process

π policy of a Markov decision process

π_T indicates a task policy

π_B indicates a backup policy

$\Delta\mathrm{t}$ time between decision steps in a Markov decision process

f_D frequency of decisions in a Markov decision process

$\mathcal{C}$ set of cost functions of a constrained Markov decision process

C cost function of a constrained Markov decision process

c threshold of a constraint considered by a constrained Markov decision process

V state-value function

Q action-value function

$\bar{Q}$ action-risk function

$\bar{q}$ risk signal

$\bar{q}_{Th}$ threshold for risk classifications

A advantage function

L objective function / loss function

g policy gradient

∇ gradient (of)

$\Pr$ probability (of)

$\mathcal{N}$ Gaussian distribution

μ mean of a probability distribution

σ^2 variance of a probability distribution

p position

v velocity (first derivative of the position)

a acceleration (second derivative of the position)

j jerk (third derivative of the position)

d_{self} closest distance to a self-collision

d_{static} closest distance to a collision with a static obstacle

d_{moving} closest distance to a collision with a moving obstacle

$d_{\mathrm{safety}_{\mathrm{self}}}$ safety distance for self-collisions

$d_{\mathrm{safety}_{\mathrm{static}}}$ safety distance for collisions with static obstacles

$d_{\mathrm{safety}_{\mathrm{moving}}}$ safety distance for collisions with moving obstacles

M mass matrix (also known as inertia matrix)

C Coriolis matrix

τ_g torques resulting from the force of gravity

τ_{ext} torques resulting from external forces

$\mathrm{K_P}$ matrix specifying the proportional gain of a controller

$\mathrm{K_I}$ matrix specifying the integral gain of a controller

$\mathrm{K_D}$ matrix specifying the derivative gain of a controller

$\mathbb{E}$ expected value

α learning rate

$\mathcal{H}$ entropy

f_S simulation frequency of a physics simulator

f_C frequency of collision checks

List of Figures

List of Tables