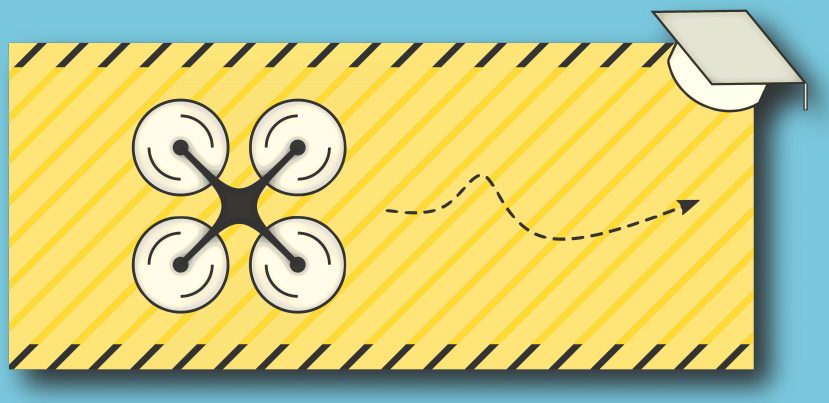
Safe Curriculum Learning for Primary Flight Control

Master of Science Thesis

T.S.C. Pollack



Technische Universiteit Delft

TU Delft

Safe Curriculum Learning for Primary Flight Control

Master of Science Thesis

by

T.S.C. Pollack

to obtain the degree of Master of Science
at the Delft University of Technology

Thesis committee:	Dr. Q. P. Chu,	TU Delft
	Dr. ir. E. van Kampen,	TU Delft, supervisor
	Dr. O. A. Sharpanskykh,	TU Delft

November 28, 2019

TU Delft

Preface

This thesis marks the end of my 6+ years as a BSc and MSc student at the TU Delft Faculty of Aerospace Engineering. I would like to thank my family and friends for their love and support all these years, who have given me the freedom and motivation to make the most out of my studies. I would also like to express my gratitude to dr. Erik-Jan van Kampen as my daily supervisor, for giving me the opportunity to work in the new and exciting field of reinforcement learning and providing me the necessary advice and insights at the challenging moments. I am up for the next opportunity, which is not very far away.

T.S.C. Pollack
Delft, November 2019

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

Roman Symbols

$\bar{c}$	Mean aerodynamic chord [m]
$\bar{q}$	Dynamic pressure [Pa]
$\boldsymbol{e}$	Error vector
$\boldsymbol{F}$	Force vector [N]
$\boldsymbol{F}(\boldsymbol{x})$	Force field
$\boldsymbol{g}$	Graviational acceleration vector [m/s^2]
$\boldsymbol{J}$	Inertia tensor [Nm^2]
$\boldsymbol{M}$	Moment vector [Nm]
$\boldsymbol{U}$	Control sequence
$\boldsymbol{u}$	Input vector
$\boldsymbol{u}^d$	Feedforward input vector
$\boldsymbol{u}^e$	Error feedback input vector
$\boldsymbol{u}_t$	Input at time t
$\boldsymbol{V}$	Velocity vector [m/s]
$\boldsymbol{v}$	Sample weight vector
$\boldsymbol{v}_r$	Risk weighting vector
$\boldsymbol{w}$	Weight vector
$\boldsymbol{x}$	State vector
$\boldsymbol{x}^r$	Reference state vector
$\boldsymbol{X}_t$	Augmented state vector at time t
$\boldsymbol{x}_t$	State at time t
$\boldsymbol{y}$	Observation vector
$\boldsymbol{Z}_t$	Augmented state-input vector at time t
ϵ	Probability of selecting a random input, change in curriculum step
$\hat{\tilde{F}}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{x}')$	Model of the joint state distribution function
$\hat{Q}_\pi(\boldsymbol{x},\boldsymbol{u})$	Approx. state-action value function
$\hat{V}_\pi(\boldsymbol{x})$	Approx. state value function
$\mathbb{1}$	Indicator function
$\mathbb{E}$	Expectation
$\mathbb{P}$	Probability
$\mathbb{R}$	Set of all real numbers
$\mathbb{T}$	Transformation matrix
$\mathscr{A}_{map}$	Knowledge mapping
$\mathscr{D}$	Training distribution
$\mathscr{D}'_\lambda$	Intermediate training distribution at step λ
$\mathscr{D}(\boldsymbol{x})$	Bounded disturbance set
$\mathscr{G}$	Hypergraph
$\mathscr{H}$	Hypothesis space
$\mathscr{K}$	Knowledge space, state constraint set
$\mathscr{Q}$	Space of attainable state-action value functions, CLF with bounded confidence intervals
$\mathscr{T}$	Set of transitions, tiling
$\mathscr{U}$	Input space
$\mathscr{V}$	Space of attainable state value functions, CLF level set
$\mathscr{X}$	State space
$\mathscr{X}'$	Augmented state space
$\mathscr{Y}$	Observation space
$\tilde{d}(\boldsymbol{x},\boldsymbol{u})$	Model prediction mismatch
$\tilde{F}$	Joint state distribution function
$\tilde{f}$	Transition joint probability density function
A	System matrix, aerodynamics
B	Input matrix
b	Body-fixed reference frame, wing span [m]
B_1	Augmented input matrix

C	Observation matrix
C_D, C_Y, C_L	Aerodynamic force coeff. [-]
C_l, C_m, C_n	Aerodynamic moment coeff. [-]
d	Number of basis functions
$d(\boldsymbol{x})$	Disturbance function
E	Vehicle-carried normal Earth reference frame
E_a	Actor loss
e_a	Actor training error
E_c	Critic loss
e_c	Critic training error
F	Command generator matrix
$F(\boldsymbol{x})$	Augmented internal dynamics
$f(\boldsymbol{x})$	Internal dynamics
$f(\boldsymbol{x}, \boldsymbol{u})$	Dynamics
g	Gravitational acceleration [m/s^2]
$G(\boldsymbol{u})$	Augmented input dynamics
$g(\boldsymbol{u})$	Input dynamics
$g(\boldsymbol{x}, \boldsymbol{u})$	Model error
G_t	Random return
$G_t^{(n)}$	n-step return
G_t^{λ}	Random λ-return
H	State-action kernel matrix, hypothesis, risk perception region
$h(\boldsymbol{x}, \boldsymbol{u})$	Prior model
$H(\mathscr{D})$	Entropy of training distribution
K_1	Augmented control feedback matrix
L	Loss function, set of LTF states, Lipschitz constant
l	Observation dimensionality
M	Markov Decision Process
m	Input space dimensionality, mass [kg]
n	State space dimensionality
P	State kernel matrix, propulsion
p	Command generator dimensionality
p, q, r	Roll, pitch, and yaw rate [rad/s]
Q	State cost matrix
$Q_{\tilde{\pi}}(\boldsymbol{x}, \boldsymbol{u})$	State-action value function
R	Input cost matrix, ellipsoid, region
R_t	Random reward or utility
r_{t+1}	Observed return for a transition at time $t+1$
S	Reference wing area [m^2]
s	Stability reference frame
T	Episode duration, batch size, augmented state matrix, set of target states, thrust
T_t	Random trajectory from time t onwards
u, v, w	Body speed components [m/s]
U_t	Random input vector
$v(\boldsymbol{x})$	Control Lyapunov Function
$V_{\tilde{\pi}}(\boldsymbol{x})$	State value function
$W(\boldsymbol{x})$	Risk perception function
$W_{\lambda}(z_{t+1})$	Training distribution weight function λ
X_t	Random state vector
Y_t	Random observation vector
Z_t	Random eligibility trace
z_{t+1}	Transition tuple
$Q_*(\boldsymbol{x}, \boldsymbol{u})$	Optimal state-action value function
$V_*(\boldsymbol{x})$	Optimal state value function

Greek Symbols

α	Learning rate, angle of attack [rad]
β	Belief, confidence level
$\boldsymbol{\delta}$	Equilibrium reaching interval
$\boldsymbol{\Delta}(\boldsymbol{x}, \boldsymbol{u})$	Bounding model
$\boldsymbol{\epsilon}$	Closeness reaching interval
$\boldsymbol{\omega}$	Angular rate vector [rad/s]
$\boldsymbol{\phi}$	Basis function vector
$\boldsymbol{\theta}$	Policy parameter vector
Δ	Difference, function approximation residual, tiling width

δ_{t+1}	Temporal difference error
η	Cut-off threshold
γ	Discount rate
κ	Learning complexity parameter
λ	Eligibility decay rate, step in curriculum sequence
Π	Policy space
$\mu_n(\cdot)$	Posterior mean conditioned on n data points
$\Phi(\boldsymbol{x})$	Potential function
Φ, Θ, Ψ	Euler attitude angles [rad]
Π	Space of attainable policies
$\Psi(\boldsymbol{x}_t^r)$	Command generator dynamics
ρ	Risk tuning parameter, intersection volume
$\Sigma_n(\cdot)$	Posterior covariance matrix conditioned on n data points
$\sigma_n(\cdot)$	Posterior variance conditioned on n data points
τ	Trajectory, reward threshold, tile, time horizon
$\tilde{\pi}, \pi$	Control policy
$\tilde{\rho}, \rho$	Reward or utility function
$\pi_*(\boldsymbol{x})$	Optimal control policy

Acronyms

ADDHP	Action-Dependent Dual Heuristic Dynamic Programming
ADGDHP	Action-Dependent Globalized Dual Heuristic Dynamic Programming
ADHDP	Action-Dependent Heuristic Dynamic Programming
ADP	Approximate/adaptive Dynamic Programming
AFF	Artificial Force Field
ANN	Artificial Neural Network
ARE	Algebraic Ricatti Equation
CLF	Control Lyapunov Function
CMAC	Cerebellar Model Arithmetic Computer
CurL	Curriculum Learning
DCRL	Deep Curriculum Reinforcement Learning
DHP	Dual Heuristic Dynamic Programming
DP	Dynamic Programming
EOM	Equations Of Motion
FSS	Fatal State Space
GDHP	Globalized Dual Heuristic Dynamic Programming
GP	Gaussian Process
GPF	Generalized Potential Field
GPI	Generalized Policy Iteration
HDP	Heuristic Dynamic Programming
HJB	Hamilton-Jacobi-Bellman
HJI	Hamilton-Jacobi-Isaacs
iADP	Incremental Approximate Dynamic Programming
IHDP	Incremental Heuristic Dynamic Programming
LBMPC	Learning-Based Model-Predictive Control
LQR	Linear Quadratic Regulation
LQT	Linear Quadratic Tracking
LS	Least-Squares
LSPI	Least-Squares Policy Iteration
LSQL	Least-Squares Q-learning
LSTD	Least-Squares Temporal Difference
LTF	Lead-To-Fatal
MC	Monte Carlo
MDP	Markov Decision Process
MPC	Model-Predictive Control
MSD	Mass-Spring-Damper
NDI	Nonlinear Dynamic Inversion
ODE	Ordinary Differential Equation
PE	Persistence of Excitation
PI	Policy Iteration
RBF	Radial Basis Function

RCLF	Robust Control Lyapunov Function
RFSS	Restricted Fatal State Space
RL	Reinforcement Learning
RNN	Recurrent Neural Network
ROA	Region Of Attraction
RSS	Restricted State Space
SGD	Stochastic Gradient Descent
SHERPA	Safety Handling Exploration with Risk Perception Algorithm
SL	Safe Learning, Supervised Learning
SPL	Self-Paced Learning
SPLD	Self-Paced Learning with Diversity
SSS	Safe State Space
TD	Temporal Difference
TL	Transfer Learning
UAV	Unmanned Aerial Vehicle
VI	Value Iteration

1 Introduction

AUTOMATIC flight control system (AFCS) design commonly relies on the availability of accurate model descriptions of the system dynamics, enabling common engineering practice such as computer-aided design (for a good reference in aircraft control design, see e.g. [94]). This process calls for an iterative design cycle that typically consists of system identification based on first-order principles, simulation modeling, control law design, ground tests, and eventually flight tests. Consequently, control law design is largely model-based, which calls for extensive verification and validation activities to ensure that the design is coherent with the real, physical system. For newly developed aerial vehicles, the need for high quality models calls for extensive aerodynamic analysis over a wide range of flow conditions based on Computational Fluid Dynamics (CFD) computations, scaled wind tunnel measurements, and flight tests. This shows that flight control system design is a complex process that may consume a considerable part of the total development time and costs of an aerial vehicle.

The increasing interest in advanced vehicle concepts and recent widespread adoption of unmanned aerial vehicles (UAVs) for a wide range of applications puts considerable strain on this model-based development cycle. For systems with complex dynamics, control performance and robustness properties may be unsatisfactory or simply unacceptable if an adequate model cannot be obtained, which may be further amplified in demanding flight conditions. Examples that fall under this category are highly advanced supersonic aircraft such as the Innovative Control Effectors concept [77] and re-entry vehicles, but also micro air vehicles (MAVs) such as Delfly [19]. For UAVs, a large demand for different concepts and applications in combination with a conflicting desire for low costs may lead to infeasible requirements in terms of development time and costs. This may very well forfeit some promising opportunities.

Modern control engineering has developed various techniques that allow for reduced model fidelity requirements. Robust control methods such as H_∞-synthesis provide reliability guarantees in terms of closed-loop stability and performance in the face of model uncertainties and disturbances [31, 39, 109]. However, although these methods are generally very effective, they can also lead to conservatism in the control system design [90]. Other methods aim at reducing the need for a global model altogether, by adopting a sensor-based approach. These include the incremental counterparts of Nonlinear Dynamic Inversion (NDI) [92] and Backstepping (BS) [93], known as Incremental Nonlinear Dynamic Inversion (INDI) [90] and Incremental Backstepping (IBS) [3], respectively. These methods reduce the impact of model mismatch by directly feeding back acceleration measurements. As a result, the only model terms required for the control law are those related to the control effectiveness. However, this leads to new problems related to synchronization of sensor measurements, especially when extensive filtering is required.

From this, a case can be postulated for fully autonomous flight control laws that shape itself online on the real system. Reinforcement learning (RL) is a very promising paradigm for generating optimal, adaptive control laws for systems with uncertain dynamics [95]. Having its roots in behavioral learning and optimal control, this framework has the potential to addresses various challenges that come with flight control design for complex aerospace systems such as those mentioned before. In case of inexpensive, small UAVs for example, RL could be adopted to learn optimal control of the system online from scratch. Alternatively, RL methods could be used to train a flight control system off-line based on a crude model description of the system dynamics, to be followed by an online fine-tuning phase. This may considerably reduce development time and costs, as knowledge about the real system is generated autonomously by the controller itself.

1.1. Contribution

Despite its appealing benefits, reinforcement learning comes with its own challenges that currently severely limit its suitability for deployment on real safety-critical systems. First, very large amounts of data are typically required during the learning phase. This is especially true for high-dimensional state and action spaces, which are typical for aerospace systems. Curriculum learning (CurL) [10] is a recently proposed paradigm that specifically deals with the question of how this data dependency can be reduced by making the learning process more tractable. This can for example be done by breaking down the complexity of a learning task and training the agent first on a range of related tasks that are significantly easier to learn. This concept exhibits strong analogies with human education, where students are presented with easier material first before moving on to more complex subjects. This principle could be leveraged in autonomous shaping of a flight control law, by subjecting the agent to an appropriate learning curriculum that allows it to gradually expand its capabilities.

A second major challenge in RL is how to perform online training in a safe manner, i.e. how to let the agent learn autonomously without damaging itself or its surroundings. This idea has given rise to another major branch of RL research known as Safe Learning (SL). A large number of methods has been proposed in this field, including safety filters [70], imposing initial domain knowledge [79], and action limiting [34]. However, little work has been performed that unite the concepts of curriculum learning and safe learning, whereas this may be of large benefit to the applicability of RL on safety-critical systems such as UAVs. Therefore, the contribution of this research project is to demonstrate how effective learning curricula can be constructed for a reinforcement learning agent to learn optimal control of an aerial vehicle in a safe manner.

1.2. Related work

Although safe curriculum learning for flight control is still a largely unexplored research area, other paradigms have been investigated that aim to address the said safety and efficiency issues that undermine online reinforcement learning applications. Recently, the idea of decomposing a target task into a pair of tasks of lower complexity has been studied in the context of flight control, using a *flexible heuristic dynamic programming* (HDP) approach [35]. This work focused on a quadrotor hover task, for which a reinforcement learning agent is trained to optimize the UAV's vertical trajectory by applying both collective and pitch inputs. A sequential learning approach is adopted where the main hover task is preceded by an MDP for which the internal state and input space have both been limited to only a subset of their original equivalents. In this first stage, the agent only needs to learn how to use the collective optimally to fly the quadrotor to a target altitude and maintain it there. Since the system remains free from any disturbances in the non-controllable and non-observable states, the resulting MDP is fully observable. Once learning has completed for this stage, the pitch state and input dimensions are added to the agent's internal representation, allowing it to further optimize its performance. This second stage is in fact the original hover task. It was demonstrated that thanks to this sequential learning approach, the learning process becomes more efficient, whereas the learned behavior also becomes more robust. The key contribution of this work was to show the effectiveness of flexible function approximators (FFAs), which form the key element in modifying the agent's internal representation.

The idea of flexible function approximation can essentially be seen as an equivalent form of transfer learning (TL), which is a branch of RL research that specifically deals with the question how the large data dependency of reinforcement learning paradigm can be reduced by transferring knowledge obtained from training on one task to another task in which learning has not yet taken place. Transfer learning takes a key position in curriculum learning for RL. Consequently, the study by [35] can be considered work in curriculum learning, although it was not explicitly referred to. Note that [35] does not consider safety during learning, which is also related to the fact that learning is done in a fully off-line (episodic) manner.

Beyond the area of curriculum learning, [68] presents another recent study which does incorporate safety in the context of sample-efficient online reinforcement learning in UAV applications. Here, the focus is on navigating through an environment cluttered by obstacles with the goal of locating an unknown target. The presented approach makes use of hierarchical reinforcement learning (HRL) techniques based on a managerial control structure to abstract away the uncertainty related to the goal-finding task. Specifically, the reinforcement learning controller is structured by two hierarchical agents, referred to as the *supermanager* and the *submanager*, respectively. To some extent, these can be viewed as outer and inner control loops, according to classical control terminology. The submanager is responsible for avoiding collisions with the obstacles, whereas the supermanager has a global view of the task and navigates the UAV across the room. Based on its representation of the room, the supermanager sends target directions to the submanager, which

only duty is then to guide the UAV safely in towards these targets. Safety is accomplished by training the submanager off-line in a virtual environment based on an approximate model of the UAV dynamics. This training environment is a much smaller, but sufficiently representative equivalent of the real environment. It is demonstrated that the safe HRL controller is able to considerable reduce the risk of collision compared to a flat RL agent for an equivalent number of training episodes.

Although the safe HRL and safe CurL paradigms are widely different approaches, their motivation can be considered somewhat similar in that they aim to provide meaningful structure to the learning task. Whereas safe CurL addresses the problem by modifying the complexity of the task, safe HRL employs the idea of state abstraction to constrain the available policy space. However, whereas for the latter prior knowledge is limited to the controller design, safe CurL also incorporates this knowledge in the task structure itself.

1.3. Research formulation

A research framework has been established to demarcate the scope of this project and steer the research activities such that the effectiveness of safe curriculum learning for online intelligent primary flight control can be demonstrated. The main research objective has been formulated as follows:

Research objective

The research objective is to further advance reinforcement learning techniques for primary flight control applications in terms of safety and learning efficiency by demonstrating how effective machine learning curricula can be constructed that promote safe control behavior during the learning process.

Here, the process of learning is considered safe if the controlled process stays clear of the fatal state space (FSS) throughout the lifetime of the decision-making agent. In the case of UAVs, this essentially forms their operational envelope intersected with the available learning space (e.g., a control room). Section 1.3.2 provides a set of definitions to further clarify the terminology. The main research question that has been posed to achieve the research objective reads as:

Main research question

How can a learning curriculum be implemented for a reinforcement learning agent to autonomously learn optimal primary flight control laws for an Unmanned Aerial Vehicle (UAV) with uncertain dynamics, such that learning efficiency is enhanced and the risk of violating safety constraints is reduced?

Further substantiation of the required research activities is necessary here. Four central themes have been identified, relating to (1) the fundamental notion of task complexity and safety of learning, (2) essential requirements for safe and effective machine learning curricula, (3) transfer of knowledge across tasks, and (4) representation of the learning agent and auxiliary safeguard mechanisms. These concepts form the foundation of the research sub-questions, which are listed on the following page.

1.3.1. Approach

The research framework forms the basis for finding relevant literature in the fields of reinforcement learning, safe learning, transfer learning, curriculum learning, and intelligent control techniques. Existing theories and views from these related, but still largely isolated fields are synthesized in order to establish a unified, coherent framework for safe curriculum learning. This is primarily based on a theoretical perspective, supported by a range of empirical experiments that serve to gain insight into the various aspects that underlie the problem of safe curriculum learning in the context of adaptive optimal control. In this view, the following approach is adopted:

1. *Establish theoretical framework for curriculum learning in the context of adaptive optimal control*; As curriculum learning takes a central role in this research, an adequate theoretical basis is established before proceeding with the main objective of safety assessment and advancement. This is done by taking key definitions, interpretations, and implementations reported in relevant literature and integrating these in a comprehensive framework that translates well to the case of adaptive optimal control.

2. *Identify key variables that determine learning efficiency and safety of a generic learning curriculum*; Based on the derived framework and resulting taxonomy, empirical analysis is performed to gain in-

Research sub-questions

1. How can safe learning be supported by reducing the complexity of a given task?
   (a) What defines task complexity?
   (b) What are the key concepts behind safe learning?
   (c) Which relationships can be identified across these concepts?
2. What are the requirements for a sequence of intermediate flight control learning stages with respect to the task domain and the agent domain, such that fast learning can be achieved in a safe way?
   (a) What is needed in terms of task complexity, ordering, and timing?
   (b) What are the capabilities that the agent should possess at each stage of the learning process?
   (c) How does the objective function need to be set up across the learning curriculum?
   (d) How to represent safety constraints?
3. How can knowledge gained in a previous learning stage (the source) be adopted to bias learning in a later stage (the target) in a safe way?
   (a) What knowledge needs to be extracted from the source?
   (b) How can this knowledge be represented?
   (c) What needs to be known about the target?
   (d) How to handle those parts of the target state and input space for which no knowledge is available?
   (e) What technique can be used to facilitate safe and effective transfer of knowledge across these stages?
4. What reinforcement learning architecture must be adopted to learn increasingly advanced flight control capabilities in a safe manner?
   (a) What elements does the learning architecture need to consist of?
   (b) What learning technique needs to be used for solving the optimal control problem?
   (c) What function approximation technique is suitable in terms of approximation power, sample efficiency, and transfer flexibility?
   (d) What exploration strategy needs to be adopted?
5. What metrics are required to quantify learning efficiency and safety?

sight into the various aspects involved in the implementation of a given learning curriculum and how these affect learning efficiency and safety.

3. *Integrate safe learning mechanisms to further support safety during learning*; Existing theories from safe learning are integrated with the newly developed framework for curriculum learning for adaptive optimal control. By taking a tentative view of the theoretical concepts involved, the safe curriculum learning paradigm is established.

4. *Identify key variables and correlations in safe curriculum learning that determine learning efficiency and safety;* Here, a empirical study is again performed to obtain further insight into the safe curriculum learning paradigm and establish how algorithms perform in practice.

5. *Apply safe curriculum learning to the case of intelligent primary flight control*; Here, the resulting framework is used in a case study where a reinforcement learning agent must learn a set of primary flight control laws autonomously using limited initial knowledge. In this step, the relationship between safe curriculum learning and intelligent flight control is further investigated in particular.

The knowledge gained from literature and the preliminary experiments is used to formulate generic answers to the research questions. This allows for an informed postulation of hypotheses for safe curriculum learning in primary flight control, and the design of an experiment to test these hypotheses.

1.3.2. Definitions

The central concepts in the research formulation must be unambiguously defined in order to clearly delineate the research scope. These definitions stipulate the respective domains in qualitative terms. This subsection introduces the key terminology used in this work, and can be used as future reference as well.

Definition 1.1: Learning task

A *learning task* is a sequential decision-making problem described as a Markov Decision Process (MDP) that needs to be solved by a decision-making agent starting from limited initial knowledge.

This definition of a learning task specifies the formal framework that will be adopted, the Markov Decision Process (MDP). In general, MDPs can be stochastic, nonlinear and time-varying. Note that partially observable MDPs (POMDPs) are excluded from this definition, meaning that the process must be fully observable.

Definition 1.2: Safe learning

The process of learning is considered *safe* if the controlled process stays clear of the fatal state space (FSS) throughout the lifetime of the decision-making agent.

The definition of safety during learning as specified here is consistent with the framework developed by [65]. The FSS concept refers to those states that prevent the agent from further meaningful learning, such as critical damage. This will be treated more formally in later chapters.

Definition 1.3: Curriculum learning

In *curriculum learning*, the learning process for a given learning task is based on a sequence of auxiliary learning tasks of reduced complexity.

This interpretation of curriculum learning gives rise to a very broad range of solutions. For example, a learning sequence can be formed by a chain of different but sufficiently related MDPs, or by learning selectively within the target MDP itself. A learning curriculum is then referred to as the complete task sequence. Note that the above definition allows for multiple levels of authority, ranging from a fixed learning curriculum specified by an external designer, to fully autonomous self-shaping by the agent.

Definition 1.4: Transfer learning

In *transfer learning*, knowledge obtained from learning one task, known as the source task, is used to bias learning in a different task, known as the target task.

Transfer learning takes a central position in machine learning curricula for reinforcement learning. For a large class of curricula (not always), knowledge may need to be transferred between different learning stages. Transfer learning often provides the necessary ingredients for doing so.

Definition 1.5: Task domain

The *task domain* includes those aspects within the boundaries of the learning problem that lie outside the authority of the decision-making agent.

Curriculum learning for reinforcement learning often calls for a simplification of a learning task, a clear decomposition of the learning problem is necessary. With this definition is the task domain, it is clear what can be done regarding the task itself to make the learning process more tractable.

Definition 1.6: Actor domain

The *actor domain* consists of those elements that constitute a decision-making agent and its supporting actors.

This definition of the actor domain effectively complements the previous demarcation of the task domain. It does not only refer to the learning agent itself, of which the constituents are also referred to as the *agent domain*, but also of supporting (i.e., external) entities such as other (more proficient) agents, controllers, and even humans. Note that this definition transcends the concept of an actor in the actor-critic framework.

Definition 1.7: Task ordering

Task ordering refers to the order in which a range of learning tasks is presented to a learning agent.

Although its definition rather trivial, task ordering is key for the majority of a-priori designed machine learning curricula.

Definition 1.8: Task timing

Timing of a given task refers to the moment in time where a learning agent is presented with a new learning task.

Whereas task ordering only refers to the order in which a range of tasks is presented, task timing explicitly refers to the point in the time where the curriculum proceeds to the next learning stage. Note that other terms, such as learning complexity, require more background from literature and will therefore not be discussed in this introductory overview.

1.4. Report structure

The thesis has been split into four parts. The main contribution is presented in Part I, which consists of a scientific article that describes the application of the proposed safe curriculum learning framework in the context of neural-network based optimal control of a generic mass-springer-damper system and a nonlinear quadrotor UAV. The article summarizes the main findings from Part II, which contains an elaborate literature study on the canonical fields of reinforcement learning, adaptive optimal control, curriculum learning, and safety learning. Moreover, it builds further on the content from Part III, which presents the empirical findings from a preliminary study where flat reinforcement learning, curriculum learning, safe learning, and safe curriculum learning are applied in the context of linear optimal control. The thesis is concluded by Part IV, where answers to the research questions are formulated and recommendations are made for future research,

I

Article

Safe Curriculum Learning for Optimal Flight Control of Unmanned Aerial Vehicles with Uncertain System Dynamics

T.S.C. Pollack *
Delft University of Technology, 2629HS Delft, The Netherlands

Reinforcement learning (RL) enables the autonomous formation of optimal, adaptive control laws for systems with complex, uncertain dynamics. This process generally requires a learning agent to directly interact with the system in an online fashion. However, if the system is safety-critical, such as an Unmanned Aerial Vehicle (UAV), learning may result in unsafe behavior. Moreover, irrespective of the safety aspect, learning optimal control policies from scratch can be inefficient and therefore time-consuming. In this research, the safe curriculum learning paradigm is proposed to address the problems of learning safety and efficiency simultaneously. Curriculum learning makes the process of learning more tractable, thereby allowing the intelligent agent to learn desired behavior more effectively. This is achieved by presenting the agent with a series of intermediate learning tasks, where the knowledge gained from earlier tasks is used to expedite learning in succeeding tasks of higher complexity. This framework is united with views from safe learning to ensure that safety constraints are adhered to during the learning curriculum. This principle is first investigated in the context of optimal regulation of a generic mass-spring-damper system using neural networks, and is subsequently applied in the context of optimal attitude control of a quadrotor UAV with uncertain dynamics.

Nomenclature

C_D, C_L	=	Aerodynamic force coefficients	c	=	Damping coefficient, N/m s^{-1}
C_T	=	Thrust constant	$f(\boldsymbol{x})$	=	Internal state dynamics
C_β, C_{β_m}	=	Blade flapping constants	k	=	Spring coefficient, N/m
C_τ	=	Rotor torque constants	l	=	Rotor arm length, m
C_{τ_D}	=	Rotational drag coefficient	m	=	Input state dimension; mass, kg
$G(\boldsymbol{x})$	=	Control effectiveness matrix	n	=	State space dimension
H	=	Risk perception region	p, q, r	=	Roll, pitch and yaw rate, rad
$\boldsymbol{J}, J$	=	Inertia, kg m^2	r	=	Observed utility
K	=	Number of policy evaluation iterations	$\boldsymbol{u}_t$	=	Input vector at time t
M	=	Markov decision process	u, v, w	=	Body speed components, m/s
N_c, N_a	=	Number of hidden nodes	$v(\boldsymbol{x})$	=	Control Lyapunov function
Q	=	State weighting matrix	$\boldsymbol{x}_t$	=	State vector at time t
R	=	Input weighting matrix	α	=	Angle of attack, rad
T_1, T_2, T_3, T_4	=	Rotor thrust, N	β	=	Sideslip angle, rad
$V_*(\boldsymbol{x})$	=	Optimal state value function	γ	=	Discount rate
$V_{\tilde{\pi}}(\boldsymbol{x})$	=	State value function	$\boldsymbol{\Delta}(\boldsymbol{x}, \boldsymbol{u})$	=	Bounding model
$W(\boldsymbol{x})$	=	Risk perception function	$\boldsymbol{\epsilon}$	=	Closeness reaching interval
$\mathcal{A}_{map}$	=	Knowledge mapping	λ	=	Step in curriculum sequence
$\tilde{\mathcal{F}}$	=	State transition function	$\pi(\boldsymbol{x})$	=	Control policy
$\mathbb{T}_{ij}$	=	Transformation matrix from frame j to i	$\tilde{\rho}$	=	Performance index
$\mathcal{U}$	=	Input space	ϕ, θ, ψ	=	Euler attitude angles, rad
$\mathcal{X}$	=	State space	$\omega_1, \omega_2, \omega_3, \omega_4$	=	Rotor rotational velocity, rad/s

*Graduate student, Control and Simulation Division, Faculty of Aerospace Engineering, Kluyverweg 1, 2629HS Delft, the Netherlands

I. Introduction

Automatic flight control system (AFCS) design commonly relies on the availability of accurate model descriptions of the system dynamics, enabling common engineering practice such as computer-aided design [1]. This process calls for an iterative design cycle that typically consists of system identification based on first-order principles, simulation modeling, control law design, ground tests, and eventually flight tests. Consequently, control law design is largely model-based, which calls for extensive verification and validation activities to ensure that the design is coherent with the real, physical system. For newly developed aerial vehicles, the need for high quality models calls for extensive engineering analysis and judgment that involves computer simulations, wind tunnel measurements, ground tests, and flight tests. This indicates that flight control system design is a complex process that may consume a considerable part of the total development time and costs of an aerial vehicle.

Modern control engineering has developed various techniques that allow for reduced model fidelity requirements. Robust control methods such as H_∞-control provide certain reliability guarantees in terms of closed-loop stability and performance in the face of model uncertainties and disturbances [2–4]. However, although these methods are generally very effective, they can also lead to conservatism in the control system design [5]. Other methods aim at reducing the need for a global model altogether, by adopting a sensor-based approach. These include the incremental counterparts of Nonlinear Dynamic Inversion (NDI) [6] and Backstepping (BS) [7], known as Incremental Nonlinear Dynamic Inversion (INDI) [5] and Incremental Backstepping (IBS) [8], respectively. These methods reduce the impact of model mismatch by directly feeding back acceleration measurements. As a result, the only model terms required for the control law are those related to the control effectiveness. However, this leads to new problems related to synchronization of sensor measurements, especially when extensive filtering is required.

From this, a case can be postulated for fully autonomous flight control laws that shape themselves online on the real system. Reinforcement Learning (RL) [9–12] is a very promising paradigm for generating optimal, adaptive control laws for systems with uncertain dynamics [13]. In case of inexpensive, small UAVs for example, RL could be adopted to learn optimal control of the system on-line from scratch. Alternatively, RL methods could be used to train a flight control system off-line based on a crude model description of the system dynamics, to be followed by an on-line fine-tuning phase. This may considerably reduce development time and costs, as knowledge about the real system is generated autonomously by the controller itself.

Motivation

Despite its appealing benefits, reinforcement learning comes with its own challenges that currently severely limit its suitability for deployment on real safety-critical systems. First, large amounts of data are typically required during the learning phase. This is especially true for high-dimensional state and input spaces, which are typical for aerospace systems. Curriculum learning [14, 15] is a machine learning paradigm that specifically deals with the data dependency issue by making the learning process more tractable. This can for example be done by breaking down the complexity of a learning task and training the agent first on a range of related tasks that are significantly easier to learn. This concept exhibits strong analogies with human education, where students are presented with easier material first before moving on to more complex subjects. This principle can be leveraged for autonomous shaping of flight control laws, by subjecting a learning agent to an appropriate task curriculum that allows it to gradually expand its capabilities.

A second major challenge in RL is how to perform online training in a safe manner, i.e. how to let the agent learn autonomously without damaging itself or its surroundings. This idea has given rise to another major branch of RL research known as Safe Learning (SL) [16]. A large number of methods has been proposed in this field, including safety filters [17, 18], imposing initial domain knowledge [19], and action limiting [20]. However, little work has been performed that unite the concepts of curriculum learning and safe learning, whereas this may be of large benefit to the applicability of RL on safety-critical systems such as UAVs. Therefore, the contribution of this research is to demonstrate how effective learning curricula can be constructed for a reinforcement learning agent to learn optimal control of an aerial vehicle in a safe manner.

Related work

Although safe curriculum learning for flight control is still a largely unexplored research area, other (related) paradigms have been investigated that aim to address the said safety and efficiency issues that undermine online reinforcement learning applications. Recently, Helmer et al. [21] proposed the framework of flexible heuristic dynamic programming (HDP), and showed how decomposing a target task into a pair of successive tasks of lower complexity can expedite learning in the context of flight control. The concept of flexible function approximation can be regarded as a

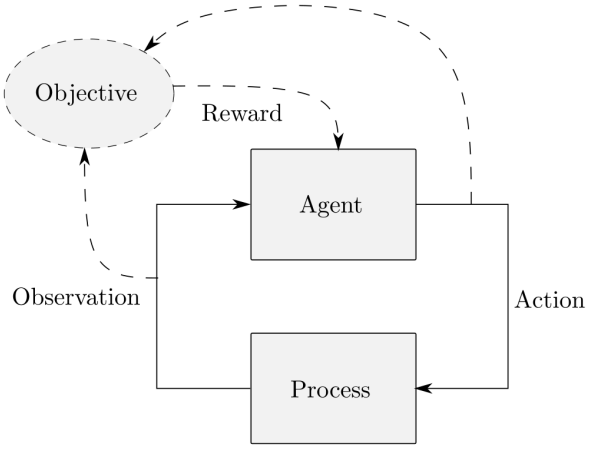


Fig. 1 The sequential decision-making problem. Adapted from [10].

form of transfer learning (TL), which is a branch of RL research that specifically deals with the question of how the large data dependency of the reinforcement learning paradigm can be reduced by transferring knowledge obtained from training on one task to another task in which learning has not yet taken place [22]. Transfer learning takes a key position in curriculum learning for RL [15]. Consequently, the study by Helmer et al. [21] can be considered work in curriculum learning, although the authors did not explicitly recognize it as such. Safety during learning was not considered in their study, which is also related to the fact that learning was performed in a fully off-line (episodic) manner.

Outline

This article is structured as follows. In Section II, the safe curriculum learning paradigm is substantiated by briefly covering the necessary background on reinforcement learning, safe learning, and curriculum learning. This is followed by a description of a safe curriculum approach to optimal control of discrete-time systems in Section III. The resulting framework is tested in a simple experiment based on a linear, unstable, cascaded mass-spring-damper (MSD) system in Section IV, where its performance is compared to other non-curriculum approaches. Subsequently, a flight control application focusing on optimal attitude control of a Parrot quadrotor Unmanned Aerial Vehicle (UAV) is described in Section V. Finally, conclusions and recommendations for further research are drawn in Section VI.

II. Fundamentals

Demonstrating how safe and effective machine learning curricula can be constructed requires an adequate substantiation of the safe curriculum learning paradigm. In this section, a general theoretical framework is formulated that describes in high-level terms the fundamental concepts and requirements for safe curriculum learning. To this end, Section II.A provides a very brief introduction to the fundamentals of reinforcement learning and adaptive/approximate dynamic programming. This is followed by a short description of curriculum learning in Section II.B, based on existing views reported in literature and the authors' interpretations. In Section II.C, key concepts in the area of safe learning are discussed. In Section II.D, these views are projected on the curriculum learning paradigm to establish the safe curriculum learning framework.

A. Reinforcement Learning and Adaptive/Approximate Dynamic Programming

Generally, RL can be applied to solve sequential decision-making problems (SDMP) formulated as Markov Decision Processes (MDPs) in the absence of an explicit model description of the system [10]. SDMPs are often regarded as discrete-time formulations of the optimal control problem, and can be visualized as shown in Figure 1. In this framework, a decision-making entity repeatedly interacts with a certain process or environment described by a state distribution function (dynamics) $\tilde{F} : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \mapsto [0, 1]$, where it receives a scalar reward or utility $R_{t+1} = r$ after each transition based on a certain objective captured by a (possibly stochastic) reward function or performance index $\tilde{\rho} : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \mapsto \mathbb{R}$ [10, 12]. Here, $\mathcal{X} \subset \mathbb{R}^n$ and $\mathcal{U} \subset \mathbb{R}^m$ represent the state and input spaces associated with the MDP. In brief, an MDP can be concisely referred to as a tuple $M : \langle \mathcal{X}, \mathcal{U}, \tilde{F}, \tilde{\rho}, \gamma \rangle$. In the field of artificial intelligence, the decision-making entity is referred to as the agent, whereas in control theory actions are generated by a controller

consisting of a set of control laws. RL methods can directly operate on data obtained from the system to learn the (near-)optimal decision sequence for the problem at hand [12]. Ultimately, the goal of the agent is to maximize the reward sequence from every state $X_t \in \mathcal{X}$, denoted as the return. In discrete-time, the return can be formulated as the algebraic sum of total collected utility, discounted at every time step by a discount rate $0 \leq \gamma \leq 1$ [10]:

$$G_t \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

The goal of maximizing (or minimizing) the discounted return G_t can be attained by the agent through an appropriate sequence of control actions that together constitute an optimal trajectory. Such a decision sequence is captured by a policy (control law) $\tilde{\pi} : \mathcal{X} \mapsto \mathcal{U}$. The expected return by following a control policy $\tilde{\pi}(\boldsymbol{u}|\boldsymbol{x})$ from a given state $X_t = \boldsymbol{x}$ gives rise to the concept of value functions, defined as [10]:

$$V_{\tilde{\pi}}(\boldsymbol{x}) \doteq \mathbb{E}_{\tilde{\pi}}\left[G_t | X_t = \boldsymbol{x}\right] = \mathbb{E}_{\tilde{\pi}}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | X_t = \boldsymbol{x}\right] \tag{2}$$

This expression is often rewritten in recursive form, in order to make the process of evaluating states and policies more tractable. This leads to the Bellman expectation equation [10]:

$$V_{\tilde{\pi}}(\boldsymbol{x}) = \mathbb{E}_{\tilde{\pi}}\left[R_{t+1} + \gamma V_{\tilde{\pi}}(X_{t+1}) | X_t = \boldsymbol{x}\right] \tag{3}$$

The value function of a given policy $\tilde{\pi}$ can be used to find new, better policies. The concept of value functions can also be applied to state-action pairs, which enables direct evaluation of any action $U_t = \boldsymbol{u}$ given the current state $X_t = \boldsymbol{x}$ and following the policy $\tilde{\pi}(\boldsymbol{u}|\boldsymbol{x})$ thereafter [10, 12]. The iterative process of value function prediction and policy improvement eventually converges to the optimal policy, which maximizes the expected return for every state $X_t \in \mathcal{X}$ [10]:

$$\pi_*(\boldsymbol{u}|\boldsymbol{x}) = 1 \iff \boldsymbol{u} = \arg\max_{\boldsymbol{u}} \mathbb{E}\left[R_{t+1} + \gamma V_*(X_{t+1}) | X_t = \boldsymbol{x}\right] \tag{4}$$

Once the optimal policy has been found, the MDP has been solved. In general, there are many different approaches to solving Equation 4. If the system dynamics $\tilde{F}$ and reward function $\tilde{\rho}$ are fully known, Dynamic Programming (DP) techniques such as Value Iteration (VI) and Policy Iteration (PI) can be used [10, 12]. In case a-priori knowledge about the dynamics is not available, these methods are no longer applicable and one has to use data-driven RL techniques such as Monte Carlo (MC) learning or temporal-difference (TD) methods [10]. A central topic underlying these data-driven approaches is the careful trade-off between exploitation of intermediate policies to further improve learned behavior, and exploration to ensure that all parts of the state space are visited sufficiently often. Within the context of optimal adaptive control, Adaptive/Approximate Dynamic Programming (ADP) [23, 24] techniques are often adopted, due to the general need to approximate the value function and policy. In this view, Adaptive Critic Designs (ACDs) [25, 26] are frequently used to enable data-driven identification of the optimal control policy.

B. Curriculum Learning

The ability of an RL algorithm to converge towards an optimal control policy for a given MDP is naturally related to the complexity of the task at hand [15]. Task complexity covers multiple aspects of a task, such as dimensionality of the state space $\mathcal{X} \subset \mathbb{R}^n$ and input space $\mathcal{U} \subset \mathbb{R}^m$, the nature of the dynamics $\tilde{F}$, or richness of the reward signal. However, a key insight is that task complexity does not exclusively define learning performance, but that there is a strong link with the learning abilities of the agent itself. This understanding gives rise to the idea of adjusting the complexity of a given learning task to the agent's competencies, in order for it to learn the desired behavior more effectively [15, 22]. In this way, a sequence of intermediate training distributions can be constructed with the goal of increasing the tractability of the learning process. This concept is used frequently in studies on animal training, where it is known as 'shaping' [14, 27]. In machine learning, the shaping sequence is referred to as a learning curriculum [15].

The concept of curriculum learning appears in various forms in a range of different fields. A key contribution was made by Bengio et al. [14] in the area of supervised learning with deep neural networks. Here, the authors successfully demonstrated effective learning curricula for shape recognition and language modeling. The proposed framework is based on the idea that for non-convex optimization tasks, the order in which training samples are presented can be used to avoid local minima [14, 28]. It is argued that by presenting the learner with an appropriate sequence of samples, the

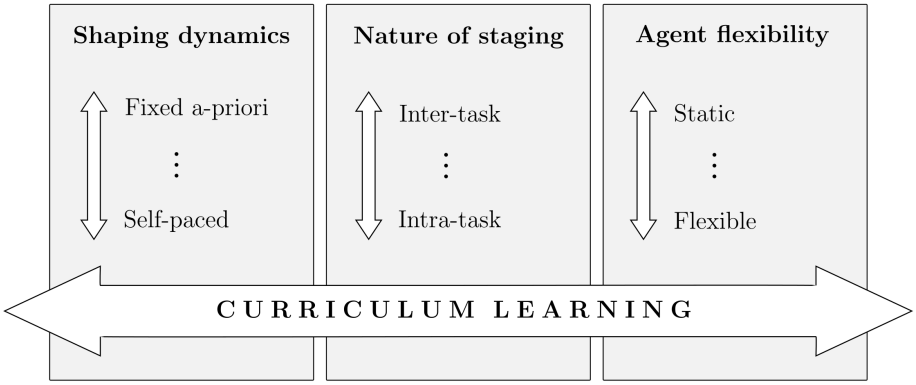


Fig. 2 Visual representation of the curriculum learning taxonomy

solution is drawn towards a region of the parameter space that is in the vicinity of the global optimum. This effectively results in a 'smoothing' of the non-convex optimization criterion. Additionally, the curriculum learning framework was also conceived in the field of transfer learning (TL) [15], which advocates that knowledge learned in one MDP can be applied in a different, but related MDP [22]. This concept can be extended to a learning sequence, where in each stage knowledge is transferred between a pair of tasks. This extends the motivation for curriculum learning beyond non-convex optimization problems. Moreover, it implies that the shaping aspect of a learning curriculum is not limited to the training distribution only, but also resides in the agent representation. This interpretation of curriculum learning can be recognized in the work by Helmer et al. [21] on flexible heuristic dynamic programming.

Although the motivations for applying curriculum learning for supervised learning or transfer learning may be different, they share a clear common philosophy. In the authors' view, the curriculum learning paradigm can be decomposed along three dimensions, as shown in Figure 2. The first element is referred to as the *shaping dynamics*. In general, a learning curriculum can be fully determined a-priori [14], or may arise from online feedback of the capabilities of the agent during the learning process [28, 29]. Curriculum learning of the first form is based on a fixed sequence of training distributions selected heuristically by a domain expert. This is an intuitive way of curriculum design and allows a domain expert to embed prior knowledge in the learning process. However, such a ranking of training distributions can be difficult to provide a-priori if it is not clear what determines the learning complexity. Moreover, feedback about the learning progress is not used. This gives rise to the philosophy of letting a learning curriculum be generated dynamically by the learner itself [28] or some agent-specific teacher [30]. This concept is known as *self-paced* learning (SPL) [28, 29]. However, SPL suffers from other problems that need to be adequately addressed.

The second dimension will be called the *nature of staging*. The main categories considered here will be designated as *inter-task* and *intra-task* learning, referring to the notion of learning within or across a (set of) MDP(s). In intra-task learning, the system state transition function (dynamics) of the target MDP $M : \langle \mathcal{X}, \mathcal{U}, \tilde{F}, \tilde{\rho}, \gamma \rangle$ is shared by any of the intermediate MDPs $M_\lambda : \langle \mathcal{X}_\lambda, \mathcal{U}_\lambda, \tilde{F}, \tilde{\rho}_\lambda, \gamma_\lambda \rangle$ that comprise the learning curriculum. This implies that complexity management is limited to strict subsets of the target state and input space and designating alternative forms of the performance index. In case the dimensionality of the state space is used as a shaping factor, an intra-task learning curriculum requires the dynamic modes associated with these states to be fully decoupled. This contrasts with inter-task learning, where the state transition function is allowed to change across the shaping sequence. Effective inter-task learning requires a high degree of correlation across tasks to ensure a positive learning effect.

The third aspect curriculum learning mentioned here relates to the flexibility of the agent domain, and will be referred to as the *agent flexibility*. In broad terms, one can make a distinction between static and dynamic agent representations. With dynamic agent representations, one can opt to apply a mapping or to switch altogether between different representations. This can relate to different types of information, such as the value function, the policy, or an internal model. Depending on the type of function approximator, dedicated techniques are required to achieve effective knowledge transfer. This is again a major topic in the field of transfer learning [22, 31].

C. Safe Learning

The application of RL in its basic form is troublesome if the process or system that is to be interacted with is safety-critical in nature. As argued by García and Fernández [16], the problem of safety may appear in both the exploitation and exploration phases of learning. The objective of maximizing or minimizing the expected discounted return may not be separable from occasional unsafe or risky transitions for some environments, whereas the random element that underlies the exploration phase may inadvertently lead the system to a dangerous state (i.e., one for which the probability of recovery equals zero). Therefore, dedicated measures are required [16, 18, 19, 32–36].

Independent of the nature of control policy, online autonomous interaction with safety-critical systems requires measures to keep the system state $\boldsymbol{x}_t$ bounded to a safe subset of the complete state space $\mathcal{X}_{safe} \subset \mathcal{X} \; \forall \; t \in [t_0, \infty\rangle$. In systems theory, a topic that captures this objective very effectively is known as Lyapunov stability [6, 37]. In Lyapunov stable systems, any trajectory starting from a state $\boldsymbol{x}_t$ remains bounded to the safe set $\mathcal{X}_{safe}$ if there exists a scalar continuous function known as a (control) Lyapunov function $v(\boldsymbol{x}_t)$ over the domain spanned by $\mathcal{X}_{safe}$. For systems that are stabilizable but do not possess Lyapunov stable internal dynamics, a stabilizing policy can be derived directly from a known control Lyapunov function (CLF) by rendering its derivative negative semi-definite. This is a very useful property that enables the design safe control policies for both linear and nonlinear systems [6, 37].

Lyapunov stability in its basic form does not consider state- or input-constrained systems or systems with bounded uncertainties or disturbances. Nevertheless, the concept can also be extended to this class of systems. Dedicated CLFs can be found that prevent constraint violation or ensure robust stabilizability. For example, one can adopt Barrier Lyapunov functions [38, 39] or Robust CLFs [37] to design control policies that meet these objectives. This shows that, in theory, the issue of safety during learning can be completely resolved once an appropriate CLF has been obtained.

However, despite their appealing properties, Lyapunov functions may not be straightforward to find for general nonlinear systems [6]. If not available, there are no guarantees that the system state remains bounded to the safe subspace $\mathcal{X}_{safe}$. In the absence of Lyapunov functions, alternative techniques are required that provide guarantees on the existence of a control strategy that ensures boundedness of state- and input-constrained systems. A central topic here is known as ergodicity [34]. If the condition of ergodicity applies, it is with certainty that some trajectory exists between any arbitrary pair of states in the safe subspace $\mathcal{X}_{safe}$. Ensuring ergodicity requires an internal belief, or model, of the system dynamics, as well as limited sensing capabilities for detecting surrounding states [17, 34]. This makes the process inherently model-based. Consequently, instead of starting out with an initial admissible policy or initial value function in the form of a CLF, safety can be guaranteed by virtue of a prior model only. This model can feature bounded uncertainties, as long as the actual system dynamics are within these bounds [17, 18]. Moreover, the prior model may be updated online to enable more effective verification of the ergodicity condition.

Although the availability of an internal model is key to the ergodicity argument, the question remains how verification of ergodicity can be achieved. One approach is to find a global backup policy that maximizes the safe subspace $\mathcal{X}_{safe}$ under the belief [34]. From a control-theoretic perspective, it can be argued that the key principles underlying this concept show correspondences to robust stabilizability. Another method is to find local backup policies that render a given state to the vicinity of another state that has been visited before. This requires online generation of trajectories based on the internal model. A heuristic approach to this technique is presented by Mannucci et al. as the Safety Handling Exploration with Risk Perception Algorithm (SHERPA) [17, 18].

D. Safe Curriculum Learning

Considering the curriculum learning paradigm, the fundamental concepts of (robust) Lyapunov stability and ergodicity retain a central position. Under minimum robustness requirements, a learning curriculum can be exploited to enhance learning safety in a similar manner as it improves learning efficiency. That is, for any intermediate MDP $M_\lambda : \left\langle \mathcal{X}_\lambda, \mathcal{U}_\lambda, \tilde{F}_\lambda, \tilde{\rho}_\lambda, \gamma_\lambda \right\rangle$ in the shaping sequence, boundedness of the system state $\boldsymbol{x}_t$ to a safe subset $\mathcal{X}^{(\lambda)}_{safe} \subset \mathcal{X}_\lambda \; \forall \; t \in [t_0, \infty\rangle$ can be guaranteed if a safe control policy $\pi^{(\lambda-1)}_{safe}(\boldsymbol{x}_t)$ can be safely re-used from a predecessor MDP $M_{\lambda-1} : \left\langle \mathcal{X}_{\lambda-1}, \mathcal{U}_{\lambda-1}, \tilde{F}_{\lambda-1}, \tilde{\rho}_{\lambda-1}, \gamma_{\lambda-1} \right\rangle$. Of central importance are the risk profile and system dynamics across the shaping sequence. For true safety, the complete risk profile should always be available at any stage in the curriculum. That is, if it is not specified a-priori that $\left\{\mathcal{X}^{\lambda}_{safe}\right\}^{\complement} \neq \left\{\mathcal{X}^{\lambda-1}_{safe}\right\}^{\complement}$, safety cannot be guaranteed when solving M_λ.

The central concept can best be described from a control theoretic perspective. For example, an agent policy $\pi_{\lambda-1}(\boldsymbol{x}_t)$ that stabilizes the system $\boldsymbol{x}_{t+1} = f_{\lambda-1}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ in $\mathcal{X}^{\lambda-1}_{safe}$ can be safely re-used if the successor system $\boldsymbol{x}_{t+1} = f_\lambda(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is also Lyapunov stable in $\mathcal{X}^{\lambda}_{safe}$ under autonomy of (a transformed form of) $\pi_{\lambda-1}(\boldsymbol{x}_t)$. The question of whether a transformation of $\pi_{\lambda-1}(\boldsymbol{x}_t)$ is required depends on the relative topology and dimensionality of the state and input

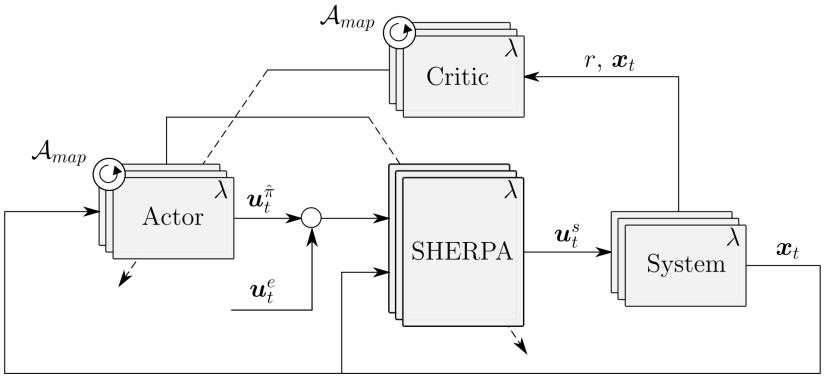


Fig. 3 Block diagram of the safe curriculum optimal control framework

spaces: in case $\mathcal{X}_\lambda \neq \mathcal{X}_{\lambda-1}$ or $\mathcal{U}_\lambda \neq \mathcal{U}_{\lambda-1}$, safe re-use of $\pi_{\lambda-1}(\boldsymbol{x}_t)$ requires an appropriate mapping $\mathcal{A}_{map}$ of $\pi_{\lambda-1} : \mathcal{X}_{\lambda-1} \mapsto \mathcal{U}_{\lambda-1}$ to $\pi_\lambda : \mathcal{X}_\lambda \mapsto \mathcal{U}_\lambda$ [22, 31, 40]. This directly relates to the agent flexibility aspect of safe curriculum learning. In general, construction of a safe $\mathcal{A}_{map}$ requires an additional source of domain knowledge.

The same elemental considerations apply to the verification of ergodicity. However, the internal model that is used for verification should be representative for every stage in the MDP. Theoretically, the availability of this model implies that safe learning in MDP M_λ can be fully achieved without a prior safe control policy $\pi_{safe}^{(\lambda-1)}(\boldsymbol{x}_t)$. However, this prior policy can still be exploited to expedite the search for ergodicity-preserving policies. This is particularly beneficial for online identification of control sequences between a given state and a state visited earlier if the subspace of ergodicity-preserving policies is relatively small compared to the total policy space. It can be argued that the latter serves as a measure of learning complexity in the context of safety. In this case, (a mapped variant of) $\pi_{safe}^{(\lambda-1)}(\boldsymbol{x}_t)$ can serve as a means to bias the search for backup policies. This again stems from the insight that for autonomous dynamic systems, ergodicity is very much in line with the concept of robust Lyapunov stability.

III. Methodology

In order to demonstrate the safe curriculum paradigm in the context of optimal control, several instruments are required. The schematic illustrated in Figure 3 shows a generic picture of the safe curriculum control framework considered in this study. A fixed learning curriculum consisting of a sequence of MDPs $M_\lambda : \langle \mathcal{X}_\lambda, \mathcal{U}_\lambda, f_\lambda, r_\lambda, \gamma_\lambda \rangle$ describing a collection of dynamic (sub)-systems and utility functions is considered. An intelligent flexible learning agent represented by a collection of feedforward neural network control policies takes a central role in the learning architecture. A collection of critic networks are used to approximate the value function. Because of the flexible nature of the agent and the changing dimensionality of $\mathcal{X}_\lambda$ and $\mathcal{U}_\lambda$ across the curriculum, an a-priori designed mapping strategy $\mathcal{A}_{map}$ is adopted to transfer the information encoded in the actor and critic networks across successive learning stages. The learning architecture is completed by the SHERPA ergodicity-preserving safety filter [17, 18], which overrides any inputs under which the ergodicity condition cannot be maintained.

The approximate dynamic programming approach used to train the agent is presented in Section III.A, which is followed by a description of the neural-network based actor-critic setup in Section III.B. The subject of task network mapping strategies is considered in Section III.C. Finally, a basic overview of the SHERPA safety filter is provided in Section III.D, together with a description of how the learning curriculum is exploited to improve its performance.

A. Learning Framework

In general, ADP schemes can be applied to find optimal control policies for a wide class of dynamical systems, including those that are time-varying, nonlinear, or stochastic in nature [23]. Here the scope will be limited to time-invariant, deterministic, control-affine, nonlinear systems of the following form:

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t) + G(\boldsymbol{x}_t)\boldsymbol{u}_t \tag{5}$$

The discrete-time formulation of the system dynamics makes the optimal control problem consistent with the digital control framework. Under the assumption that the system is Lipschitz continuous and that is it stabilizable, an optimal feedback policy $\hat{\pi}_*(\boldsymbol{x})$ can be found that minimizes the expected (discounted) utility accumulated over time [23, 41]. In

this research, the scope will be limited to optimal regulation, which implies that the goal is to bring all states of the system to zero such that the performance index is optimized. However, the problem formulation can be readily extended to tracking control as well. In general, the performance index can take any shape, but it is often defined as a utility that is quadratic in the state and input variables [23, 42]:

$$r(\boldsymbol{x}_t, \boldsymbol{u}_t) = \boldsymbol{x}_t^T Q \boldsymbol{x}_t + \boldsymbol{u}_t^T R \boldsymbol{u}_t \tag{6}$$

A similar utility can be specified based on observed outputs. Since $\lim_{t\to\infty} r(\boldsymbol{x}_t, \boldsymbol{u}_t) = 0$ under an admissible control policy, the following (undiscounted) definition of the value function applies [41, 42]:

$$V_\pi(\boldsymbol{x}) \doteq \sum_{t=0}^{\infty} r(\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{7}$$

In this case, no discounting of the expected future return is required, i.e. $\gamma = 1$. Consequently, the optimal value function can be written as follows:

$$V_*(\boldsymbol{x}_t) = \min_{\boldsymbol{u}} \left[r(\boldsymbol{x}_t, \boldsymbol{u}_t) + V_*(\boldsymbol{x}_{t+1}) \right] \tag{8}$$

In optimal control theory and ADP, this equation is also known as the discrete-time Hamilton-Jacobi-Bellman (HJB) equation [41, 42]. Subsequently, the optimal control policy $\pi_*(\boldsymbol{x})$ can be derived directly from the discrete-time formulation of the system dynamics. Setting $\frac{\partial V_*(\boldsymbol{x}_t)}{\partial \boldsymbol{u}_t} = 0$ and expanding the partial derivative using the chain rule, the following is obtained [42, 43]:

$$\boldsymbol{u}^*(\boldsymbol{x}_t) = -\frac{1}{2} R^{-1} G^T(\boldsymbol{x}_t) \frac{\partial V_*(\boldsymbol{x}_{t+1})}{\partial \boldsymbol{x}_{t+1}} \tag{9}$$

The discrete-time HJB equation is solved online through Heuristic Dynamic Programming (HDP) based on Generalized Policy Iteration (GPI). In GPI, the policy evaluation step only consists of a finite number of K iterations towards the true value function $V_\pi(\boldsymbol{x})$ [42]. This makes it a compromise between Value Iteration (VI) and Policy Iteration (PI), for which $K = 1$ and $K \to \infty$, respectively. Whereas GPI is less sample-efficient compared to PI, this comes at the benefit of not needing an initial admissible control policy. Consequently, for $k = 0, 1, \ldots, K-1$, the policy evaluation step takes the following form:

$$\hat{V}_{\pi_j}^{(k+1)}(\boldsymbol{x}_t) = r(\boldsymbol{x}_t, \boldsymbol{u}_t) + \hat{V}_{\pi_j}^{(k)}(\boldsymbol{x}_{t+1}) \tag{10}$$

Here, $\hat{V}_{\pi_j}^{(\bullet)}(\boldsymbol{x}_t)$ represents a compact approximation of the true intermediate value function in the form of a critic network. The policy update step follows directly from Equation 9:

$$\boldsymbol{u}^{(j+1)}(\boldsymbol{x}_t) = -\frac{1}{2} R^{-1} G^T(\boldsymbol{x}_t) \frac{\partial \hat{V}_{\pi_j}^{(K)}(\boldsymbol{x}_{t+1})}{\partial \boldsymbol{x}_{t+1}} \tag{11}$$

In principle, explicit evaluation of this equation yields the optimal control directly without the need for a separate actor network. However, this requires a prediction of the future state $\boldsymbol{x}_{t+1}$ based on a full model of the system dynamics. A separate representation of the control policy omits this requirement, as proposed by [43–45], but the policy update step is still dependent on complete knowledge of the input dynamics $G(\boldsymbol{x}_t)$. Although this term could be learned from online interaction as well [46, 47], in this work it is assumed that this term is available a-priori.

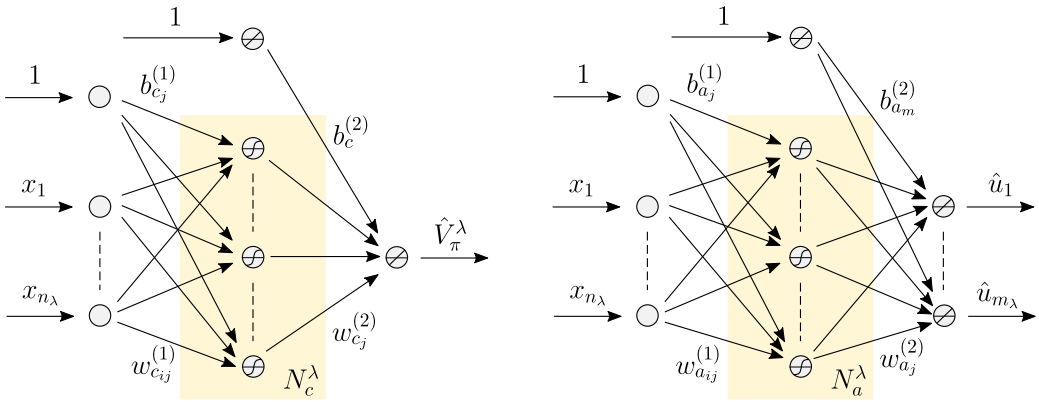
B. Actor-Critic Design and Training

The actor and critic have both been modeled as two-layer feedforward artificial neural networks (ANNs) with a single hidden layer and additional bias nodes. As illustrated in Figure 4a, the critic takes the system state $\boldsymbol{x}_t \in \mathcal{X} \subset \mathbb{R}^{n_\lambda}$ and returns an approximate estimate of the value function $\hat{V}_\pi^{(k)}(\boldsymbol{x}_t)$ based on a forward pass through the network:

$$\hat{V}_\pi^{(k)}(\boldsymbol{x}_t) = \sum_{j=1}^{N_c^\lambda} w_{c_j}^{(2)}(k) \phi_{c_j} \left(\sum_{i=1}^{n_\lambda} w_{c_{ij}}^{(1)}(k) x_i + b_{c_j}^{(1)}(k) \right) + b_c^{(2)}(k) \tag{12}$$

Similarly, the actor network maps the system state $\boldsymbol{x}_t \in \mathcal{X} \subset \mathbb{R}^{n_\lambda}$ towards the greedy control input $\pi(\boldsymbol{x}_t) \in \mathcal{U} \subset \mathbb{R}^{m_\lambda}$:

$$\hat{u}_m^{(k)}(\boldsymbol{x}_t) = \sum_{j=1}^{N_a^\lambda} w_{a_{mj}}^{(2)}(k) \phi_{a_j} \left(\sum_{i=1}^{n_\lambda} w_{a_{ij}}^{(1)}(k) x_i + b_{a_j}^{(1)}(k) \right) + b_{a_m}^{(2)}(k) \tag{13}$$



(a) Critic **(b) Actor**

Fig. 4 Generic neural network architectures at learning stage λ

The generic architecture of the actor network is also visualized in Figure 4b. For both the actor and critic networks, hyperbolic tangent functions $\phi_{c_j}(\bullet) = \phi_{a_j}(\bullet) = \tanh(\bullet)$ have been selected as the activation functions in the hidden layer. This type of functions is very suitable for approximating the value function and policy, thanks to their global character and the fact that they are continuously differentiable. The critic is trained through bootstrapping by minimizing the Bellmann or temporal-difference (TD) error [42, 45]:

$$e_c^{(k)}(\boldsymbol{x}_t) = \hat{V}_{\pi_j}^{(k-1)}(\boldsymbol{x}_t) + r(\boldsymbol{x}_t, \boldsymbol{u}_t) - \hat{V}_{\pi_j}^{(k)}(\boldsymbol{x}_{t+1}) \tag{14}$$

Adjustment of the critic weights is achieved through backpropagation based on the Levenberg-Marquadt gradient descent algorithm, which generally results in much faster convergence compared to first-order optimization methods [23]. The loss function is taken as the quadratic TD-error:

$$E_c^{(k)}(\boldsymbol{x}_t) = \tfrac{1}{2} e_c^{(k)}(\boldsymbol{x}_t)^2 \tag{15}$$

The gradient for each weight can be computed analytically as follows:

$$\frac{\partial E_c^{(k)}(\boldsymbol{x}_t)}{\partial w_{c_j}^{(2)}(k)} = \frac{\partial E_c^{(k)}(\boldsymbol{x}_t)}{\partial e_c^{(k)}(\boldsymbol{x}_t)} \frac{\partial e_c^{(k)}(\boldsymbol{x}_t)}{\partial \hat{V}_{\pi_j}^{(k)}(\boldsymbol{x}_t)} \frac{\partial \hat{V}_{\pi_j}^{(k)}(\boldsymbol{x}_t)}{\partial w_{c_j}^{(2)}(k)} = e_c^{(k)}(\boldsymbol{x}_t) \tanh\left(\sum_{i=1}^{n_\lambda} w_{c_{ij}}^{(1)}(k) x_i + b_{c_j}^{(1)}(k)\right) \tag{16}$$

$$\frac{\partial E_c^{(k)}(\boldsymbol{x}_t)}{\partial w_{c_{ij}}^{(1)}(k)} = \frac{\partial E_c^{(k)}(\boldsymbol{x}_t)}{\partial e_c^{(k)}(\boldsymbol{x}_t)} \frac{\partial e_c^{(k)}(\boldsymbol{x}_t)}{\partial \hat{V}_{\pi_j}^{(k)}(\boldsymbol{x}_t)} \frac{\partial \hat{V}_{\pi_j}^{(k)}(\boldsymbol{x}_t)}{\partial \phi_{c_j}(k)} \frac{\partial \phi_{c_j}(k)}{\partial w_{c_{ij}}^{(1)}(k)} = e_c^{(k)}(\boldsymbol{x}_t) w_{c_j}^{(2)}(k) \left(1 - \tanh^2\left(\sum_{i=1}^{n_\lambda} w_{c_{ij}}^{(1)}(k) x_i + b_{c_j}^{(1)}(k)\right)\right) x_i \tag{17}$$

The same derivation applies to the bias terms. The actor network is trained to approximate the greedy input from Equation 11, as proposed by [23, 45]

$$\boldsymbol{e}_a^{(j+1)}(\boldsymbol{x}_t) = \boldsymbol{u}^{(j+1)}(\boldsymbol{x}_t) - \hat{\boldsymbol{u}}^{(j+1)}(\boldsymbol{x}_t) \tag{18}$$

where:

$$\boldsymbol{u}^{(j+1)}(\boldsymbol{x}_t) = -\frac{1}{2} R^{-1} G^T(\boldsymbol{x}_t) W_c^{(2)}(K) \operatorname{diag}\left(1 - \tanh^2\left(W_c^{(1)}(K)\boldsymbol{x}_{t+1} + \boldsymbol{b}_c^{(1)}(K)\right)\right) W_c^{(1)}(K) \tag{19}$$

with $W_c^{(1)}(K)$, $W_c^{(2)}(K)$, and $\boldsymbol{b}_c^{(1)}(K)$ representing the network weights and bias from policy evaluation step K in matrix and vector form, respectively. The loss function defined as the square of this difference:

$$E_a^{(j+1)}(\boldsymbol{x}_t) = \tfrac{1}{2} \boldsymbol{e}_a^{(j+1)}(\boldsymbol{x}_t)^2 \tag{20}$$

The gradients can be derived analogously as for the critic. Training is performed in batch format for every intermediate policy $\pi_j(\boldsymbol{x}_t)$ based on a collection of visited states $\mathcal{X}_{visit_{\pi_j}}^\lambda \subset \mathcal{X}_{safe}^\lambda \subset \mathcal{X}^\lambda$. This is done after online interaction with the system. This implies that training is not performed online, but in-between online exploitation cycles.

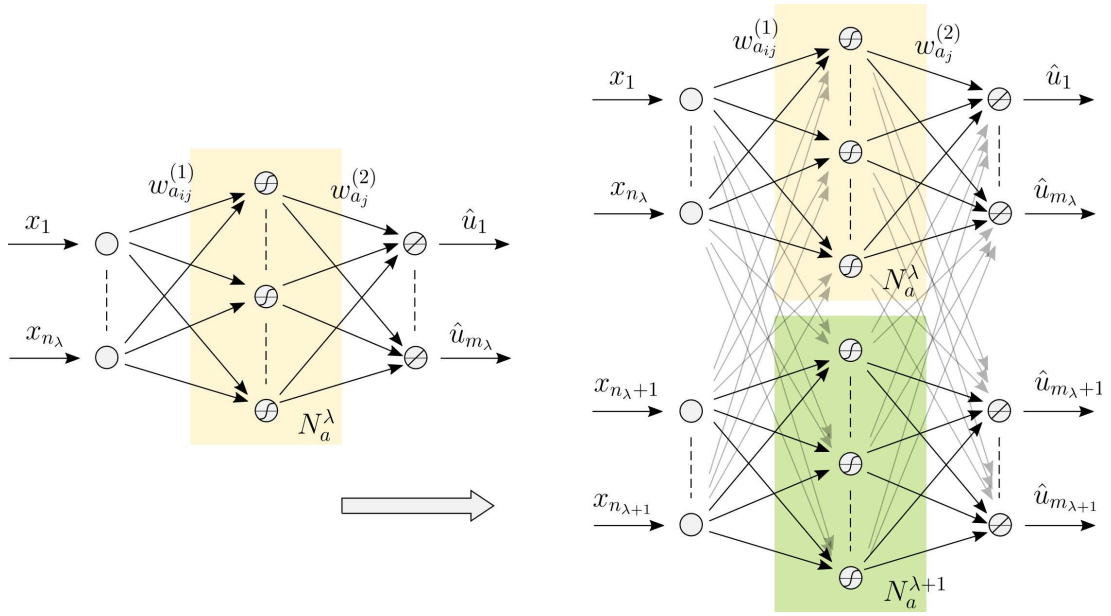


Fig. 5 Greedy network mapping strategy for the actor network; bias nodes not shown for clarity

C. Task Network Mappings

The actor and critic networks represent valuable knowledge that is to be built upon as the learning curriculum progresses. Depending on the relatively topology, dimensionality, and semantics of the related MDPs, appropriate mappings $\mathcal{A}_{map}$ need to be applied to transfer this knowledge across successive learning stages. This concept takes a central position in the field of transfer learning, and was introduced by Taylor et al. [31] in the domain of reinforcement learning. Inspired by this work, task network mappings are adopted in the proposed safe curriculum control setup to map the ANN weights and biases between successive representations of the actor and the critic.

As explained in Section II.D, a primary objective for these mappings is that the dynamic system is stable under autonomy of the mapped policy $\hat{\pi}_0^{(\lambda)}(\boldsymbol{x_t})$. This implies that from the perspective of safety, any mapping can be applied that ensures the absence of unstable modes. In case of systems that are naturally stable, this means that the zero mapping is sufficient to satisfy this condition. A secondary objective is that learning efficiency should be enhanced, which opens many possibilities for designing $\mathcal{A}_{map}$ [21, 31]. An intuitive approach is to directly copy weights to network links that are semantically similar, which is designated as *greedy* mapping. For example, in the case of the actor, this is in line with the idea that similar policies should be adopted for any new states $\boldsymbol{x}_t^{'} = \text{proj}_{\mathcal{X}^{\lambda+1}\setminus\mathcal{X}^\lambda}(\boldsymbol{x}_t)$ that are governed by similar dynamics as any states counteracted before. For the hidden-layer weight matrix $W_a^{(1)}(\bullet)$, this implies that the following operation is applied:

$$\begin{bmatrix} w_{a11}^{(1)} & \cdots & w_{a1n^\lambda}^{(1)} \\ \vdots & \ddots & \vdots \\ w_{aN_a^\lambda 1}^{(1)} & \cdots & w_{aN_a^\lambda n^\lambda}^{(1)} \end{bmatrix} \mapsto \begin{bmatrix} w_{a11}^{(1)} & \cdots & w_{a1n^\lambda}^{(1)} & & & \\ \vdots & \ddots & \vdots & & & \\ w_{aN_a^\lambda 1}^{(1)} & \cdots & w_{aN_a^\lambda n^\lambda}^{(1)} & & & \\ & & & w_{a(N_a^\lambda+1)(n^\lambda+1)}^{(1)} & \cdots & w_{a(N_a^\lambda+1)n^{\lambda+1}}^{(1)} \\ & & & \vdots & \ddots & \vdots \\ & & & w_{aN_a^{\lambda+1}(n^\lambda+1)}^{(1)} & \cdots & w_{aN_a^{\lambda+1}n^{\lambda+1}}^{(1)} \end{bmatrix} \tag{21}$$

This process is also visualized by Figure 5 for further illustration. The same approach is also used for the output layer weights $W_a^{(2)}(\bullet)$ and the bias terms. For the critic network, the process is identical. The greedy mapping strategy works well for the experiments discussed in Sections IV and V. However, adequate design of $\mathcal{A}_{map}$ is highly problem-dependent, which implies that other strategies are generally required for other types of curricula. This requires some minimum domain knowledge, but this is commonly available when facing an optimal control problem.

D. Safety Mechanism

The quadratic design of the utility function specified by Equation 6 implies that safety is not encoded in the bare learning framework. To this end, an external local ergodicity-preserving safety filter is adopted, which omits the need for global safe backup policies to be available a-priori. Under authority of this filter, not only remains the system state bounded under unstable agent policies, but the use of safe online exploration/exploitation cycles also becomes a possibility. To keep learning going throughout the safe subset $\mathcal{X}^{\lambda}_{safe} \subset \mathcal{X}^{\lambda}$, random exogenous exploration inputs $\boldsymbol{u}^{e}_{t}$ are applied on an occasional basis to excite the dynamics of the system. By virtue of the safety filter, it is ensured that these exploration inputs do not drive the system state outside of the safe set. As explained in Section II.C, this requires some form of (approximate) predictive model to be available. Also, any transitions generated under authority of the safety filter are off-policy, and can therefore not be used for training in the on-policy HDP learning scheme.

The safe curriculum control architecture proposed in this study adopts the Safety Handling Exploration with Risk Perception Algorithm (SHERPA), which was developed by Mannucci et al. [17, 18] in the context of UAV exploration. SHERPA is essentially a heuristic search algorithm, which reduces the knowledge required for safe exploration to an overestimate of the system dynamics and experience gained online by the agent. It is based on the assumption that fatal events are related to fatal states instead of actions, giving rise to the concept of Fatal State Space (FSS) [18]:

$$\mathcal{X}_{fatal} = \left\{ \boldsymbol{x}' | \forall \boldsymbol{x} \in \mathcal{X}, \forall \boldsymbol{u} \in \mathcal{U}, \tau = \left(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}' \right) \in \mathcal{T}_{fatal} \right\} \tag{22}$$

with $\mathcal{T}_{fatal}$ the set of transitions that lead to fatal occurrences. The risk modes are assumed to be known a-priori and collectively form the Restricted State Space (RSS). The intersection of the RSS with the FSS gives rise to the Restricted Fatal State Space (RFSS):

$$RFSS = RSS \cap \mathcal{X}_{fatal} \tag{23}$$

The RFSS is unknown a-priori. However, it is assumed that the algorithm can perceive how close the current system state is with respect to the nearest fatal state. This is referred to as the risk perception capability of the algorithm. Risk perception is defined as a function $W(\boldsymbol{x})$ that returns 1 if risk is detected in a limited perception region H. If no risk is detected, all states in the region H are added to a belief of the safe state space $\tilde{\mathcal{X}}_{safe} \subseteq \mathcal{X}_{safe}$. In this way, the safe exploration area is gradually expanded. The framework is complemented by the concept of Lead-to-Fatal (LTF) states, which are not part of the FSS, but will evolve into the FSS with certainty [18]. The set of LTF states is defined as [18]:

$$L = \{ \boldsymbol{x} | \forall \boldsymbol{u}(\boldsymbol{x}(t_0), t),\ \exists t : \boldsymbol{\sigma}(\boldsymbol{x}(t_0), \boldsymbol{u}(t), t) \in FSS \} \tag{24}$$

With $\boldsymbol{\sigma}(\boldsymbol{x}(t_0), \boldsymbol{u}(t), t)$ representing a state-trajectory between time t and t_0. The key idea is that from any state $\boldsymbol{x}_t$, the agent must be able to return to a state $\boldsymbol{x}_p$ visited before that lies within $\tilde{\mathcal{X}}_{safe} \subseteq \mathcal{X}_{safe}$, i.e. it must avoid both fatal states as well as lead-to-fatal states. The predictive capabilities of the SHERPA algorithm are based on an overestimate of the system dynamics, referred to as an internal bounding model. Given the total set of transitions $\mathcal{T}$ governed by the dynamics of the system described by Equation 5, the bounding model $\boldsymbol{\Delta}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ ensures the following [18]:

$$\forall \tau = \left\{ \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}' \right\} \in \mathcal{T} : \boldsymbol{x}' \in \boldsymbol{\Delta}(\boldsymbol{x}, \boldsymbol{u}) \tag{25}$$

With this approach, if all trajectories τ predicted by the bounding model lie within $\tilde{\mathcal{X}}_{safe} \subseteq \mathcal{X}_{safe}$ and do not include any LTF states, safety of the system can be guaranteed with probability one. The SHERPA algorithm integrates all of these aspects in its search for control backups, which enables the agent to explore indefinitely. If the predicted distribution of states following a proposed input $[\boldsymbol{x}_{t+1}] = \boldsymbol{\Delta}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is completely in the known safe state space, safe return to the neighborhood of a state $\boldsymbol{x}_p$ visited earlier is guaranteed if a control backup sequence $\boldsymbol{U}_b(t)$ can be found for $[\boldsymbol{x}_{k+1}]$ [18]. A strictly feasible control sequence $\boldsymbol{U}_b = \{\boldsymbol{u}_t, \ldots, \boldsymbol{u}_{t+m}\}$ is a control backup if [18],

$$\forall i \in \{1, 2, \ldots, m\}, [\boldsymbol{x}_{t+i}] \subset \tilde{\mathcal{X}}_{safe_t} \tag{26}$$

and $[\boldsymbol{x}_{k+t}] \cap L = \varnothing$, i.e.,

$$\forall \boldsymbol{x}_{t+m} \in [\boldsymbol{x}_{t+m}],\ \exists p \leq t : (\boldsymbol{x}_{t+m} - \boldsymbol{x}_p) \in [\boldsymbol{\epsilon}] \tag{27}$$

meaning that the bounded uncertainty cannot exceed a reaching interval $[\boldsymbol{\epsilon}]$. This reaching interval forms the closeness condition that must be satisfied by the backup sequence, and must be chosen heuristically.

Following the insights from Section II.D, the search for $\boldsymbol{U}_b(t)$ can be expedited by operating on the the bounding model $\boldsymbol{\Delta}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ in a closed-loop form, i.e. under the authority of the agent policy. Using the task network mapping

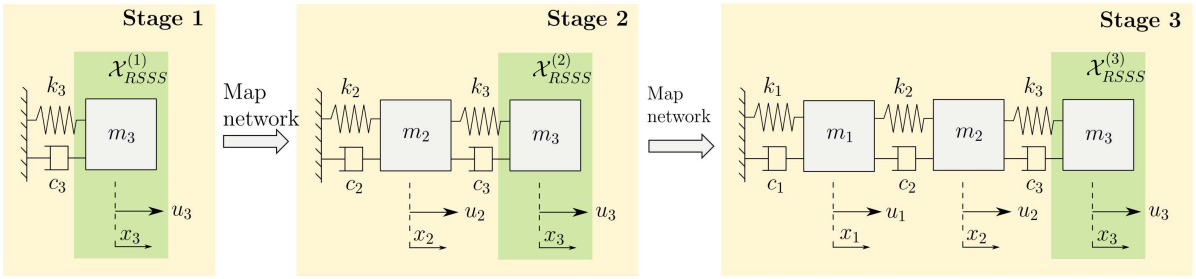


Fig. 6 Inter-task curriculum setup for the cascaded mass-spring-damper optimal regulation experiment

presented in the Section III.C , the learning curriculum serves to reinforce this aspect of the SHERPA algorithm. The bounding model is then adapted with every policy update and learning stage, and can therefore be denoted as $\boldsymbol{\Delta}^{\lambda}_{\pi_j}(\boldsymbol{x}_t, \boldsymbol{u}_t)$.

SHERPA has been developed based on Interval Analysis (IA) techniques [17, 18]. However, the rapid growth of inclusion intervals under even moderate uncertainties generally leads to conservative trajectory predictions. Therefore, the proposed safe curriculum control framework takes a different approach based on model sampling. This implies the condition expressed by Equation 25 can only be met probabilistically, where the probability of adequate bounding model prediction is directly related to the parametric extremity of the actual system with respect to the total model space. This aspect is taken into account in the experiments discussed in Sections IV and V.

IV. Fundamental Experiments

This section presents the findings from a study based on a cascaded linear mass-spring-damper (MSD) system that has been performed to investigate the proposed safe curriculum control framework. The advantage of this setup is that validation of the results is possible using LQR theory, and that its complexity can be altered in a straightforward manner. For example, one can adjust the dimensionality of the the (observable) state-space, and modify the number of actuators to be controlled by the learning agent. By focusing on this simple system first, valuable experience with the framework can be obtained before proceeding to quadrotor implementation in Section V. The experimental setup is briefly described in Section IV.A, and is followed by a presentation of the results in Section IV.B.

A. Experimental Setup

The experimental setup is illustrated in Figure 6. The target system consists of three masses m_i that are connected along a single dimension via springs with constants k_i and dampers with coefficients c_i. A fixed inter-task learning curriculum has been designed to make the target learning task more tractable, with the dimensionality of the state and action spaces serving as the primary shaping factors. At any stage in the learning curriculum, the actor and critic networks* are scaled accordingly using the greedy task network mapping strategy from Section III.C.

An overview of the parameters defining the MSD sequence is given in Table 1. As a result of the negative value of k_3, the system is inherently unstable. Each mass features a unique actuator, which implies that the system is fully controllable. For the utility function, the following structure applies:

$$r(\boldsymbol{x}_t^{(\lambda)}, \boldsymbol{u}_t^{(\lambda)}) = 25 \left(\mathbf{x}_t^{(\lambda)}\right)^T \mathbf{x}_t^{(\lambda)} + \left(\boldsymbol{u}_t^{(\lambda)}\right)^T \boldsymbol{u}_t^{(\lambda)} \tag{28}$$

where $\mathbf{x}_t^{(\lambda)} \subset \boldsymbol{x}_t^{(\lambda)}$ represents the position vector. Safety resides in the fact that the outer mass m_3 must remain within a given region around its equilibrium point, which is indicated as $\mathcal{X}^{(\lambda)}_{RSSS} = RSS \cap \mathcal{X}^{(\lambda)}_{safe}$. The SHERPA safety filter is equipped with a bounding model of the system that features a ±25% parametric uncertainty range for $\{k_1, k_2, k_3\}$ and $\{c_1, c_2, c_3\}$. The available input authority for backups and exogenous exploration inputs is limited to ±10 N.

Proper configuration of the safe curriculum control setup involves tuning a relatively large number of hyperparameters. These relate primarily to exploration, the ANN training scheme, and the SHERPA safety filter. Online interaction times have been set to $\{120, 180, 240\}$ seconds for every policy exploitation cycle, during which random exogenous

*For this kind of LQR problems, less complex learning strategies based on least-squares regression are generally better suited compared to techniques that adopt gradient descent [42, 44, 48]. This follows from the fact that the optimal value function is quadratic.

Table 1 Cascaded mass-spring-damper system characteristics

Element #	m [kg]	k [N/m]	c [N / ms^{-1}]
1	0.6	5	4
2	1.2	7	2
3	0.8	−3	3

exploration inputs $\boldsymbol{u}_i^e$ are applied intermittently. Due to the fact that data may be scarce in some regions of the state space, especially for learning stages 2 and 3, the number of training epochs is kept at a relatively low value of 10 for every update step in the GPI HDP loop. This prevents the ANN training algorithm from overfitting and destroying the greedy actor-critic relationship. For the policy evaluation step, the number of iterations has been set to $K = 3$. For the actor and critic networks, the number of hidden nodes is set to $\{32, 64, 96\}$. This results in considerable approximation power, although the learning task could also be solved using a much smaller number of nodes. These settings were seen to work sufficiently in practice. Initialization of the network weights and biases is performed on a range between $[-0.01, 0.01]$, which ensures that the actor network approximates the zero policy if learning has not taken place.

For SHERPA, the risk perception limits have been set to ±0.2 m. This implies that the algorithm will only detect the boundary of $\mathcal{X}_{RSSS}^{(3)}$ once the position of the outer mass element comes within a 0.2 m distance. The reaching intervals ϵ_x and $\epsilon_{\dot{x}}$ for position and velocity have been set at 0.1 m and 0.2 m/s, respectively. For the probabilistic bounding model, the number of model samples has been set to 10. The maximum number of iterations for selecting a control action and finding a control backup sequence have been set to 5 and 10, respectively. These settings were selected primarily to keep the algorithm's computational complexity at a minimum[†].

For tuning of the backup search itself, some domain knowledge is required. For this experiment, the minimum and maximum lengths of the backup sequence have been set to 0.2 and 0.6 seconds, respectively. The backup consists of a sequence of input bands that keep the control input constant over a multiple number of time steps, which in this case amounts to 0.2 seconds. In this way, the likelihood that an effective control sequence is identified is increased. Finally, a state will only be added to the internal memory of visited steps every 0.3 seconds in order to prevent excessive memory requirements.

B. Results and Discussion

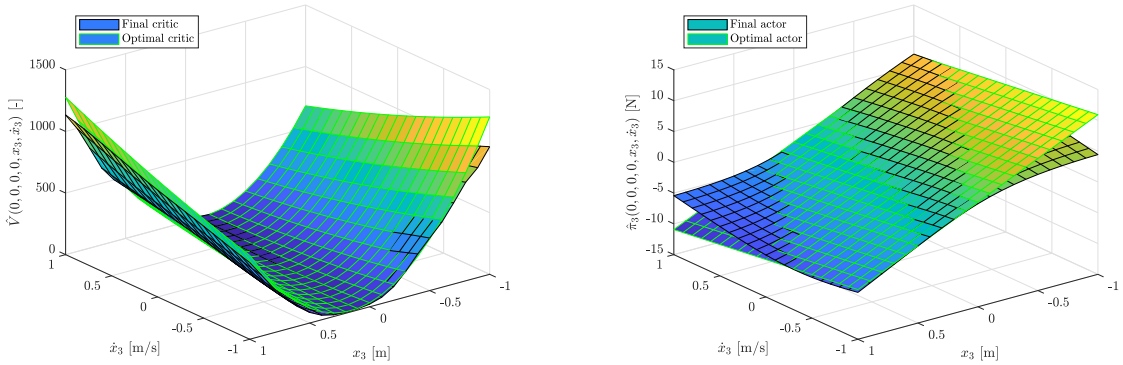
In order to draw sound conclusions on the effectiveness of the safe curriculum learning paradigm, the results of three learning strategies are examined. First, a benchmark case is considered without the use of a learning curriculum or safety filter. In this case, learning takes place directly in the stage 3 target task, and there is nothing that prevents the system from violating the imposed safety constraint. Exploration is achieved by means of random initialization of the system in $\mathcal{X}_{safe}^{(3)}$, which makes the learning problem episodic in nature. Figure 7 visualizes typical cross-sections of the actor and critic networks at the end of learning. These show that the optimal value function and policy that follow from the solution of the discrete-time Algebraic Ricatti Equation (ARE) are approximated quite well, except in the regions where x_3 is in the vicinity of the fatal state space and the velocity vector points towards its direction. This is an intuitive result, as this situation will only rarely (if ever) be encountered under a stable policy.

Figure 8 shows a few learning statistics in terms of the normalized Root Mean Square Error (RMSE) with respect to the optimal control policy and the number of constraint violations. The normalized RMSE is obtained using the Root Mean Square (RMS) control input from the optimal control policy:

$$\overline{RMSE} = \sqrt{\left(\frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{u}^*(\boldsymbol{x}_i) - \hat{\boldsymbol{u}}(\boldsymbol{x}_i)\right)^2\right)} \Big/ \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{u}^*(\boldsymbol{x}_i)\right)^2} \tag{29}$$

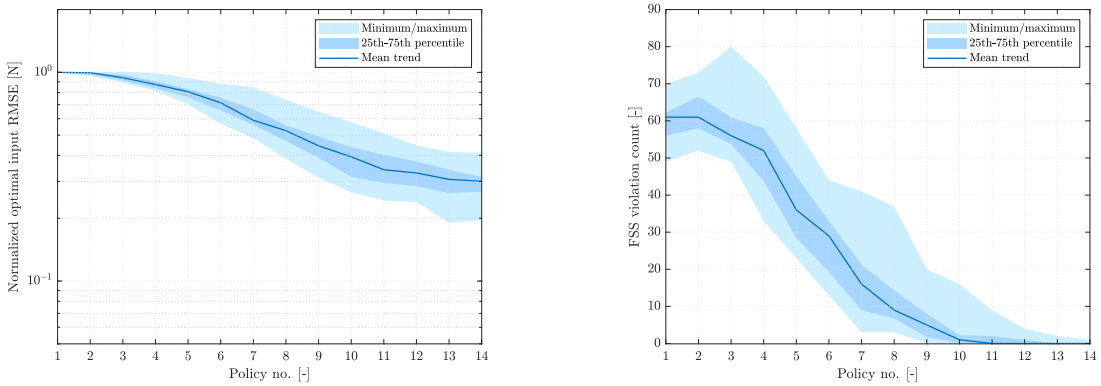
At the start of learning, the normalized RMSE is always equal to 1 as a result of the zero policy. This value reduces with every policy iteration, as shown in Figure 8a. As the actor policy converges towards the optimal control, it is able to stabilize the system. This is reflected in the low number of constraint violations in Figure 8b at the end of learning. However, prior to this point, a significant number of violations is incurred.

[†]Note that the proposed algorithm does not run in real-time.



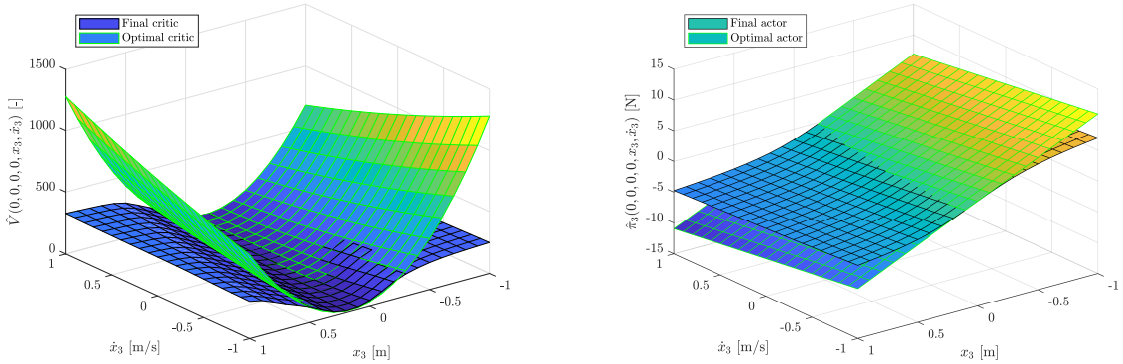
(a) Value function **(b) Policy**

Fig. 7 Example actor and critic outputs at the end of learning; non-curriculum approach, no safety filter



(a) Normalized policy RMSE **(b) Constraint violations**

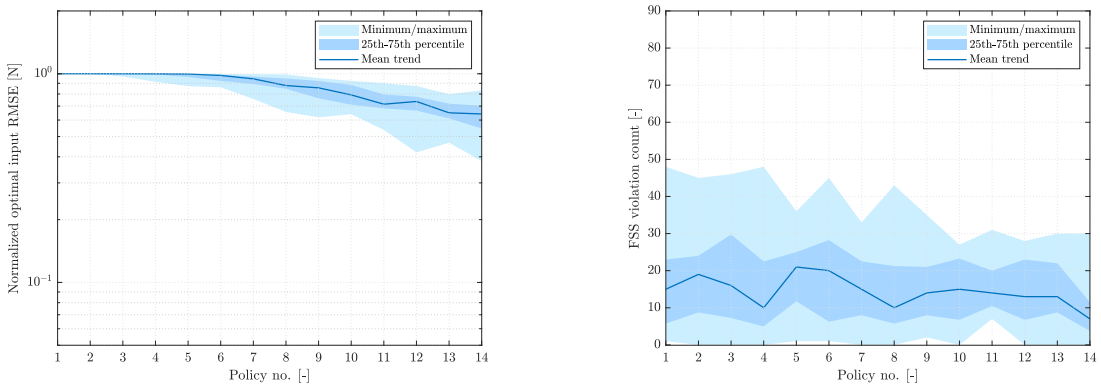
Fig. 8 Learning statistics for 25 independent learning trials; non-curriculum approach, no safety filter



(a) Value function **(b) Policy**

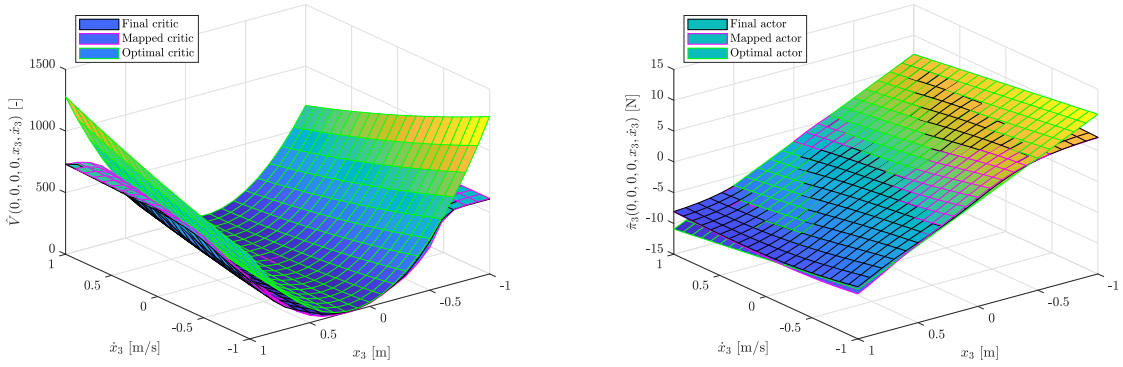
Fig. 9 Example actor and critic outputs at the end of learning; non-curriculum approach, SHERPA enabled

In order to examine the impact of the SHERPA safety filter on the learning performance, a second experiment is performed where SHERPA is enabled in the control loop without the use of a learning curriculum. Typical outputs of



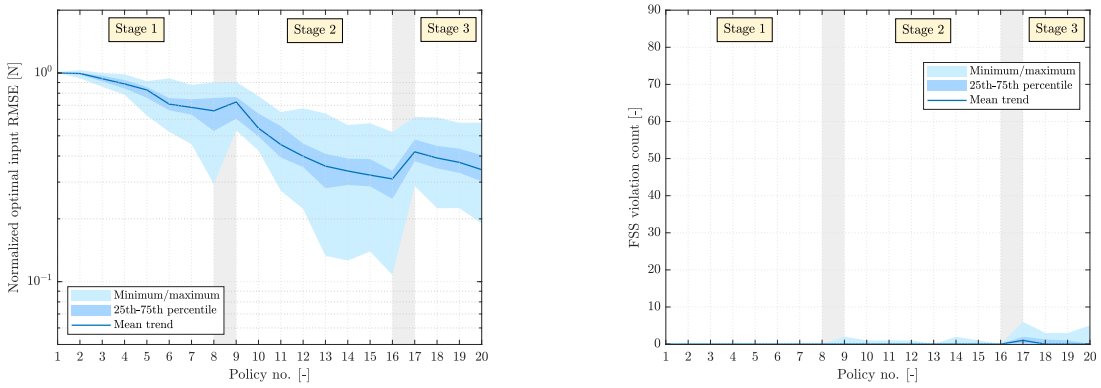
(a) Normalized policy RMSE **(b) Constraint violations**

Fig. 10 Learning statistics for 25 independent learning trials; non-curriculum approach, SHERPA enabled



(a) Value function **(b) Policy**

Fig. 11 Example actor and critic outputs at the end of learning; safe curriculum learning using SHERPA



(a) Normalized policy RMSE **(b) Constraint violations**

Fig. 12 Learning statistics for 25 independent learning trials; safe curriculum learning using SHERPA

the actor and critic networks are visualized in Figure 9. The learning statistics are shown in Figure 10. These results indicate that the learning pace is reduced compared to the non-safe approach. Although the number of constraint

violations is smaller, especially in the early learning phase, SHERPA is often not capable of identifying feasible control backups within the limited number of iterations available. This is a direct consequence of the fact that the subspace of ergodicity-preserving policies is relatively small compared to the total policy space.

The final experiment aims to investigate the extent to which the learning curriculum can expedite the search for ergodicity-preserving policies, and to quantify the effect on learning efficiency. Figure 11 illustrates the actor and critic network outputs at the end of learning, together with the optimal and mapped value functions and policies. This shows that the mapped control policy is already a good approximation of the optimal policy. As a result, both the system itself as well as the bounding model $\boldsymbol{\Delta}^{(3)}_{\pi_0}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ are stable under the transformed predecessor policy. The impact on learning efficiency and safety is illustrated in Figure 12. These results show that the number of constraint violations is reduced considerably with respect to the prior two experiments. In terms of learning efficiency, similar normalized RMSE levels are reached as for the non-safe, non-curriculum approach in the first experiment, although the spread is larger. Note that the particular selection of online interaction times, combined with the total number of policy updates, results in exactly the same number of collected samples for all experiments.

V. Flight Control Application

The results obtained from the cascaded linear mass-spring-damper experiments illustrate the potential of safe curriculum learning in expediting learning efficiency and safety in the context of optimal control. This yields an adequate basis for demonstrating the paradigm for autonomous learning of optimal control laws in the context of flight control. In this section, an experiment is described for optimal attitude regulation of a nonlinear quadrotor model in both pitch and roll axes. This experiment serves primarily as a proof-of-concept. A description of the system dynamics including additional simplifications made is given first in Subsection V.A, which is followed by an overview of the learning framework in V.B. The results are presented in Section V.C.

A. Quadrotor Simulation Model

The system used for this experiment is modeled after a Parrot AR 2.0 drone, and has been of use in earlier RL research as well [21, 49]. Rotational and translational control of the UAV is achieved by changing the rotational velocity of the individual motors. For the translational dynamics, the Earth frame serves as an inertial reference frame with the gravity vector acting perpendicular to the surface. Following Newton's second law and focusing on the body-fixed frame of reference, this implies that the governing equations of motion take the following form:

$$\dot{\boldsymbol{V}}^b_E = \frac{1}{m}\left(\boldsymbol{F}^b_F + \boldsymbol{F}^b_P\right) + \boldsymbol{g}^b - \boldsymbol{\omega}^b_{b/E} \times \boldsymbol{V}^b_E \tag{30}$$

where $\boldsymbol{V}^b_E = \begin{bmatrix} u & v & w \end{bmatrix}^T$ represents the velocity vector, $\boldsymbol{\omega}^b_{b/E} = \begin{bmatrix} p & q & r \end{bmatrix}^T$ the angular velocity vector, $\boldsymbol{g}^b$ the gravitational acceleration vector, and $\boldsymbol{F}^b_F$ and $\boldsymbol{F}^b_P$ the forces associated with the vehicle frame and rotors, respectively. The exogenous forces generated by the rotors largely dominate state-dependent aerodynamic forces. The steady-state rotor thrust in hover scales quadratically with rotational velocity according to momentum theory [49, 50]:

$$T_P = \sum_{i=1}^{4} T_i = \sum_{i=1}^{4} C_T \omega_i^2 \tag{31}$$

where C_T is a lumped positive scalar. In this experiment, the simplification is made that thrust levels follow this static relationship in any condition, and are therefore independent of effective rotor speed. Moreover, the assumption of perfect motors is made, which implies that the rotors reach a desired rotational velocity instantaneously. Blade flapping is incorporated for both the longitudinal and lateral directions, and also follows from a static relationship:

$$X_{bf} = -\sum_{i=1}^{4} T_i \sin(C_\beta v_{P_i}) \qquad Y_{bf} = \sum_{i=1}^{4} T_i \sin(-C_\beta u_{P_i}) \tag{32}$$

with C_β serving as the tilt constant, and u_{P_i} and v_{P_i} representing the u- and v-components of the local rotor speed. This yields the following expression for $\boldsymbol{F}^b_P$:

$$\boldsymbol{F}^b_P = \begin{bmatrix} X_{bf} & Y_{bf} & -T_P \end{bmatrix}^T \tag{33}$$

The airframe itself generates lift and drag forces in the aerodynamic frame of reference:

$$\boldsymbol{F}_F^b = \mathbb{T}_{ab}^{-1}(\alpha,\beta)\boldsymbol{F}_F^a = \mathbb{T}_{ab}^{-1}(\alpha,\beta)\bar{q}S\begin{bmatrix}-C_D & 0 & -C_L\end{bmatrix}^T \tag{34}$$

The rotational equations of motion are governed by the Euler equations:

$$\dot{\boldsymbol{\omega}}_{b/E}^b = \boldsymbol{J}^{-1}\left(\boldsymbol{M}_F^b + \boldsymbol{M}_P^b - \boldsymbol{\omega}_{b/E}^b \times \boldsymbol{J}\boldsymbol{\omega}_{b/E}^b\right) \tag{35}$$

where $\boldsymbol{M}_F^b$ and $\boldsymbol{M}_P^b$ are the moments associated with the vehicle frame and rotors, respectively, and $\boldsymbol{J}$ represents the diagonal inertia matrix. The moments generated by the rotors are again dominant, and are given as follows [49]:

$$\boldsymbol{M}_P^b = \begin{bmatrix} 0 & -l & 0 & l \\ l & 0 & -l & 0 \\ -C_\tau & C_\tau & -C_\tau & C_\tau \end{bmatrix}\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} + \begin{bmatrix} L_{bf} \\ M_{bf} \\ 0 \end{bmatrix} \tag{36}$$

where C_τ is the rotor torque constant and l represents the rotor arm length. The terms related to blade flapping take the following form:

$$L_{bf} = -C_{\beta_m}\sum_{i=1}^{4} C_\beta v_{P_i} \qquad\qquad M_{bf} = C_{\beta_m}\sum_{i=1}^{4} C_\beta u_{P_i} \tag{37}$$

where C_{β_m} is a fixed scalar. The moment contribution by the vehicle airframe is very minor, and only appears as a rotational drag term:

$$\boldsymbol{M}_F^b = \begin{bmatrix} 0 & 0 & \frac{1}{2}C_{\tau_D}\rho r^2 \end{bmatrix} \tag{38}$$

Finally, the vehicle attitude and position are governed by the following equations:

$$\dot{\boldsymbol{\Phi}} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi] \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix}\begin{bmatrix} p \\ q \\ r \end{bmatrix} \qquad \dot{\mathbf{x}}_E^E = \mathbb{T}_{bE}^{-1}(\phi,\theta)\boldsymbol{V}_E^b \tag{39}$$

In the model, the attitude dynamics are simulated using quaternions to prevent the occurrence of singularities [49]. It is assumed that perfect sensors are available, with zero bias and noise, and that there are no external disturbances acting on the system. The equations of motion are transformed to discrete time by applying the forward Euler method and a small simulation time step of 0.005 seconds.

B. Learning Framework

A fixed two-stage intra-task learning curriculum is established, in which learning is limited to the longitudinal dynamics only in the first stage, and is extended to the lateral axis in the subsequent stage. The observed outputs and the inputs available to the agent during these phases are as follows:

$$\boldsymbol{y}_t^{(1)} = \begin{bmatrix} u & w & q & \theta \end{bmatrix}^T \qquad \boldsymbol{u}_t^{a(1)} = \begin{bmatrix} M_P \end{bmatrix}^T \tag{40}$$

$$\boldsymbol{y}_t^{(2)} = \begin{bmatrix} u & v & w & p & q & \phi & \theta \end{bmatrix}^T \qquad \boldsymbol{u}_t^{a(2)} = \begin{bmatrix} L_P & M_P \end{bmatrix}^T \tag{41}$$

The greedy task network mapping described in Section III.C is used to bias learning in the lateral axis. The inputs are taken as the moments generated through differential rotor thrust, and are allocated to rotor speed using the relations:

$$\begin{bmatrix} \Delta T_1 \\ \Delta T_2 \\ \Delta T_3 \\ \Delta T_4 \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{2l} \\ -\frac{1}{2l} & 0 \\ 0 & -\frac{1}{2l} \\ \frac{1}{2l} & 0 \end{bmatrix}\begin{bmatrix} L_p \\ M_p \end{bmatrix} \qquad \omega_i = \sqrt{C_T^{-1}(T_{i_0} + \Delta T_i)} \tag{42}$$

Table 2 Quadrotor UAV safe curriculum learning hyperparameters

Parameter	Value	Parameter	Value
Policy evaluation iteration size K	5 [-]	Attitude, speed risk perception range	10 [deg], 1.0 [m/s]
Exploration/backup authority limits	±0.2 [Nm]	Reaching intervals $\epsilon_{u,v}$, $\epsilon_{p,q}$, $\epsilon_{\phi,\theta}$	0.5 [m/s], 10 [deg/s], 5 [deg]
Actor/critic number of hidden nodes	{32,64} [-]	Model uncertainty ΔC_β, ΔC_{β_m}	± 0.25 [-]
Number of training epochs	20 [-]	Backup horizon range	[0.15, 0.90] [sec]
Network weight initialization range	[-0.01, 0.01]	Number of control input iterations	3 [-]
Number of bounding model samples	2 [-]	Number of backup iterations	5 [-]

where T_{i_0} represents the trim thrust in hovering flight. By allocating the agent inputs in this way, the continuous-time moment effectiveness matrix simply equals the inverted inertia matrix.

Although the inactive state dimensions are to a large extent decoupled from the agent-controlled dynamics at any stage in the learning curriculum, their inherently unstable nature requires separate measures to keep the total system state bounded. To this end, a proportional multi-variable feedback control law is used as an external supervisor:

$$\boldsymbol{u}_t^{s(\lambda)} = K_s^{(\lambda)} \begin{bmatrix} p & q & r & \theta & \psi \end{bmatrix}^T \tag{43}$$

Although the supervisor is designed to stabilize the UAV in hover condition, its performance is far from optimal. In the first stage, the supervisor has both the roll and yaw axes under its authority, whereas it only controls yaw in the second stage. Although it never takes control of the pitch channel, it can be used to initialize the actor network such that the initial policy is stable around the hover condition. This implies that a large degree of safety is incorporated already at the onset of learning. For the target task utility function, the following structure is adopted:

$$r(\boldsymbol{y}_t^{(2)}, \boldsymbol{u}_t^{a(2)}) = \begin{bmatrix} u & v & \phi & \theta \end{bmatrix} \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 25 & 0 \\ 0 & 0 & 0 & 25 \end{bmatrix} \begin{bmatrix} u \\ v \\ \phi \\ \theta \end{bmatrix} + \begin{bmatrix} L_P & N_P \end{bmatrix} \begin{bmatrix} I_{xx} & 0 \\ 0 & I_{yy} \end{bmatrix} \begin{bmatrix} L_P \\ N_P \end{bmatrix} \tag{44}$$

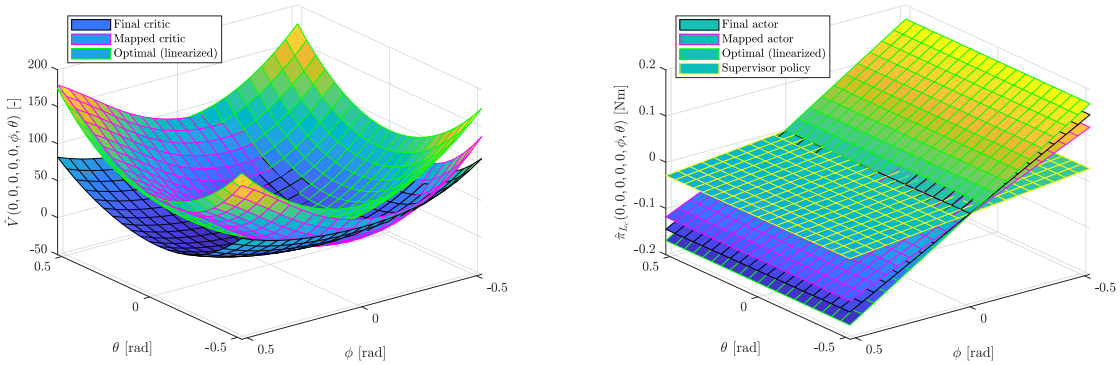
Although optimal control of attitude is the primary goal, it has been observed that the learning process becomes more tractable by incorporating additional penalty terms for velocity. For the first curriculum stage, the utility function is reduced to the longitudinal terms only.

Regarding the safety aspect of the learning task, two risk modes are established that define the safe subset $\mathcal{X}_{safe}^{(\lambda)} \subset \mathcal{X}^{(\lambda)}$. The first mode limits the attitude angle to ± 30 degrees, whereas the second mode specifies that the body speeds u and v should not exceed 3 m/s. Accordingly, the risk perception limits of the SHERPA safety filter have been set to 10 degrees and 1 m/s, respectively. For the internal bounding model, a reduced description of the nonlinear quadrotor dynamics is adopted which only includes the dominant dynamics related to exogenous rotor thrust and blade flapping. For the blade flapping contribution, parametric uncertainties of ±25% are used for the C_β and C_{β_m} terms. As these parameters can be lumped into a single term $C_\beta C_{\beta_m}$ for the rotational dynamics, as illustrated by equations 37, only two model samples corresponding to the lumped infimum and supremum are adopted. Since the internal bounding model $\Delta_{\pi_j}^{\lambda}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ operates on the full state and input space, it also contains an internal representation of the supervisor.

Finally, appropriate values need to be selected for the various hyperparameters associated with learning and the SHERPA safety filter. These are summarized in Table 2. These hyperparameters generally have a significant impact on the learning performance, and obtaining adequate settings can be a tedious task.

C. Results and Discussion

Figure 13 visualizes the cross-sections of the critic and actor outputs as a function of attitude for the nonlinear target task after completion of the learning curriculum. To be able to evaluate learning performance, the optimal control policy and corresponding value function obtained by solving the discrete-time ARE for the quadrotor dynamics

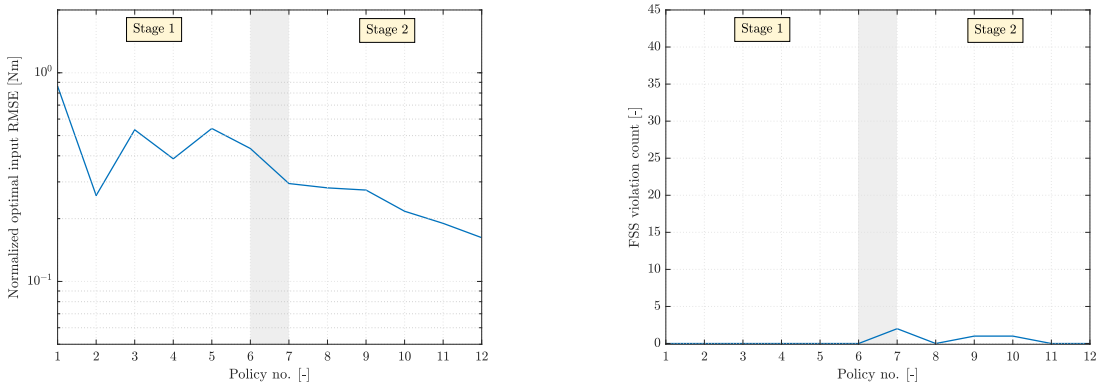


(a) Value function **(b) Lateral policy**

Fig. 13 Example actor and critic outputs as a function of pitch and roll Euler angles at the end of learning, quadrotor safe learning curriculum under SHERPA authority; the optimal outputs are based on the system linearized around the hover condition

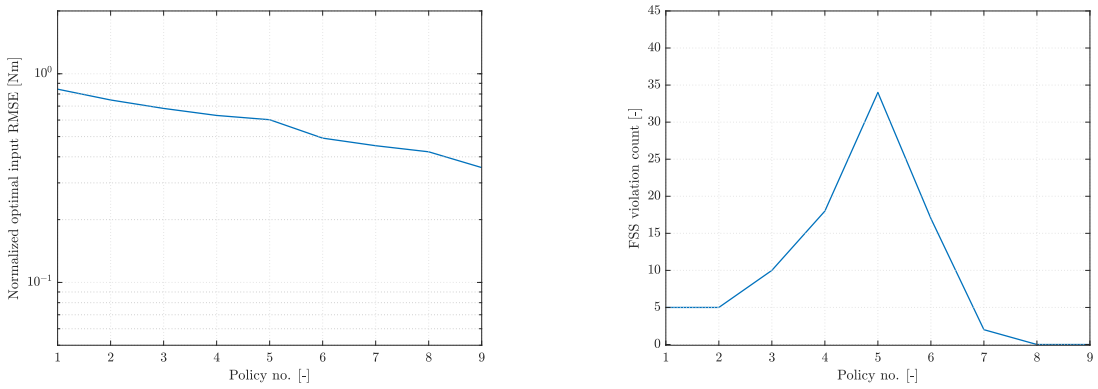
linearized around the hover condition are shown as well. This shows that the optimal solution is approximated reasonably well by both the mapped and final actor-critic networks. The learning statistics corresponding to the safe curriculum demonstration run are shown in Figure 14. To give an indication of the favorable properties of the safe curriculum learning paradigm, Figure 15 shows the same information when learning takes place directly in the target task.

The observation is made that the curriculum approach results in lower normalized RMSE levels with less constraint violations. Despite the fact that any of the intermediate control policies are stable and therefore in principle safe for both experiments, the bounding model uncertainty complicates the identification of safe backup sequences. This is especially problematic for the target task, where uncertainties propagate in both the pitch and roll dimensions. Under the low-gain supervisor control law, these uncertainties result in a relatively large variation in the closed-loop bounding model response, which further reduces the likelihood of finding a backup control sequence that satisfies the closeness condition. By adopting a curriculum approach, the model uncertainty impact is better controlled, as variations in the first learning stage remain confined to the pitch dimension only. Moreover, the control policy obtained from the first stage is of relatively high-gain, which limits the impact of model variations.



(a) Normalized policy RMSE **(b) Constraint violations**

Fig. 14 Nonlinear quadrotor demonstration learning statistics, safe learning curriculum under SHERPA authority; normalized policy RMSE is determined using the optimal policy for the system linearized around the hover condition



(a) Normalized policy RMSE **(b) Constraint violations**

Fig. 15 Nonlinear quadrotor demonstration run learning statistics, non-curriculum approach under SHERPA authority; normalized policy RMSE is determined using the optimal policy for the system linearized around the hover condition

VI. Conclusions and Recommendations

This paper has demonstrated the safe curriculum learning paradigm to autonomously learn optimal control laws for systems with parametric uncertainties in a safe and efficient online manner. Under minimum robustness requirements, curriculum learning can be exploited to simultaneously enhance safety and efficiency during learning by transferring safe agent policies across successive stages in an MDP sequence. A neural-network based actor-critic curriculum learning scheme supported by the SHERPA ergodicity-preserving safety filter has been proposed to investigate the framework in the context of general nonlinear, time-invariant systems with known input dynamics and partially uncertain internal dynamics. Using the concept of a-priori designed task network mappings, the transfer of neural networks weights across successive representations of the actor and the critic in the learning sequence is adopted to enhance learning efficiency and expedite the search for ergodicity-preserving backup control sequences. The latter is achieved by having SHERPA operate on internal closed-loop bounding models under authority of the actor policy.

The safe curriculum framework has been demonstrated in a simple inter-task learning experiment based on a linear, unstable, cascaded mass-spring-damper system. The results show simultaneous improvement of learning efficiency and safety compared to non-curriculum approaches, which is a direct consequence of the fact that the SHERPA safety filter is able to exploit the stabilizing and uncertainty-limiting properties of the actor policy in the most demanding stages of the learning curriculum under the proposed network mapping strategy. Subsequent proof-of-concept of the framework in the context of optimal pitch and roll attitude control of a quadrotor UAV has resulted in similar observations, with the search for backup policies being more effective under the curriculum approach as a result of reduced model uncertainty propagation.

The results from this paper show that informed application of the safe curriculum paradigm has the potential to advance the applicability of online reinforcement learning techniques for safety-critical systems such as UAVs. However, there are many challenges and directions for improvement that need to be addressed in future research. First, the level of a-priori knowledge required should be further reduced to have a true advantage over the traditional flight control design cycle in terms of model-dependency. This could for example be achieved by applying self-paced and model-learning techniques. The former incorporates feedback from the agent learning progress, and may therefore result in more effective learning sequences. Online model-learning could be investigated to achieve reliable bounding model contractions, which may allow for higher initial model uncertainties and eventually reduced uncertainty propagation as the model is adapted using online system information. A curriculum approach could be considered here as well.

Second, safety and learning could be further integrated by introducing feedback from the SHERPA safety filter to the learning agent. Third, the curriculum framework could be further explored by quantifying the effects of non-observable dynamics and developing minimum conditions for positive knowledge transfer for improving learning efficiency. Finally, further research is needed to reduce the computational complexity of the algorithm and minimize the effects of hyperparameters. Since considerable effort must be spent to obtain adequate settings for neural network training and the safety filter, informed guidelines should be developed before the framework can be used in real applications.

References

[1] Stevens, B., Lewis, F., and Johnson, E., *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*, Wiley, 2015.

[2] Khargonekar, P. P., Petersen, I. R., and Zhou, K., "Robust stabilization of uncertain linear systems: quadratic stabilizability and H^{∞} control theory," *IEEE Transactions on Automatic Control*, Vol. 35, No. 3, 1990, pp. 356–361. doi:10.1109/9.50357.

[3] Veillette, R. J., Medanic, J. B., and Perkins, W. R., "Design of reliable control systems," *IEEE Transactions on Automatic Control*, Vol. 37, No. 3, 1992, pp. 290–304. doi:10.1109/9.119629.

[4] Guang-Hong Yang, Lam, J., and Jianliang Wang, "Reliable H_{∞} control for affine nonlinear systems," *IEEE Transactions on Automatic Control*, Vol. 43, No. 8, 1998, pp. 1112–1117. doi:10.1109/9.704984.

[5] Sieberling, S., Chu, Q. P., and Mulder, J. A., "Robust Flight Control Using Incremental Nonlinear Dynamic Inversion and Angular Acceleration Prediction," *Journal of Guidance, Control, and Dynamics*, Vol. 33, No. 6, 2010, pp. 1732–1742. doi:10.2514/1.49978.

[6] Slotine, J., and Li, W., *Applied nonlinear control*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[7] Sonneveldt, L., Chu, Q., and Mulder, J., "Nonlinear Flight Control Design Using Constrained Adaptive Backstepping," *Journal of Guidance, Control, and Dynamics*, Vol. 30, No. 2, 2007, pp. 322–336. doi:10.2514/1.25834.

[8] Acquatella, P., van Kampen, E., and Chu, Q., "Incremental backstepping for robust nonlinear flight control," *EuroGNC 2013, 2nd CEAS Specialist Conference on Guidance, Navigation & Control*, 2013.

[9] Kaelbling, L., Littman, M., and Moore, A., "Reinforcement Learning: A Survey," *Journal of artificial intelligence research*, Vol. 4, 1996, pp. 237–285.

[10] Sutton, R., and Barto, A., *Reinforcement learning: An introduction*, MIT press, Cambridge, MA, 1998.

[11] Szepesvári, C., "Algorithms for reinforcement learning," *Synthesis lectures on artificial intelligence and machine learning*, Morgan & Claypool Publishers, 2010.

[12] Busoniu, L., Babuska, R., Schutter, D., and Ernst, D., *Reinforcement Learning and Dynamic Programming Using Function Approximators*, Automation and Control Engineering, CRC Press, Inc., 2010.

[13] Sutton, R. S., Barto, A. G., and Williams, R. J., "Reinforcement learning is direct adaptive optimal control," *IEEE Control Systems Magazine*, Vol. 12, No. 2, 1992, pp. 19–22. doi:10.1109/37.126844.

[14] Bengio, Y., Louradour, J., Collobert, R., and Weston, J., "Curriculum Learning," *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, pp. 41–48. doi:10.1145/1553374.1553380.

[15] Taylor, M. E., "Assisting Transfer-Enabled Machine Learning Algorithms: Leveraging Human Knowledge for Curriculum Design," *Agents that Learn from Human Teachers, Papers from the 2009 AAAI Spring Symposium, Technical Report SS-09-01, Stanford, California, USA, March 23-25, 2009*, 2009, pp. 141–143.

[16] García, J., and Fernández, F., "A Comprehensive Survey on Safe Reinforcement Learning," *Journal of Machine Learning Research*, Vol. 16, No. 42, 2015, pp. 1437–1480.

[17] Mannucci, T., Van Kampen, E., de Visser, C. C., and Chu, Q. P., "SHERPA: a safe exploration algorithm for Reinforcement Learning controllers," *AIAA Guidance, Navigation, and Control Conference, AIAA SciTech Forum*, 2015. doi:10.2514/6.2015-1757.

[18] Mannucci, T., van Kampen, E., de Visser, C., and Chu, Q., "Safe Exploration Algorithms for Reinforcement Learning Controllers," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 29, No. 4, 2018, pp. 1069–1081. doi: 10.1109/TNNLS.2017.2654539.

[19] Perkins, T., and Barto, A., "Lyapunov design for safe reinforcement learning," *Journal of Machine Learning Research*, Vol. 3, 2003, pp. 803–832. doi:10.1162/jmlr.2003.3.4-5.803.

[20] Held, D., McCarthy, Z., Zhang, M., Shentu, F., and Abbeel, P., "Probabilistically safe policy transfer," *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5798–5805. doi:10.1109/ICRA.2017.7989680.

[21] Helmer, A., de Visser, C. C., and van Kampen, E., "Flexible Heuristic Dynamic Programming for Reinforcement Learning in Quad-Rotors," *2018 AIAA Information Systems-AIAA Infotech @ Aerospace*, 2018. doi:10.2514/6.2018-2134.

[22] Taylor, M., and Stone, P., "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, Vol. 10, No. 1, 2009, pp. 1633–1685.

[23] Liu, D., Wei, Q., Wang, D., Yang, X., and Li, H., *Adaptive dynamic programming with applications in optimal control*, Springer, 2017.

[24] Wang, F., Zhang, H., and Liu, D., "Adaptive Dynamic Programming: An Introduction," *IEEE Computational Intelligence Magazine*, Vol. 4, No. 2, 2009, pp. 39–47. doi:10.1109/MCI.2009.932261.

[25] Werbos, P., "Approximate dynamic programming for realtime control and neural modelling," *Handbook of intelligent control: neural, fuzzy and adaptive approaches*, 1992, pp. 493–525.

[26] Prokhorov, D. V., and Wunsch, D. C., "Adaptive critic designs," *IEEE Transactions on Neural Networks*, Vol. 8, No. 5, 1997, pp. 997–1007. doi:10.1109/72.623201.

[27] Skinner, B. F., "Reinforcement today." *American Psychologist*, Vol. 13, No. 3, 1958, pp. 94–99. doi:10.1037/h0049039.

[28] Kumar, M. P., Packer, B., and Koller, D., "Self-Paced Learning for Latent Variable Models," *Advances in Neural Information Processing Systems 23*, 2010, pp. 1189–1197.

[29] Jiang, L., Meng, D., Yu, S.-I., Lan, Z., Shan, S., and Hauptmann, A., "Self-Paced Learning with Diversity," *Advances in Neural Information Processing Systems 27*, 2014, pp. 2078–2086.

[30] Narvekar, S., Sinapov, J., and Stone, P., "Autonomous Task Sequencing for Customized Curriculum Design in Reinforcement Learning," *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 2017, pp. 2536–2542. doi:10.24963/ijcai.2017/353.

[31] Taylor, M. E., Stone, P., and Liu, Y., "Transfer Learning via Inter-Task Mappings for Temporal Difference Learning," *Journal of Machine Learning Research*, Vol. 8, No. 1, 2007, pp. 2125–2167.

[32] Rosenstein, M. T., and Barto, A. G., "Reinforcement learning with supervision by a stable controller," *Proceedings of the 2004 American Control Conference*, Vol. 5, 2004, pp. 4517–4522 vol.5. doi:10.23919/ACC.2004.1384022.

[33] Hans, A., Schneegaß, D., Schäfer, A. M., and Udluft, S., "Safe Exploration for Reinforcement Learning," *Proceedings of the 16th European Symposium on Artificial Neural Networks*, 2008, pp. 143–148.

[34] Moldovan, T. M., and Abbeel, P., "Safe Exploration in Markov Decision Processes," *Proceedings of the 29th International Coference on International Conference on Machine Learning*, 2012, pp. 1451–1458.

[35] Akametalu, A. K., Fisac, J. F., Gillula, J. H., Kaynama, S., Zeilinger, M. N., and Tomlin, C. J., "Reachability-based safe learning with Gaussian processes," *53rd IEEE Conference on Decision and Control*, 2014, pp. 1424–1431. doi: 10.1109/CDC.2014.7039601.

[36] Berkenkamp, F., Turchetta, M., Schoellig, A., and Krause, A., "Safe model-based reinforcement learning with stability guarantees," *Proceedings of Neural Information Processing Systems*, 2017, pp. 908–918.

[37] Freeman, R., and Kokotovic, P. V., *Robust nonlinear control design: state-space and Lyapunov techniques*, Springer Science & Business Media, 2008.

[38] Wieland, P., and Allgöwer, F., "Constructive Safety using Control Barrier Functions," *IFAC Proceedings Volumes*, Vol. 40, No. 12, 2007, pp. 462 – 467. doi:https://doi.org/10.3182/20070822-3-ZA-2920.00076, 7th IFAC Symposium on Nonlinear Control Systems.

[39] Tee, K. P., Ge, S. S., and Tay, E. H., "Barrier Lyapunov Functions for the control of output-constrained nonlinear systems," *Automatica*, Vol. 45, No. 4, 2009, pp. 918 – 927. doi:https://doi.org/10.1016/j.automatica.2008.11.017.

[40] Lazaric, A., *Transfer in Reinforcement Learning: A Framework and a Survey*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 143–173. doi:10.1007/978-3-642-27645-3_5.

[41] Kiumarsi, B., Vamvoudakis, K. G., Modares, H., and Lewis, F. L., "Optimal and Autonomous Control Using Reinforcement Learning: A Survey," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 29, No. 6, 2018, pp. 2042–2062. doi:10.1109/TNNLS.2017.2773458.

[42] Lewis, F. L., and Vrabie, D., "Reinforcement learning and adaptive dynamic programming for feedback control," *IEEE Circuits and Systems Magazine*, Vol. 9, No. 3, 2009, pp. 32–50. doi:10.1109/MCAS.2009.933854.

[43] Al-Tamimi, A., Lewis, F. L., and Abu-Khalaf, M., "Discrete-Time Nonlinear HJB Solution Using Approximate Dynamic Programming: Convergence Proof," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Vol. 38, No. 4, 2008, pp. 943–949. doi:10.1109/TSMCB.2008.926614.

[44] Kiumarsi-Khomartash, B., Lewis, F. L., Naghibi-Sistani, M., and Karimpour, A., "Optimal tracking control for linear discrete-time systems using reinforcement learning," *52nd IEEE Conference on Decision and Control*, 2013, pp. 3845–3850. doi:10.1109/CDC.2013.6760476.

[45] Kiumarsi, B., and Lewis, F. L., "Actor–Critic-Based Optimal Tracking for Partially Unknown Nonlinear Discrete-Time Systems," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 26, No. 1, 2015, pp. 140–151. doi:10.1109/TNNLS.2014.2358227.

[46] van Kampen, E., Chu, Q. P., and Mulder, J. A., "Continuous Adaptive Critic Flight Control Aided with Approximated Plant Dynamics," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2006. doi:10.2514/6.2006-6429.

[47] Grondman, I., Vaandrager, M., Busoniu, L., Babuska, R., and Schuitema, E., "Efficient Model Learning Methods for Actor–Critic Control," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Vol. 42, No. 3, 2012, pp. 591–602. doi:10.1109/TSMCB.2011.2170565.

[48] Kiumarsi, B., Lewis, F. L., Naghibi-Sistani, M., and Karimpour, A., "Optimal Tracking Control of Unknown Discrete-Time Linear Systems Using Input-Output Measured Data," *IEEE Transactions on Cybernetics*, Vol. 45, No. 12, 2015, pp. 2770–2779. doi:10.1109/TCYB.2014.2384016.

[49] Molenkamp, D., van Kampen, E., de Visser, C. C., and Chu, Q. P., "Intelligent Controller Selection for Aggressive Quadrotor Manoeuvring," *AIAA Information Systems-AIAA Infotech @ Aerospace*, 2017. doi:10.2514/6.2017-1068.

[50] Mahony, R., Kumar, V., and Corke, P., "Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor," *IEEE Robotics Automation Magazine*, Vol. 19, No. 3, 2012, pp. 20–32. doi:10.1109/MRA.2012.2206474.

II

Literature Review

This part is structured as follows. Chapter 2 serves as a basic introduction to the fundamentals of reinforcement learning, including Markov Decision Processes, Dynamic Programming, and sample-based learning. This introductory presentation is supported by a range of theories and algorithms that are elemental to general reinforcement learning applications. Chapter 3 presents a more focused view of reinforcement learning in the context of adaptive optimal control, by giving a very basic introduction to general optimal control, linear quadratic regulation and tracking, and adaptive critic designs. This is followed by an introduction to curriculum learning in Chapter 4, which presents a coherent taxonomy for this field. Moreover, some examples from literature in the context of supervised learning and reinforcement learning are discussed, together with a brief overview of transfer learning techniques. Subsequently, Chapter 5 presents the area of Safe Learning by discussing a range of different approaches and views as reported in literature.

This part is based on previously graded work.

2

Reinforcement Learning Fundamentals

REINFORCEMENT Learning (RL) in its canonical form originates from the artificial intelligence community, where it is formulated as a data-driven learning technique for attaining sequences of control actions or decisions that optimize a given long-term objective [38, 98, 99]. In this context, RL can be applied to address sequential decision making problems formulated as Markov Decision Processes (MDPs) in the absence of an explicit model description of the system and its surroundings. Formulated this way, RL methods can directly operate on data obtained from the system to learn the (near-)optimal decision sequence for the problem at hand [18]. This can be done either off-line based on a-priori collected data from the process that needs to be interacted with, or completely online through direct interaction and autonomous collection of data.

The RL paradigm closely resembles other techniques that optimize a control or decision sequence across time [80, 110]. Dynamic Programming (DP) [8, 12] stems from the field of operations research, where it used for finding temporary solutions that minimize a certain cost or objective function. DP methods require complete knowledge about the dynamics of the system and its surroundings, making it distinct from the canonical RL framework. However, this knowledge does not need to be captured by a strict analytic formulation, but can be directly exploited based on simulations [18]. From the perspective of control theory, it is strongly related to the problem of optimal control, which is concerned with optimizing a certain measure of a controlled process or system over time that often transcends stability constraints [98, 110]. DP methods are commonly applied off-line by iterating and converging towards the (near-)optimal solution, which can be subsequently implemented on-line if desired. Alternatively, model-predictive control (MPC) can be seen as a form of on-line DP in the case of receding-horizon procedures (RHPs) [18]. Note that DP is strictly speaking not a learning approach *per se*, which is another clear distinction from RL.

The abundance of methods to optimize temporary control actions or decisions given a certain process illustrates the widespread need for such techniques. Applications can be found in artificial intelligence, automatic control, economics, and operations research [18]. This has lead to independent developments and discoveries in each of these fields. Multiple attempts have been made to unite these different branches of research [13, 89, 95]. Hybrid methods have also emerged, such as model-learning RL. In this framework, RL methods are used to identify a model of the system and its surroundings, which is subsequently used to exercise planning future actions [98]. This shows that RL is a very wide framework that cannot be seen in isolation from any of the other paradigms mentioned here.

This chapter focuses on the fundamentals of RL/DP. It starts with a formulation of the sequential decision making problem in Section 2.1, which introduces the framework and terminology adopted throughout the rest of this study. The corresponding MDP formulation is introduced in Section 2.2. Subsequently, canonical DP methods are discussed in Section 2.3 to provide the necessary background before proceeding to RL. Methods for value prediction, which forms a central element in RL algorithms, are discussed in Section 2.4. Finally, the concept of learning optimal behavior or control is treated in Section 2.5, together with an in-depth discussion of canonical RL methods. The majority of the material presented in this chapter, including derivations, equations, theories, and algorithms, has been based on [18, 98], as well as the lecture series[1] on the basics of reinforcement learning by David Silver at University College London (UCL). Therefore, specific referencing to these sources is often omitted.

[1] *Advanced Topics in Machine Learning* (*COMPM050*/*COMPGI13*) at University College London, London, United Kingdom, by D. Silver (2015); retrieved from `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html`.

2.1. The Sequential Decision-Making Problem

A sequential decision making problem is often regarded as a discrete-time formulation of the optimal control problem. In this framework, a decision-making entity repeatedly interacts with a certain *process* or *environment*, where it receives a scalar *reward* or *utility* after each transition based on a certain *objective*. In the field of artificial intelligence, the decision-making entity is referred to as the *agent*, whereas in control theory actions are generated by a *controller* consisting of a set of *control laws*. Although the present study is performed from a control theoretical perspective, the former terminology will be adopted for consistency with RL/DP literature. Figure 2.1 illustrates the sequential decision making problem.

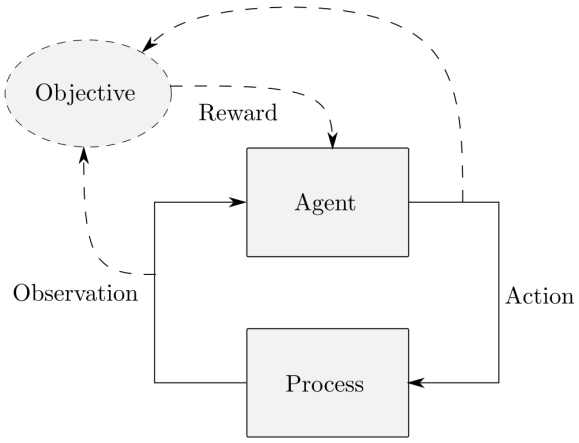


Figure 2.1: The sequential decision-making problem. Adapted from [98].

In general, the dynamics exhibited by the controlled process can be (highly) nonlinear, time-varying, and stochastic. In order to optimize the long-term objective, it commonly occurs that the agent needs to make decisions that are sub-optimal in the short term. An agent that not does take future considerations into account is generally known as *myopic* [98]. In this work, far-sighted agents will be considered.

A central topic within RL that makes it distinct from the classical optimal control framework is the need to *explore*. Starting with zero or limited knowledge about the controlled process or environment, exploration allows the agent to identify the optimal control strategy. The concept of exploration is in conflict with the need to *exploit* the knowledge already acquired by the agent, in order to improve its current behavior. The exploration/exploitation dilemma is unique to RL, and balancing these aspects appropriately generally poses a very difficult problem to solve.

The discrete-time formulation of the sequential decision process is consistent with digital computation frameworks [110]. However, continuous-time equivalents also exist, as discussed in [14, 20, 56]. The key principles of the sequential decision making process formulated here can be largely carried over to the continuous-time case. However, in this work, the discrete-time formulation will be adopted.

2.2. Markov Decision Processes

A Markov Decision Process (MDP) formalizes the sequential decision-making framework. In a stochastic setting, the process or environment is described by a random variable denoted as $X_t \in \mathscr{X} \subset \mathbb{R}^n$ with realizations $\boldsymbol{x}$, known as the *state*. The evolution of the state is conditioned on the control action $\boldsymbol{u}$, which is generated by the random variable $U_t \in \mathscr{U} \subset \mathbb{R}^m$. For continuous, multi-dimensional state spaces, the process dynamics are described by a transition joint probability function $\tilde{f} : \mathscr{X} \times \mathscr{U} \times \mathscr{X} \mapsto \mathbb{R}^{n+}$, resulting in a state distribution function $\tilde{F} : \mathscr{X} \times \mathscr{U} \times \mathscr{X} \mapsto [0,1]$ [18]:

$$\tilde{F}(\boldsymbol{x}, \boldsymbol{u}, \underline{\boldsymbol{x}'}) \doteq \mathbb{P}\left(X_{t+1} \in \underline{\boldsymbol{x}} | X_t = \boldsymbol{x}, U_t = \boldsymbol{u}\right) = \int_{\underline{x}'_n} \cdots \int_{\underline{x}'_1} \tilde{f}(\boldsymbol{x}, \boldsymbol{u}, x'_1 \ldots x'_n) \mathrm{d}x'_1 \ldots \mathrm{d}x'_n \tag{2.1}$$

indicating the probability that the next state $\boldsymbol{x}'$ is within a region of the state space $\underline{\boldsymbol{x}'} = \left[\underline{x}'_1, \ldots, \underline{x}'_n\right]^T \subseteq \mathscr{X}$. If the state space is countable, the above probability can be written as $\tilde{F}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}') = \mathbb{P}\left(X_{t+1} = \boldsymbol{x}' | X_t = \boldsymbol{x}, U_t = \boldsymbol{u}\right)$, which is more commonly seen in literature on MDPs. Note that it is very difficult in general to derive an analytical formulation for $\tilde{f}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')$ [18]. The above expression demonstrates the *Markov* property, which states

that the current state vector is a sufficient description for the entire state history, i.e. $\mathbb{P}[X_{t+1} = \boldsymbol{x}'|X_t, U_t] = \mathbb{P}[X_{t+1} = \boldsymbol{x}'|X_0, U_0, \ldots, X_t, U_t]$ [97]. This implies that knowledge about the realizations $\boldsymbol{x}$ and $\boldsymbol{u}$ is sufficient to predict the future state $\boldsymbol{x}'$. Note that Equation 2.1 forms the stochastic equivalent of the deterministic discrete-time transition function, which is formulated as $\boldsymbol{x}' = f(\boldsymbol{x}, \boldsymbol{u})$.

The agent perceives the state $\boldsymbol{x}$ of the process through an observation $\boldsymbol{y}$ generated by the random variable $Y_t \in \mathscr{Y} \subset \mathbb{R}^l$, where l can generally be dissimilar from n. In an MDP however, it is assumed that $\mathscr{Y} = \mathscr{X}$ and that $\boldsymbol{y} = \boldsymbol{x}\ \forall t$. From a control theoretical perspective, this implies that the process is fully observable, and every observation is not corrupted by noise. An MDP that does not limit itself to this assumption is known as a *partially observable* MDP (POMDP). The reward obtained by the agent can be formulated as the expectation of the random variable R_t, which is described by the reward function $\tilde{\rho} : \mathscr{X} \times \mathscr{U} \times \mathscr{X} \mapsto \mathbb{R}$:

$$\tilde{\rho}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}') = \mathbb{E}\left[R_{t+1}|X_t = \boldsymbol{x}, U_t = \boldsymbol{u}, X_{t+1} = \boldsymbol{x}'\right] \tag{2.2}$$

However, it is common to assign deterministic rewards for every transition. This implies that the expectation operator can be discarded, which can be reflected in the reward function by dropping the tilde notation.

The goal of the agent is to maximize the reward sequence from every state $\boldsymbol{x}$, denoted as the *return*. In discrete-time, the return can be formulated as the algebraic sum of total collected reward, discounted at every time step by a *discount rate* $0 \leq \gamma \leq 1$. This leads to the following statistic:

$$G_t \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.3}$$

This definition of the return refers to the total discounted reward over the complete lifetime of the agent. The function of the discount rate γ is twofold: (1) it provides a way to specify the value of future rewards, making the agent more or less sensitive to future events; and (2) it keeps the return bounded in case the lifetime of the agent approaches infinity. In case $\gamma = 0$, the agent adopts myopic behavior. On the other hand, if termination of the agent-process of the agent is guaranteed, it is possible to use undiscounted returns by setting $\gamma = 1$. This only applies to processes that are *episodic* in nature. Note that other discounting functions can be used, such as exponential discounting (e.g., as done in [20]), or the infinite-horizon average reward. In the MDP framework however, it is common to use the multiplicative factor γ instead.

The goal of maximizing the discounted return G_t can be attained by the agent by selecting an appropriate sequence of control actions that lead to an optimal trajectory. Such a decision sequence is captured by a *policy* (control law) $\tilde{\pi} : \mathscr{X} \times \mathscr{U} \mapsto [0, 1]$, which can in general be stochastic:

$$\tilde{\pi}(\underline{\boldsymbol{u}}|\boldsymbol{x}) = \mathbb{P}\left(U_t \in \underline{\boldsymbol{u}}|X_t = \boldsymbol{x}\right) \tag{2.4a}$$

where $\underline{\boldsymbol{u}} = \left[\underline{u}_1, \ldots, \underline{u}_m\right]^T \subseteq \mathscr{U}$ reprents a finite range of control inputs in this case. Note that if the action space $\mathscr{U}$ is discrete, the above expression simplifies to the more common definition $\tilde{\pi}(\boldsymbol{u}|\boldsymbol{x}) = \mathbb{P}(U_t = \boldsymbol{u}|X_t = \boldsymbol{x})$. Depending on the application, deterministic policies are also frequently adopted. Dropping again the tilde notation to indicate non-stochasticity,

$$\boldsymbol{u} = \pi(\boldsymbol{x}) \tag{2.4b}$$

hence, $\pi : \mathscr{X} \mapsto \mathscr{U}$ in the deterministic case. The concepts of return G_t and policy $\tilde{\pi}$ (adopting the stochastic notation to ensure generality) can be combined by estimating the *value* of being in a certain state $\boldsymbol{x}$ and following policy $\tilde{\pi}$ afterwards. This leads to the concept of *value functions*, formulated as:

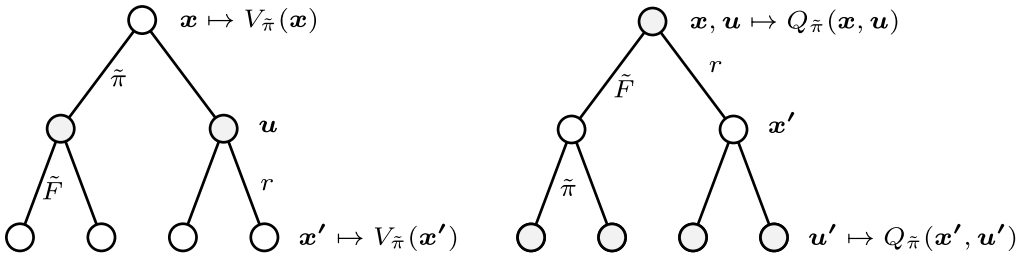
$$V_{\tilde{\pi}}(\boldsymbol{x}) \doteq \mathbb{E}_{\tilde{\pi}}\left[G_t|X_t = \boldsymbol{x}\right] = \mathbb{E}_{\tilde{\pi}}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|X_t = \boldsymbol{x}\right] \tag{2.5a}$$

This is the definition of the *state value function*, which specifies the value as the expected return when starting from state $\boldsymbol{x}$ and following the policy $\tilde{\pi}$ thereafter. This equation can be rewritten in recursive form:

$$V_{\tilde{\pi}}(\boldsymbol{x}) = \mathbb{E}_{\tilde{\pi}}\left[R_{t+1} + \gamma G_{t+1}|X_t = \boldsymbol{x}\right] \tag{2.5b}$$

For discrete state and action spaces, the *Bellman equation* can be obtained by expanding the expectation:

$$\begin{aligned} V_{\tilde{\pi}}(\boldsymbol{x}) &= \sum_{\mathscr{U}} \tilde{\pi}(\boldsymbol{u}|\boldsymbol{x}) \sum_{\mathscr{X}} \tilde{F}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}') \left(\tilde{\rho}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}') + \gamma \mathbb{E}_{\tilde{\pi}}\left[G_{t+1}|X_{t+1} = \boldsymbol{x}'\right]\right) \\ &= \sum_{\mathscr{U}} \tilde{\pi}(\boldsymbol{u}|\boldsymbol{x}) \sum_{\mathscr{X}} \tilde{F}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}') \left(\tilde{\rho}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}') + \gamma V_{\tilde{\pi}}(\boldsymbol{x}')\right) \end{aligned} \tag{2.5c}$$



(a) State-value function $V_{\tilde{\pi}}(\boldsymbol{x})$ **(b)** Action-value function $Q_{\tilde{\pi}}(\boldsymbol{x}, \boldsymbol{u})$

Figure 2.2: Example back-up diagrams. Adapted from [98].

Writing this expression in expectation form again, the *Bellman expectation equation* is obtained:

$$V_{\tilde{\pi}}(\boldsymbol{x}) = \mathbb{E}_{\tilde{\pi}}\left[R_{t+1} + \gamma V_{\tilde{\pi}}(X_{t+1}) | X_t = \boldsymbol{x}\right] \tag{2.5d}$$

This result equally applies to MDPs with continuous state and action spaces as well. The value function concept can also be applied to state-action pairs, leading to the definition of the *state-action value function* $Q_{\tilde{\pi}}(\boldsymbol{x}, \boldsymbol{u})$ (also known as the Q-function). The state-action value function is defined as the expected return when starting from state $\boldsymbol{x}$, taking action $\boldsymbol{u}$, and following policy $\tilde{\pi}$ afterwards. This results in an equivalent formulation as for the state value function:

$$Q_{\tilde{\pi}}(\boldsymbol{x}, \boldsymbol{u}) \doteq \mathbb{E}_{\tilde{\pi}}\left[G_t | X_t = \boldsymbol{x}, U_t = \boldsymbol{u}\right] = \mathbb{E}_{\tilde{\pi}}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | X_t = \boldsymbol{x}, U_t = \boldsymbol{u}\right] \tag{2.6a}$$

A similar expansion for discrete state and action spaces can be performed as done in equations 2.5b and 2.5c, resulting in the Bellman equation for action values:

$$Q_{\tilde{\pi}}(\boldsymbol{x}, \boldsymbol{u}) = \sum_{\mathscr{X}} \tilde{F}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')\left(\tilde{\rho}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}') + \gamma \sum_{\mathscr{U}} \tilde{\pi}(\boldsymbol{u}|\boldsymbol{x}) Q_{\tilde{\pi}}(\boldsymbol{x}', \boldsymbol{u}')\right) \tag{2.6b}$$

Subsequently, the equivalent Bellman expectation equation can be obtained:

$$Q_{\tilde{\pi}}(\boldsymbol{x}, \boldsymbol{u}) = \mathbb{E}_{\tilde{\pi}}\left[R_{t+1} + \gamma Q_{\tilde{\pi}}(X_{t+1}, U_{t+1}) | X_t = \boldsymbol{x}, U_t = \boldsymbol{u}\right] \tag{2.6c}$$

Based on these definitions, the state-value and state-action value functions are related as:

$$V_{\tilde{\pi}}(\boldsymbol{x}) = \sum_{\mathscr{U}} \tilde{\pi}(\boldsymbol{u}|\boldsymbol{x}) Q_{\tilde{\pi}}(\boldsymbol{x}, \boldsymbol{u}) \tag{2.7}$$

As suggested by [98], equations 2.5c and 2.6b can be portrayed by means of *back-up* diagrams in the case of discrete state and action spaces. Back-up diagrams for $V_{\tilde{\pi}}(\boldsymbol{x})$ and $Q_{\tilde{\pi}}(\boldsymbol{x}, \boldsymbol{u})$ are visualized in Figure 2.2. These diagrams illustrate a one-step decision process, where the stochastic nature of the process is reflected by the range of possible transitions for a given state and commanded control action. The latter aspect underscores the limited authority of the agent in the general case.

The value function is inherently connected to a given policy $\tilde{\pi}$. Therefore, it can be maximized (or minimized) by adopting the *optimal policy*. This formalizes the sequential decision-making problem. The optimal state value and state-action value functions are defined as:

$$V_*(\boldsymbol{x}) \doteq \max_{\Pi} V_{\tilde{\pi}}(\boldsymbol{x}) \tag{2.8a}$$

$$Q_*(\boldsymbol{x}, \boldsymbol{u}) \doteq \max_{\Pi} Q_{\tilde{\pi}}(\boldsymbol{x}, \boldsymbol{u}) \tag{2.8b}$$

with Π the total space of attainable policies. Using these definitions and referring to equations 2.6b and 2.7, the optimal state-value and state-action value functions are related as:

$$V_*(\boldsymbol{x}) = \max_{\mathscr{U}} Q_*(\boldsymbol{x}, \boldsymbol{u}) \tag{2.8c}$$

$$Q_*(\boldsymbol{x}, \boldsymbol{u}) = \sum_{\mathscr{X}} \tilde{F}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')\left(\tilde{\rho}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}') + \gamma V_*(\boldsymbol{x}')\right) \tag{2.8d}$$

From these expressions, the *Bellman optimality equations* can be obtained:

$$V_*(\boldsymbol{x}) = \max_{\mathscr{U}} \sum_{\mathscr{X}} \tilde{F}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}') \left(\tilde{\rho}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}') + \gamma V_*(\boldsymbol{x}') \right) \tag{2.8e}$$

$$Q_*(\boldsymbol{x}, \boldsymbol{u}) = \sum_{\mathscr{X}} \tilde{F}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}') \left(\tilde{\rho}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}') + \gamma \max_{\mathscr{U}} Q_*(\boldsymbol{x}', \boldsymbol{u}') \right) \tag{2.8f}$$

Which can again be written in expectation form:

$$V_*(\boldsymbol{x}) = \max_{\mathscr{U}} \mathbb{E}\left[R_{t+1} + \gamma V_*(X_{t+1}) | X_t = \boldsymbol{x} \right] \tag{2.8g}$$

$$Q_*(\boldsymbol{x}, \boldsymbol{u}) = \mathbb{E}\left[R_{t+1} + \gamma \max_{\mathscr{U}} Q_*(X_{t+1}, \boldsymbol{u}') | X_t = \boldsymbol{x}, U_t = \boldsymbol{u} \right] \tag{2.8h}$$

As before, these results also generalize to the case of continuous state and action spaces. Note that the optimal state value an state-action value functions of an MDP are independent of any policy, which is reflected in the expectation operator. The optimal policy $\tilde{\pi}_*$ is then given by:

$$\tilde{\pi}_*(\boldsymbol{u}|\boldsymbol{x}) = 1 \iff \boldsymbol{u} = \arg\max_{\mathscr{U}} Q_*(\boldsymbol{x}, \boldsymbol{u}) \tag{2.9a}$$

This is known as a *greedy* policy. Note that in the continuous case, $\boldsymbol{u}$ represents a near-optimal control action. This result shows that the optimal control action can be obtained directly by optimizing over the optimal state-action value function given a state $\boldsymbol{x}$. Conversly, using the state-value function for deriving the optimal policy requires a one-step lookahead over all possible state transitions (see also Figure 2.2a):

$$\tilde{\pi}_*(\boldsymbol{u}|\boldsymbol{x}) = 1 \iff \boldsymbol{u} = \arg\max_{\mathscr{U}} \mathbb{E}\left[R_{t+1} + \gamma V_*(X_{t+1}) | X_t = \boldsymbol{x} \right] \tag{2.9b}$$

This completes the description of MDPs. It is common to refer to an MDP as a tuple $M : \langle \mathscr{X}, \mathscr{U}, \tilde{F}, \tilde{\rho}, \gamma \rangle$ describing the agent, the process, the objective, and the discount rate. This notation will also be adopted throughout the rest of this study.

2.3. Dynamic Programming

In case complete knowledge about a given MDP is available, the problem of finding the optimal policy $\tilde{\pi}_*$ can be addressed by adopting a class of methods collectively referred to as *Dynamic Programming* (DP) methods. DP algorithms explicitly exploit the knowledge base captured by the $\langle \mathscr{X}, \mathscr{U}, \tilde{F}, \tilde{\rho}, \gamma \rangle$ tuple and use this to solve the Bellman optimality equation 2.8g (or 2.8h for state-action values). DP methods are applied in an off-line setting: upon convergence, the optimal control law is stored and passed on to the online agent, which will be restricted from further learning. In this view, finding the optimal control policy reduces to an off-line optimization problem. However, this problem is generally nonlinear due to the presence of the maximum operator in the Bellman optimality equation. This implies that a closed form solution is commonly not available, and that iterative approaches are required. Two methods that have received widespread attention are *Value Iteration* (VI) and *Policy Iteration* (PI). These approaches are discussed in Subsections 2.3.1 and 2.3.2, respectively.

2.3.1. Value Iteration

With Value Iteration (VI), the Bellman optimality equation is rewritten in iterative form (i.e. an update rule) and applied successively upon convergence to the optimal value function. VI can be performed based on either state-values or state- action values, where the distinction is made clear by referring to these methods as *V-iteration* or *Q-iteration*, respectively [18]. For the former, the following update rule is defined:

$$V^{(j+1)}(\boldsymbol{x}) \doteq \max_{\mathscr{U}} \mathbb{E}\left[R_{t+1} + \gamma V^{(j)}(X_{t+1}) | X_t = \boldsymbol{x} \right] \tag{2.10a}$$

Algorithm 2.1: V-iteration for general discrete-time stochastic MDPs [18, 98]

input : State space $\mathscr{X}$, action space $\mathscr{U}$, transition dynamics $\tilde{F}$, reward function $\tilde{\rho}$, discount rate γ, threshold η
output: Near-optimal state value function $V_*(\boldsymbol{x}) \leftarrow V^{(j+1)}(\boldsymbol{x})$

initialize $V_0(\boldsymbol{x})$;
repeat
  for $\forall \boldsymbol{x} \in \mathscr{X}$ **do**
    $V^{(j+1)}(\boldsymbol{x}) \leftarrow \max_{\mathscr{U}} \mathbb{E}\left[R_{t+1} + \gamma V^{(j)}(X_{t+1}) | X_t = \boldsymbol{x}\right]$
  end
  $j \leftarrow j+1$;
until $\left\| V^{(j+1)}(\boldsymbol{x}) - V^{(j)}(\boldsymbol{x}) \right\|_\infty < \eta$;

Algorithm 2.2: Q-iteration for general discrete-time stochastic MDPs [18, 98]

input : State space $\mathscr{X}$, action space $\mathscr{U}$, transition dynamics $\tilde{F}$, reward function $\tilde{\rho}$, discount rate γ, threshold η
output: Near-optimal state-action value function $Q_*(\boldsymbol{x}, \boldsymbol{u}) \leftarrow Q^{(j+1)}(\boldsymbol{x}, \boldsymbol{u})$

initialize $Q_0(\boldsymbol{x}, \boldsymbol{u})$;
repeat
  for $\forall \boldsymbol{x} \in \mathscr{X}$, $\forall \boldsymbol{u} \in \mathscr{U}$ **do**
    $Q^{(j+1)}(\boldsymbol{x}, \boldsymbol{u}) \leftarrow \mathbb{E}\left[R_{t+1} + \gamma \max_{\mathscr{U}} Q^{(j)}(X_{t+1}, \boldsymbol{u}') | X_t = \boldsymbol{x}, U_t = \boldsymbol{u}\right]$
  end
  $j \leftarrow j+1$;
until $\left\| Q^{(j+1)}(\boldsymbol{x}, \boldsymbol{u}) - Q^{(j)}(\boldsymbol{x}, \boldsymbol{u}) \right\|_\infty < \eta$;

To complete one iteration, the V-iteration method sweeps through the entire state space $\mathscr{X}$. As the right-hand side of the Bellman optimality equation 2.8g is a contraction mapping with factor $\gamma < 1$, Banach's fixed-point theorem states that the above update rule converges asymptotically to $V_*(\boldsymbol{x})$ as $j \to \infty$ [18, 99]. Upon convergence, the optimal policy can be derived by applying equation 2.9b. However, since convergence to the optimal value function only takes place in the limit, a stopping condition needs to be set that ensures termination of the iteration cycle in finite time. For example, a fixed threshold $\eta > 0$ can be defined that implies convergence to the suboptimal value function if the condition $\left\| V^{(j+1)}(\boldsymbol{x}) - V^{(j)}(\boldsymbol{x}) \right\|_\infty < \eta$ is met, such that $V^{(j+1)}(\boldsymbol{x}) \to V_*(\boldsymbol{x})$ as $\eta \to 0$ [18]. For Q-iteration, an update rule similar to equation 2.10a is defined as:

$$Q^{(j+1)}(\boldsymbol{x}, \boldsymbol{u}) \doteq \mathbb{E}\left[R_{t+1} + \gamma \max_{\mathscr{U}} Q^{(j)}(X_{t+1}, \boldsymbol{u}') | X_t = \boldsymbol{x}, U_t = \boldsymbol{u}\right] \tag{2.10b}$$

The V-iteration and Q-iteration algorithms are summarized in pseudo-code by algorithms 2.1 and 2.2, respectively. Note that by virtue of the Bellman optimality equations, there is no need for an explicit representation of the policy during the iteration sequence. Similarly, the initial value function used for initialization of the algorithm can be set arbitrarily. This implies that, in general, the iterative sequence of value functions may not correspond to any existing policies $\pi_j(\boldsymbol{x})$, $j = 0, 1, 2\ldots$ before convergence of the algorithm, as the method merely operates in value space. Nevertheless, a real policy can always be selected by acting greedily on the intermediate estimate of the optimal value function.

2.3.2. Policy Iteration

The main concept behind Policy Iteration (PI) is formed by evaluating the value function of individual policies, which is subsequently used to find new, better policies. This allows the use of the Bellman expectation equations 2.5d for state-value functions and 2.6c for state-action value functions, which are more tractable computationally compared to the Bellman optimality equations. However, this advantage comes at the cost of additional iteration cycles, as the search towards the optimal policy now consists of evaluating multiple

non-optimal intermediate policies. PI therefore involves two computation cycles, consisting of multiple *policy evaluation* and *policy improvement* steps. For policy evaluation, the following update rule applies for state-values:

$$V_{\pi_j}^{(k+1)}(\boldsymbol{x}) = \mathbb{E}_{\pi_j}\left[R_{t+1} + \gamma V_{\pi_j}^{(k)}(X_{t+1}) | X_t = \boldsymbol{x}\right] \tag{2.11a}$$

And for state-action values:

$$Q_{\pi_j}^{(k+1)}(\boldsymbol{x}, \boldsymbol{u}) = \mathbb{E}_{\pi_j}\left[R_{t+1} + \gamma Q_{\pi_j}^{(k)}(X_{t+1}, U_{t+1}) | X_t = \boldsymbol{x}, U_t = \boldsymbol{u}\right] \tag{2.11b}$$

Similarly to V-iteration and Q-iteration, the above update rules represent a contraction mapping and are therefore guaranteed to converge to the true value function for the policy that is being evaluated, i.e. to $V_{\pi_j}(\boldsymbol{x})$ respectively $Q_{\pi_j}(\boldsymbol{x}, \boldsymbol{u})$ as $k \to \infty$. Note that in the case of discrete state and action spaces, the expectation in the above update rules can be expanded as in the Bellman equations 2.5c and 2.6b, reducing the policy evaluation step to a set of $|\mathscr{X}|$ respectively $|\mathscr{X}||\mathscr{U}|$ linear equations. In case the cardinality of the state and action spaces is reasonably low, policy evaluation can be performed by a single computation involving matrix algebra. This illustrates the advantage of the Bellman equation over the Bellman optimality equation.

Upon convergence of the policy evaluation step, a new policy can be found by acting greedily on the values found for the previous policy. For state values:

$$\pi_{j+1}(\boldsymbol{x}) = \arg\max_{\mathscr{U}} \mathbb{E}_{\pi_j}\left[R_{t+1} + \gamma V_{\pi_j}(X_{t+1}) | X_t = \boldsymbol{x}\right] \tag{2.12a}$$

And for state-action values:

$$\pi_{j+1}(\boldsymbol{x}) = \arg\max_{\mathscr{U}} Q_{\pi_j}(\boldsymbol{x}, \boldsymbol{u}) \tag{2.12b}$$

Algorithms 2.3 and 2.4 summarize the computational strategy behind GPI using state values and state-action values, respectively. This alternating process of finding new, improved policies by acting greedily on the value functions on intermediate policies is guaranteed to converge to the optimal policy. Similar to the case of VI, the policy evaluation step requires a stopping criterion such as a fixed threshold to prevent the time to converge from going to infinity. What is different however, is that convergence towards the true intermediate value functions is actually not necessary for overall convergence of the algorithm. Instead, it is sufficient to establish the trend towards the true value function of the policy that is being evaluated and than acting greedily upon the incomplete estimate. This concept is known as *Generalized Policy Iteration* (GPI), which states that the interactive process of loose value recursion and greedy policy selection is guaranteed to converge as long as the entire state or state-action space is updated throughout the process [98]. In this view, VI forms an extreme example of GPI, as it consistently updates the policy after only a single policy evaluation iteration.

Algorithm 2.3: Policy iteration for general discrete-time stochastic MDPs using state values [18, 98]

input : State space $\mathscr{X}$, action space $\mathscr{U}$, transition dynamics $\tilde{F}$, reward function $\tilde{\rho}$, discount rate γ, thresholds η_π and η_V
output: Near-optimal policy $\pi_*(\boldsymbol{x}) \leftarrow \pi_{j+1}(\boldsymbol{x})$ and state value function $V_*(\boldsymbol{x}) \leftarrow V_{\pi_{j+1}}^{(k+1)}(\boldsymbol{x})$

initialize $\pi_0(\boldsymbol{x})$;
repeat
  initialize $V_{\pi_j}^{(0)}(\boldsymbol{x})$;
  repeat
    for $\forall \boldsymbol{x} \in \mathscr{X}$ **do**
      $V_{\pi_j}^{(k+1)}(\boldsymbol{x}) \leftarrow \mathbb{E}_{\pi_j}\left[R_{t+1} + \gamma V_{\pi_j}^{(k)}(X_{t+1}) | X_t = \boldsymbol{x}\right]$
    end
  until $\left\| V_{\pi_j}^{(k+1)}(\boldsymbol{x}) - V_{\pi_j}^{(k)}(\boldsymbol{x}) \right\|_\infty < \eta_V$;
  $\pi_{j+1}(\boldsymbol{x}) \leftarrow \arg\max_{\mathscr{U}} \mathbb{E}_{\pi_k}\left[R_{t+1} + \gamma V_{\pi_j}^{(k+1)}(X_{t+1}) | X_t = \boldsymbol{x}\right]$;
  $j \leftarrow j + 1$;
until $\left\| \pi_{j+1}(\boldsymbol{x}) - \pi_j(\boldsymbol{x}) \right\|_\infty < \eta_\pi$;

Algorithm 2.4: Policy iteration for general discrete-time stochastic MDPs using state-action values [18, 98]

input : State space $\mathscr{X}$, action space $\mathscr{U}$, transition dynamics $\tilde{F}$, reward function $\tilde{\rho}$, discount rate γ, thresholds η_π and η_Q
output: Near-optimal policy $\pi_*(\boldsymbol{x}) \leftarrow \pi_{j+1}(\boldsymbol{x})$ and state-action value function $Q_*(\boldsymbol{x},\boldsymbol{u}) \leftarrow Q^{(k+1)}_{\pi_{j+1}}(\boldsymbol{x},\boldsymbol{u})$

initialize $\pi_0(\boldsymbol{x})$;
repeat
  initialize $Q^{(0)}_{\pi_j}(\boldsymbol{x},\boldsymbol{u})$;
  repeat
    for $\forall \boldsymbol{x} \in \mathscr{X}$, $\forall \boldsymbol{u} \in \mathscr{U}$ **do**
      $Q^{(k+1)}_{\pi_j}(\boldsymbol{x},\boldsymbol{u}) \leftarrow \mathbb{E}_{\pi_j}\left[R_{t+1} + \gamma Q^{(k)}_{\pi_j}(X_{t+1},U_{t+1}) | X_t = \boldsymbol{x}, U_t = \boldsymbol{u}\right]$
    end
  until $\left\| Q^{(k+1)}_{\pi_{j+1}}(\boldsymbol{x},\boldsymbol{u}) - Q^{(k)}_{\pi_{j+1}}(\boldsymbol{x},\boldsymbol{u}) \right\|_\infty < \eta_Q$;
  $\pi_{j+1}(\boldsymbol{x}) \leftarrow \arg\max_{\mathscr{U}} Q^{(k+1)}_{\pi_j}(\boldsymbol{x},\boldsymbol{u})$;
  $j \leftarrow j+1$;
until $\left\| \pi_{j+1}(\boldsymbol{x}) - \pi_j(\boldsymbol{x}) \right\|_\infty < \eta_\pi$;

2.4. Model-free Value Prediction

DP methods are well-suited for solving MDPs if complete knowledge about the $\langle \mathscr{X}, \mathscr{U}, \tilde{F}, \tilde{\rho}, \gamma \rangle$ tuple is available and can be conveniently represented. If one of these prerequisites is not met, it is generally not possible to perform a full-width backup during each GPI iteration. Therefore, sample-based methods form an appealing alternative. This class of methods uses transition *samples* of (a subset of) the random variables $(X_t, U_t, R_{t+1}, X_{t+1}, U_{t+1})$ obtained from real (or simulated) experience with the controlled process to make an estimate of the true value function. Under the assumption of ergodicity, the expected return represented by the value function can be approximated if all elements of the state and action $\mathscr{X}$ and $\mathscr{U}$ spaces are visited sufficiently often. This implies that the full-width backup used in DP is replaced by a sample-based backup.

In this section, two methods for sample-based estimation of the value function are discussed. The first class is formed by *Monte Carlo* (MC) methods, which take advantage of the simple idea of taking the average of the returns observed after visiting a certain state or state-action pair. The second category is formed by *Temporal Difference* (TD) methods, which use single transition pairs to estimate the value function by means of bootstrapping. These methods are discussed in Subsections 2.4.1 and 2.4.2, respectively. A combination of these methods is also possible, resulting from the idea that a sequence of multiple transition pairs is likely to be more data-efficient than using only a single sample for each update. A well-known method that takes advantage of this idea is known as TD(λ), which is also related to the concept of *eligibility traces*. This material discussed in Subsection 2.4.3. A discussion of how these methods are used in finding the optimal control sequence for a given MDP is subject of the next subsection.

2.4.1. Monte Carlo Learning

Monte Carlo (MC) learning is based on the assumption that the expected return captured by the concept of the value function (Equation 2.5a) can be replaced by the empirical return following the assumption of ergodicity of the random process. In order for the empirical return to be well-defined, MC methods are commonly applied to *episodic* MDPs only, i.e. MDPs that are guaranteed to terminate in finite time. Given a sequence of transition pairs (X_t, U_t, R_{t+1}), $t = 0, 1, 2 \ldots, T$ over the length of a single episode n, the return for a given state is determined in accordance to equation 2.3

$$G^{(n)}_t = \sum_{k=0}^{T-t} \gamma^k R_{t+k+1} \tag{2.13}$$

Subsequently, the value function after encountering the state $\boldsymbol{x}$ N times can be approximated as follows:

$$V_{\bar{\pi}}^{(N)}(\boldsymbol{x}) = \frac{1}{N}\sum_{n=0}^{N} G^{(n)}(\boldsymbol{x}) \tag{2.14}$$

where $G(\boldsymbol{x})$ represents the return observed after encountering state $\boldsymbol{x}$. By virtue of the law of large numbers, this episodic update sequence is guaranteed to converge to $V_{\bar{\pi}}(\boldsymbol{x})$ as $N \to \infty$. The above update rule is known as *every-visit* MC, as each state $\boldsymbol{x}$ is included in the observation history every time it is encountered. A method that is very closely related is *first-visit* MC, which only takes into account the first encounter with state $\boldsymbol{x}$ in each episode. Both approaches can be translated one-to-one to the concept of state-action value functions as well. Note that in contrast to DP methods such as VI and PI, MC learning employs deep backup updates at each iteration to adjust the value function estimate.

MC policy evaluation results in an unbiased estimate of the true value function $V_{\bar{\pi}}(\boldsymbol{x})$. This is because the return from each episode is viewed as an independent, identically distributed estimate of the true value function [98]. However, each update can only take place after an episode terminates. This is a significant downside of MC methods compared to TD methods, which will be discussed next. Another downside is that the estimated value function typically has considerable variance during the initial phases of learning, due to the large variance associated with the return.

2.4.2. Temporal-difference Learning

For non-episodic MDPs, an alternative learning scheme is required that is able to improve the value function estimate continuously. This calls for an recursive update law that adjusts the current estimate based on recent observations of the reward signal. Temporal-difference (TD) learning takes this idea and combines it with the notion of empirical return to update the value function estimate after a given small number of transitions. It is a bootstrapping method that uses the sum of observed reward and the current estimate of the total return as the target of the update rule. The simplest form is known as TD(0)-learning, which only includes the one-step reward R_{t+1} in the TD target. The update rule after j iterations is given as [98]:

$$V_{\pi}^{(j+1)}(X_t) = V_{\pi}^{(j)}(X_t) + \alpha_j \left[R_{t+1} + \gamma V_{\pi}^{(j)}(X_{t+1}) - V_{\pi}^{(j)}(X_t) \right] \tag{2.15a}$$

Here, α_j represents the step-size parameter or learning rate. The term in brackets is known as the TD-error, and forms an indication of how the estimate at iteration j compares to the target estimate [98]:

$$\delta_{t+1}^{(j)} \doteq R_{t+1} + \gamma V_{\pi}^{(j)}(X_{t+1}) - V_{\pi}^{(j)}(X_t) \tag{2.15b}$$

Note that the TD target $R_{t+1} + \gamma V_{\pi}^{(j)}(X_{t+1})$ is biased due to the presence of the estimate [98]. TD learning essentially employs sample-based equivalents of the shallow backup updates that are central to VI and PI methods, making it the least comprehensive type of learning compared to DP and MC methods. However, in case the value function is stored in the form of a table or is represented by a linear function approximator, TD learning is guaranteed to converge to the true value function $V_{\pi}(X_t)$ if all states are visited infinitely often and the following conditions are met by α_j [98]:

$$\sum_{j=0}^{\infty} \alpha_j = \infty \qquad \sum_{j=0}^{\infty} \alpha_j^2 < \infty \tag{2.15c}$$

In practice however, the step-size is often taken as a fixed constant. Although this violates the above condition, this is often necessary if the process is non-stationary or if there is only a limited training set available [99]. Note that an update law similar to Equation 2.15a can be given for state-action values as well.

2.4.3. Eligibility Traces

The elemental concepts behind temporal-difference learning can be extended to enable bootstrapping over multiple time steps, resulting in the notion of n-step TD methods [98]. This has the primary advantage that information from the reward signal is propagated across larger time scales, resulting in more effective learning. This is useful especially in case the reward signal is sparse, or if the process dynamics are slow compared to the sample time [98]. Using the definition of the return from Equation 2.3 and the one-step TD update law from Equation 2.15a, the learning scheme for n-step TD comes down to the following:

$$V_{\pi}^{(j+1)}(X_t) = V_{\pi}^{(j)}(X_t) + \alpha_j \left[\sum_{k=0}^{n-1} \gamma^k R_{t+k+1} + \gamma^n V_{\pi}^{(j)}(X_{t+n}) - V_{\pi}^{(j)}(X_t) \right] \tag{2.16}$$

The estimate of $V_\pi^{(j)}(X_{t+n})$ is only required for relatively distant states. By virtue of the discount rate γ, this means that errors in the current estimate of the value function will have a smaller impact on the updated estimate compared to one-step TD learning. As a result, n-step TD methods are less biased for larger n [98].

Although the concept of using n-step returns greatly improves learning efficiency, the algorithm suffers from a few shortcomings: first, it turns out that every MDP has its own optimal setting for n, which means that the method is not very robust in terms of performance across different tasks; and second, learning is delayed over n transitions as a result of the algorithm's forward view in time [98]. TD(λ) learning addresses both of these issues by taking the average over all n-step TD targets and transforming the learning rule to a backward-view update mechanism. A central concept here is the λ-return, which takes the weighted average of the TD targets over all n to form a new forward-view update target:

$$G_t^\lambda \doteq (1-\lambda)\sum_{n=0}^{\infty}\lambda^{n-1}G_t^{(n)} \tag{2.17a}$$

Here, $\lambda \in [0,1]$. Moreover:

$$G_t^{(n)} \doteq \sum_{k=0}^{n-1}\gamma^k R_{t+k+1} + \gamma^n V_\pi(X_{t+n}) \tag{2.17b}$$

In case $\lambda = 0$, the λ-return reduces to the one-step TD target. Similarly, in the case of episodic MDPs, setting $\lambda = 1$ results in the conventional MC update. Therefore, these are special cases that have also become known as TD(0) and TD(1) learning, respectively.

Using the λ-return directly in the value update law suffers from the same issues as conventional MC-learning, for the reason that one can only calculate the average over all $n \to T$ once the episode has terminated. Eligibility traces provide a way to implement TD(λ) learning mechanistically, i.e. in a fully online fashion. For every state X_t, an eligibility signal $Z_t(\boldsymbol{x}) \in R^+$ is stored based on the frequency at which that state has been encountered, as well as how recent these encounters have been. This results in the following eligibility update at every time step:

$$Z_t(\boldsymbol{x}) = \gamma\lambda Z_{t-1}(\boldsymbol{x}) + \mathbb{1}(X_t = \boldsymbol{x}) \tag{2.18a}$$

Consequently, the eligibility trace is used directly in the update law for the value function for all $\boldsymbol{x} \in \mathscr{X}$:

$$V_\pi^{(j+1)}(\boldsymbol{x}) = V_\pi^{(j)}(\boldsymbol{x}) + \alpha_j \delta_{t+1}^{(j)} Z_t(\boldsymbol{x}) \tag{2.18b}$$

It can be shown that for episodic MDPs, forward- and backward-view TD(λ) generate fully equivalent value function updates [97]. In the online case, the definition of the eligibility trace as presented here only results in an approximation of the forward-view implementation. Exact equivalence for online TD(λ) is possible by using an extended form of the eligibility trace, as presented in [86]. However, this material will not be further discussed here.

2.5. Learning Control

Following the concept of GPI introduced in Subsection 2.3.2, the model-free value prediction algorithms introduced in the previous section can be used to find the optimal control policy for a given MDP in a completely model-free fashion. By acting greedily on (an approximation of) the predicted value function repeatedly, new, better policies can be found. This process ultimately leads to the optimal solution of the MDP. However, since policy evaluation is now based on samples instead of full model-based state sweeps as in DP, always acting greedily in a deterministic sense will generally not lead to the best possible control law. This is a profound difference between DP and RL. Since greedy sample-based GPI will only lead to a very small number of states and actions that are actually visited in the case of non-uniform initialization, the value function will be completely unknown for the majority of the state-action space. Consequently, convergence to the policy that is truly optimal in a global sense can be achieved only if the agent goes through a wide range of different initializations, or if it occasionally takes an action that brings it to little or unexplored parts of the state-action space. Exploration is a key ingredient to finding the optimal policy in RL. However, exploration is also in conflict with the general goal of RL, that is, to find the optimal policy quickly and efficiently: in case exploration is taken too far or too long, learning takes longer than strictly necessary. This is known as the exploration/exploitation dilemma, as already introduced before in Section 2.1. In this section, a standard exploration technique known as ϵ-greedy will be used to fulfill the basic requirement of exploration in RL.

This section is devoted to various techniques that employ the basic idea of sample-based GPI to find optimal control policies for MDPs. The most basic form, known as tabular learning, will be discussed first. Here, simple tables are used to store information about every single state and/or action. This is followed by an introduction to approximate learning techniques, which use compact representations in the form of function approximators for the value function and/or policy. Finally, more advanced techniques such as memory-based and model-based learning will be discussed. These aim to decrease the sample complexity (i.e. number of samples needed) to find the optimal control policy.

2.5.1. Tabular Learning

RL solution methods can generally be categorized along two lines. First, different types of value prediction methods can be used, such as MC learning, TD(0) learning, and TD(λ) methods. Second, there is a distinction between on-policy and off-policy learning methods, where the latter makes use of a behavior policy to evaluate an unrelated target policy. For tabular methods, it is often customary to act upon the state-action values due to the simplicity of the policy update step. It was seen before that the one-step lookahead search for policy updates can be mitigated by adopting the Q-function, as e.g. illustrated by the case of PI methods in Equations 2.11a 2.11b. This is the key to some very powerful and well-known RL algorithms.

On-policy methods will be discussed first, starting with learning based on MC value prediction. One of the most fundamental learning algorithms here is known as MC exploratory starts (ES), where the agent is initialized differently in every episode. In this way, exploration is not encoded in the policy, but can be assured thanks to the episodic nature of MC methods. Policy improvement is done on an episode-by-episode basis. This implies that for intermediate policies, only the general shape of the true value function is revealed; nevertheless, in accordance with the convergence guarantees provided by GPI, the process will converge to the optimal value function and policy as the number of episodes goes to infinity. However, a practical algorithm requires a premature stopping condition, such as a maximum number of iterations or a fixed threshold for changes in the value function. This implies that in general, only a near-optimal solution can be obtained. Note that this limitation applies equally well to Dynamic Programming methods, as discussed in Section 2.3. Algorithm 2.5 summarizes the full algorithm based on MC every-visit value prediction.

It has become clear that for non-episodic MDPs, MC techniques fail to be of any use, meaning that one has to employ TD techniques to achieve convergence to the optimal value function and policy. Using state-action values in the TD(0) update law and acting greedily on intermediate estimates of the value function results in a well-known learning algorithm that has a particular simple form. This algorithm is known as SARSA, which thanks its name to the five-tuple $\langle X_t, U_t, R_{t+1}, X_{t+1}, U_{t+1}\rangle$ that defines the learning rule:

Algorithm 2.5: Monte-Carlo Exploratory Starts (MC-ES) [98]

input : Discount rate γ, max no. episodes n_{max}
output: Near-optimal state-action value function $Q_*(\boldsymbol{x},\boldsymbol{u}) \leftarrow Q_{\pi_n}(\boldsymbol{x},\boldsymbol{u})$ and policy $\pi_*(\boldsymbol{x}) \leftarrow \pi_n(\boldsymbol{x})$, $\forall \boldsymbol{x} \in \mathscr{X}, \forall \boldsymbol{u} \in \mathscr{U}$

initialize $Q_{\pi_0}(\boldsymbol{x},\boldsymbol{u})$, $\pi_0(\boldsymbol{x})$, RETURNS$(\boldsymbol{x},\boldsymbol{u})$ arbitrarily;
$n \leftarrow 0$;
repeat
  sample $\boldsymbol{x}_0 \in \mathscr{X}$, $\boldsymbol{u}_0 \in \mathscr{U}$ from strictly positive PDF;
  run episode n following $\pi_n(\boldsymbol{x})$;
  for *all* $\boldsymbol{x}$, $\boldsymbol{u}$ *seen in episode* **do**
    compute $G(\boldsymbol{x})$;
    RETURNS$(\boldsymbol{x},\boldsymbol{u}) \leftarrow$ append (RETURNS$(\boldsymbol{x},\boldsymbol{u}), G(\boldsymbol{x})$) ;
    $Q_{\pi_n}(\boldsymbol{x},\boldsymbol{u}) \leftarrow$ mean (RETURNS$(\boldsymbol{x},\boldsymbol{u})$) ;
  end
  for *all* $\boldsymbol{x} \in \mathscr{X}$ **do**
    $\pi_{n+1}(\boldsymbol{x}) \leftarrow \arg\max_{\mathscr{U}} Q_{\pi_n}(\boldsymbol{x},\boldsymbol{u})$;
  end
  $n \leftarrow n+1$;
until $n \rightarrow n_{max}$;

Algorithm 2.6: SARSA [18, 98]

input : Discount rate γ, learning schedule $\{\alpha_t\}_{t=0}^{\infty}$, exploration schedule $\{\epsilon_t\}_{t=0}^{\infty}$
output: Near-optimal state-action value function $Q_*(\boldsymbol{x},\boldsymbol{u}) \leftarrow Q^{(t+1)}(\boldsymbol{x},\boldsymbol{u})$ and policy $\pi_*(\boldsymbol{x}) \leftarrow \pi_{t+1}(\boldsymbol{x})$, $\forall \boldsymbol{x} \in \mathscr{X}, \forall \boldsymbol{u} \in \mathscr{U}$

initialize $Q^{(0)}(\boldsymbol{x},\boldsymbol{u})$ arbitrarily;
measure $\boldsymbol{x}_0$;
$\boldsymbol{u}_0 \leftarrow \begin{cases} \operatorname{argmax}_{\mathscr{U}} Q^{(0)}(\boldsymbol{x}_0,\boldsymbol{u}) & \text{with probability } 1-\epsilon \\ \text{random } \boldsymbol{u} \in \mathscr{U} & \text{with probability } \epsilon \end{cases}$;
for $t = 0, 1, 2, \ldots$ **do**
  apply $\boldsymbol{u}_t$, observe $\boldsymbol{x}_{t+1}$, r_{t+1};
  $\boldsymbol{u}_{t+1} \leftarrow \begin{cases} \operatorname{argmax}_{\mathscr{U}} Q^{(t)}(\boldsymbol{x}_{t+1},\boldsymbol{u}) & \text{with probability } 1-\epsilon \\ \text{random } \boldsymbol{u} \in \mathscr{U} & \text{with probability } \epsilon \end{cases}$;
  $Q^{(t+1)}(\boldsymbol{x}_t,\boldsymbol{u}_t) \leftarrow Q^{(t)}(\boldsymbol{x}_t,\boldsymbol{u}_t) + \alpha_t \left[r_{t+1} + \gamma Q^{(t)}(\boldsymbol{x}_{t+1},\boldsymbol{u}_{t+1}) - Q^{(t)}(\boldsymbol{x}_t,\boldsymbol{u}_t) \right]$;
end

$$Q_\pi^{(j+1)}(\boldsymbol{x},\boldsymbol{u}) = Q_\pi^{(j)}(\boldsymbol{x},\boldsymbol{u}) + \alpha_j \left[r_{t+1} + \gamma Q_\pi^{(j)}(\boldsymbol{x}',\boldsymbol{u}') - Q_\pi^{(j)}(\boldsymbol{x},\boldsymbol{u}) \right] \tag{2.19}$$

Updating the policy is done continuously (i.e., $j = t$) by acting greedily on the estimate state-action value function. However, the need for exploration implies that greedy policy improvements will not be sufficient for convergence to the value function and policy. Since different initializations are not applicable in the non-episodic case, one has to adopt exploration strategies such as ϵ-greedy policy updates to ensure coverage of the full state-action space. With ϵ-greedy, actions are selected according to the current estimate of the value function with probability $1-\epsilon$, whereas random actions are generated with probability ϵ. With this strategy, SARSA leads to the optimal solution of the MDP with probability one, given that all state-actions pairs are encountered infinitely often and that the policy is fully greedy in the limit (i.e., as $t \to \infty$) [98]. Algorithm 2.6 illustrates the algorithm for the non-episodic case.

Consequently, the basic SARSA algorithm can be extended with the advantages offered by the n-step return and eligibility traces that define the TD(λ) value prediction technique. This leads to an alternative algo-

Algorithm 2.7: SARSA(λ) [98]

input : Discount rate γ, learning schedule $\{\alpha_t\}_{t=0}^{\infty}$, exploration schedule $\{\epsilon_t\}_{t=0}^{\infty}$, return weight λ
output: Near-optimal state-action value function $Q_*(\boldsymbol{x},\boldsymbol{u}) \leftarrow Q^{(t+1)}(\boldsymbol{x},\boldsymbol{u})$ and policy $\pi_*(\boldsymbol{x}) \leftarrow \pi_{t+1}(\boldsymbol{x})$, $\forall \boldsymbol{x} \in \mathscr{X}, \forall \boldsymbol{u} \in \mathscr{U}$

initialize $Q^{(0)}(\boldsymbol{x},\boldsymbol{u})$ arbitrarily;
initialize $Z_0(\boldsymbol{x},\boldsymbol{u}) = 0$ $\forall \boldsymbol{x} \in \mathscr{X}$, $\forall \boldsymbol{u} \in \mathscr{U}$;
measure $\boldsymbol{x}_0$;
$\boldsymbol{u}_0 \leftarrow \begin{cases} \operatorname{argmax}_{\mathscr{U}} Q^{(0)}(\boldsymbol{x}_0,\boldsymbol{u}) & \text{with probability } 1-\epsilon \\ \text{random } \boldsymbol{u} \in \mathscr{U} & \text{with probability } \epsilon \end{cases}$;
for $t = 0, 1, 2, \ldots$ **do**
  apply $\boldsymbol{u}_t$, observe $\boldsymbol{x}_{t+1}$, r_{t+1};
  $\boldsymbol{u}_{t+1} \leftarrow \begin{cases} \operatorname{argmax}_{\mathscr{U}} Q^{(t)}(\boldsymbol{x}_{t+1},\boldsymbol{u}) & \text{with probability } 1-\epsilon \\ \text{random } \boldsymbol{u} \in \mathscr{U} & \text{with probability } \epsilon \end{cases}$;
  $\delta_{t+1} \leftarrow r_{t+1} + \gamma Q^{(t)}(\boldsymbol{x}_{t+1},\boldsymbol{u}_{t+1}) - Q^{(t)}(\boldsymbol{x}_t,\boldsymbol{u}_t)$;
  $Z_{t+1}(\boldsymbol{x}_t,\boldsymbol{u}_t) \leftarrow Z_t(\boldsymbol{x}_t,\boldsymbol{u}_t) + 1$;
  for *all* $\boldsymbol{x} \in \mathscr{X}$, $\boldsymbol{u} \in \mathscr{U}$ **do**
    $Q^{(t+1)}(\boldsymbol{x},\boldsymbol{u}) \leftarrow Q^{(t)}(\boldsymbol{x},\boldsymbol{u}) + \alpha_t \delta_{t+1} Z_{t+1}(\boldsymbol{x},\boldsymbol{u})$;
    $Z_{t+1}(\boldsymbol{x},\boldsymbol{u}) \leftarrow \gamma\lambda Z_{t+1}(\boldsymbol{x},\boldsymbol{u})$;
  end
end

rithm known as SARSA(λ), presented as Algorithm 2.7. Note that this algorithm is much more expensive computationally as a result of the need to sweep through the entire state-action space at every time step. However, the sample complexity is considerably lower thanks to the enhanced information propagation properties of the λ-return.

The attention will now be drawn to off-policy learning methods. With this kind of techniques, it is possible to learn about a target policy $\tilde{\pi}(\boldsymbol{u}|\boldsymbol{x})$ while following some other policy $\tilde{\pi}'(\boldsymbol{u}|\boldsymbol{x})$, known as the behavior policy. The target policy can be deterministic, such as the optimal policy $\pi_*(\boldsymbol{x})$. Off-policy learning enables some appealing opportunities, such as learning the optimal policy from a-priori (i.e. fixed) data generated using some other, sub-optimal policy. This requires a technique for estimating the value function for $\tilde{\pi}(\boldsymbol{u}|\boldsymbol{x})$ from the returns generated by the behavior policy $\tilde{\pi}'(\boldsymbol{u}|\boldsymbol{x})$. In the case of MC control, this can be achieved by taking the weighted average of the observed returns, based on the relative probability that these returns occur under the target and the behavior policy [98]. Consequently, each return must be scaled with the probability that the full trajectory after the first encounter with state X_t occurs under the target and behavior policies. Denoting T_t as the random variable that describes the trajectory $U_t, X_{t+1}, U_{t+1}, \ldots, X_T$ and τ as the observed instance, the following scaling factor applies in the general case of a stochastic target policy [98]:

$$\frac{\mathbb{P}_n(T_t=\tau)}{\mathbb{P}'_n(T_t=\tau)} = \frac{\prod_{k=t}^{T-1}\tilde{\pi}(\boldsymbol{u}|\boldsymbol{x})\tilde{F}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{x}')}{\prod_{k=t}^{T-1}\tilde{\pi}'(\boldsymbol{u}|\boldsymbol{x})\tilde{F}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{x}')} = \prod_{k=t}^{T-1}\frac{\tilde{\pi}(\boldsymbol{u}|\boldsymbol{x})}{\tilde{\pi}'(\boldsymbol{u}|\boldsymbol{x})} \tag{2.20}$$

with n referring to the n-th episode of the learning procedure. Note that this ratio exists only if $\mathbb{P}'_n(T_t=\tau)\neq 0$, which implies that $\tilde{\pi}'(\boldsymbol{u}|\boldsymbol{x})>0\ \forall \boldsymbol{x}\in\mathscr{X}$. This requirement is met if the behavior policy is e.g. an ϵ-greedy policy. Silver explains that this procedure is also known as importance sampling.

Consequently, the full off-policy MC learning algorithm for estimating the optimal policy $\pi(\boldsymbol{x})$ can be given as Algorithm 2.8. Note that estimating the value function of a given target policy is only possible if the probability of a given action selected under that policy is nonzero[2]. For the deterministic target (optimal)

[2] If it would be zero, the numerator in equation 2.20 would be driven to zero as well, resulting in that pair to be excluded from the weighted average.

Algorithm 2.8: Off-policy Monte-Carlo [98]

input : Discount rate γ, max no. episodes n_{max}, behavior policy exploration schedule $\{\epsilon_t\}_{t=0}^{T}$: $\epsilon>0\ \forall t\in[0,T]$
output: Near-optimal state-action value function $Q_*(\boldsymbol{x},\boldsymbol{u})\leftarrow Q_{\pi_n}(\boldsymbol{x},\boldsymbol{u})$ and policy $\pi_*(\boldsymbol{x})\leftarrow\pi_n(\boldsymbol{x}),\ \forall\boldsymbol{x}\in\mathscr{X},\forall\boldsymbol{u}\in\mathscr{U}$

initialize $Q_{\pi_0}(\boldsymbol{x},\boldsymbol{u})$, $\pi_0(\boldsymbol{x})$ arbitrarily, $p(\boldsymbol{x},\boldsymbol{u})\leftarrow 0$, $q(\boldsymbol{x},\boldsymbol{u})\leftarrow 0$;
$n\leftarrow 0$;
repeat
  run episode n following $\tilde{\pi}'(\boldsymbol{u}|\boldsymbol{x})$;
  set t_0 as last time for which $\boldsymbol{u}_t\neq\pi(\boldsymbol{x})$;
  for *all* $\boldsymbol{x}$, $\boldsymbol{u}$ *seen in episode* **do**
    if *time of first encounter* $t>t_0$ **then**
      $W_t\leftarrow\prod_{k=t}^{T-1}\frac{1}{\tilde{\pi}'(\boldsymbol{u}|\boldsymbol{x})}$;
      $p(\boldsymbol{x},\boldsymbol{u})\leftarrow p(\boldsymbol{x},\boldsymbol{u})+W_tG_t$;
      $q(\boldsymbol{x},\boldsymbol{u})\leftarrow q(\boldsymbol{x},\boldsymbol{u})+W_t$;
      $Q_{\pi_n}(\boldsymbol{x},\boldsymbol{u})\leftarrow\frac{p(\boldsymbol{x},\boldsymbol{u})}{q(\boldsymbol{x},\boldsymbol{u})}$;
    end
  end
  for *all* $\boldsymbol{x}\in\mathscr{X}$ **do**
    $\pi_n(\boldsymbol{x})=\arg\max_{\mathscr{U}}Q_{\pi_n}(\boldsymbol{x},\boldsymbol{u})$;
  end
  $n\leftarrow n+1$;
until $n\rightarrow n_{max}$;

Algorithm 2.9: Q-learning [18, 98]

input : Discount rate γ, learning schedule $\{\alpha_t\}_{t=0}^{\infty}$, exploration schedule $\{\epsilon_t\}_{t=0}^{\infty}$
output: Near-optimal state-action value function $Q_*(\boldsymbol{x},\boldsymbol{u}) \leftarrow Q^{(t+1)}(\boldsymbol{x},\boldsymbol{u})$ and policy $\pi_*(\boldsymbol{x}) \leftarrow \pi_{t+1}(\boldsymbol{x})$, $\forall \boldsymbol{x} \in \mathscr{X}, \forall \boldsymbol{u} \in \mathscr{U}$

initialize $Q^{(0)}(\boldsymbol{x},\boldsymbol{u})$ arbitrarily;
measure $\boldsymbol{x}_0$;
for $t = 0, 1, 2, \ldots$ **do**
  $\boldsymbol{u}_t \leftarrow \begin{cases} \arg\max_{\mathscr{U}} Q^{(t)}(\boldsymbol{x}_t,\boldsymbol{u}) & \text{with probability } 1-\epsilon \\ \text{random } \boldsymbol{u} \in \mathscr{U} & \text{with probability } \epsilon \end{cases}$;
  apply $\boldsymbol{u}_t$, observe $\boldsymbol{x}_{t+1}$, r_{t+1};
  $Q^{(t+1)}(\boldsymbol{x}_t,\boldsymbol{u}_t) \leftarrow Q^{(t)}(\boldsymbol{x}_t,\boldsymbol{u}_t) + \alpha_t\left[r_{t+1} + \gamma \max_{\mathscr{U}} Q^{(t)}(\boldsymbol{x}_{t+1},\boldsymbol{u}') - Q^{(t)}(\boldsymbol{x}_t,\boldsymbol{u}_t)\right]$;
end

policy considered here, this condition is violated for any actions that are selected are than those that are greedy under the current target. Therefore, the algorithm is restricted to learning only about trajectories that occur after the last non-greedy action has been selected. Another consequence is that the numerator in the weighting scale is equal to one.

Off-policy learning can be translated equally well to TD methods. For TD(0)-learning, this results in the very well-known Q-learning algorithm, invented by Watkins [111]. If the optimal policy $\pi(\boldsymbol{x})$ is again selected as the target policy, the update law for Q-learning reduces to a form that is very similar to that of the basic SARSA algorithm (Equation 2.19):

$$Q_\pi^{(j+1)}(\boldsymbol{x},\boldsymbol{u}) = Q_\pi^{(j)}(\boldsymbol{x},\boldsymbol{u}) + \alpha_j\left[r_{t+1} + \gamma \max_{\mathscr{U}} Q_\pi^{(j)}(\boldsymbol{x}',\boldsymbol{u}') - Q_\pi^{(j)}(\boldsymbol{x},\boldsymbol{u})\right] \tag{2.21}$$

Similarly to SARSA, the policy is updated continuously by acting greedily on the current estimate of the optimal state-action value function. Alternatively, Q-learning can also be viewed as a stochastic sample-based

Algorithm 2.10: Watkins' $Q(\lambda)$ [97]

input : Discount rate γ, learning schedule $\{\alpha_t\}_{t=0}^{\infty}$, exploration schedule $\{\epsilon_t\}_{t=0}^{\infty}$, return weight λ
output: Near-optimal state-action value function $Q_*(\boldsymbol{x},\boldsymbol{u}) \leftarrow Q^{(t+1)}(\boldsymbol{x},\boldsymbol{u})$ $\forall \boldsymbol{x} \in \mathscr{X}, \forall \boldsymbol{u} \in \mathscr{U}$

initialize $Q^{(0)}(\boldsymbol{x},\boldsymbol{u})$ arbitrarily;
measure $\boldsymbol{x}_0$;
$\boldsymbol{u}_0 \leftarrow \begin{cases} \arg\max_{\mathscr{U}} Q_{\pi_0}(\boldsymbol{x}_0,\boldsymbol{u}) & \text{with probability } 1-\epsilon \\ \text{random } \boldsymbol{u} \in \mathscr{U} & \text{with probability } \epsilon \end{cases}$;
for $t = 0, 1, 2, \ldots$ **do**
  apply $\boldsymbol{u}_t$, observe $\boldsymbol{x}_{t+1}$, r_{t+1};
  $\boldsymbol{u}_{t+1} \leftarrow \begin{cases} \arg\max_{\mathscr{U}} Q^{(t)}(\boldsymbol{x}_{t+1},\boldsymbol{u}) & \text{with probability } 1-\epsilon \\ \text{random } \boldsymbol{u} \in \mathscr{U} & \text{with probability } \epsilon \end{cases}$;
  $\boldsymbol{u}^* \leftarrow \arg\max_{\mathscr{U}} Q^{(t)}(\boldsymbol{x}_{t+1},\boldsymbol{u}')$;
  $\delta_{t+1} \leftarrow r_{t+1} + \gamma Q^{(t)}(\boldsymbol{x}_{t+1},\boldsymbol{u}^*) - Q^{(t)}(\boldsymbol{x}_t,\boldsymbol{u}_t)$;
  $Z_{t+1}(\boldsymbol{x}_t,\boldsymbol{u}_t) \leftarrow Z_t(\boldsymbol{x}_t,\boldsymbol{u}_t) + 1$;
  for *all* $\boldsymbol{x} \in \mathscr{X}$, $\boldsymbol{u} \in \mathscr{U}$ **do**
    $Q^{(t+1)}(\boldsymbol{x},\boldsymbol{u}) \leftarrow Q^{(t)}(\boldsymbol{x},\boldsymbol{u}) + \alpha_t \delta_{t+1} Z_{t+1}(\boldsymbol{x},\boldsymbol{u})$;
    if $\boldsymbol{u}_{t+1} = \boldsymbol{u}^*$ **then**
      $Z_{t+1}(\boldsymbol{x},\boldsymbol{u}) \leftarrow \gamma\lambda Z_{t+1}(\boldsymbol{x},\boldsymbol{u})$;
    else
      $Z_{t+1}(\boldsymbol{x},\boldsymbol{u}) \leftarrow 0$
    end
  end
end

form of VI (see Algorithm 2.2) [18]. Algorithm 2.9 presents the full algorithm for the case of non-episodic MDPs. Note that the learning rate and exploration must comply with the same requirements as for SARSA.

The last case that will be considered here is off-policy learning using TD(λ) value prediction. This method is known as Watkins' $Q(\lambda)$ learning, and makes use of advantages offered by the λ-return to obtain a faster estimate of the optimal state-action value function. However, it can only do so to a certain extent as a result of the off-policy estimation strategy that is central to Q-learning. Since the trajectory that follows after an exploratory action cannot be traced back to the target policy, the λ-return is valid only for those state-action sequences that are generated up to an exploratory action. Nevertheless, ϵ-greedy behavior policies with small values for ϵ will exhibit a considerable increase in learning efficiency, although not as large as for SARSA(λ). Algorithm 2.10 presents the backward implementation of Watkins' $Q(\lambda)$ using eligibility traces. Note that the issue of broken eligibility traces is dealt with by setting $Z_{t+1}(\boldsymbol{x},\boldsymbol{u})$ equal to zero $\forall\ \boldsymbol{x}\in\mathscr{X},\ \boldsymbol{u}\in\mathscr{U}$ if the selected input for the next time step is off-policy.

It must be mentioned that there exists an alternative to Watkins' $Q(\lambda)$ in in the form of Peng's $Q(\lambda)$ [97]. This method aims to make full use of the advantages offered by eligibility traces and in the case of off-policy TD-learning, but is less intuitive. However, further discussion of this material is outside of the scope of this thesis.

2.5.2. Approximate Learning

The tabular learning methods presented in the previous section are suitable for MDPs where the state-action space is discrete and the cardinality $|\mathscr{X}||\mathscr{U}|$ remains within manageable proportions. However, as the cardinality of a given learning problem increases, the memory requirements for storing the value function for every distinct state-action pair $(\boldsymbol{x},\boldsymbol{u})$ start to grow exponentially. This is true for any of the DP and RL methods presented so far in this chapter, and is known as the *curse of dimensionality* [18]. In case the MDP involves continuous state and/or action spaces, tabular methods are no longer able to represent the value function exactly, but can only render some gridded approximation to it. The curse of dimensionality then dictates that there is a limit to the tabular resolution that can be achieved.

Yet, the unfitness of tabular methods for this class of MDPs is not limited to severe memory requirements only [98]. It generally has a large impact on the total amount of data that is required for learning as well (i.e., sample complexity). A state-action pair that is encountered once will only be visited again with very low probability[3]. This also implies that there are large regions of the state-action space for which little or no data is available, even though good estimates could be found based on experience from similar state-action pairs. This calls for methods that *generalize* limited experience to larger subsets of the state-action space. [18, 98].

Based on this discussion, there is a strong need for compact representations of the value function that significantly reduce the requirements for storage memory and sample complexity. *Function approximation* (FA) provides the necessary tools here. Broadly speaking, a distinction can be made between parametric approximation and non-parametric approximation methods [18]. For now, the focus will be on parametric implementations. Parametric approximation essentially approximates some (unknown) target set by an a-priori specified functional form $F(\boldsymbol{w})$, where $\boldsymbol{w}\in\mathbb{R}^d$ represents the weight vector [18, 98]. This functional form can take any shape, such as a linear combination of basis functions, or an artificial neural network (ANN). For now, the functional $F(\boldsymbol{w})$ is considered as any mapping $F:\mathbb{R}^d\mapsto\mathscr{V}$ or $F:\mathbb{R}^d\mapsto\mathscr{Q}$, where $\mathscr{V}$ and $\mathscr{Q}$ represent the space of attainable state- and state-action value functions, respectively [18]. Consequently, the following notation applies [18]:

$$\hat{V}_\pi(\boldsymbol{x}) \doteq [F(\boldsymbol{w})](\boldsymbol{x}) \tag{2.22a}$$

$$\hat{Q}_\pi(\boldsymbol{x},\boldsymbol{u}) \doteq [F(\boldsymbol{w})](\boldsymbol{x},\boldsymbol{u}) \tag{2.22b}$$

These approximation mappings change the nature of the backup actions that are central to the value prediction step. The value updates now take place on the scale of a complete subset associated with a given weight $w\in\boldsymbol{w}$, as opposed to single values only[4]. Each backup target now serves as a training sample that follows from an unknown distribution, which calls for supervised learning methods [98]. Consequently, the objective of learning can then be formulated as minimizing the squared error between the true value function and the approximated estimate, which can be achieved through e.g. gradient descent [98]:

[3] In the continuous case, this probability equals zero.

[4] Silver explains that a table is in fact a special case of function 'approximation', where each distinct element in the universal set is described by a single weight.

$$\begin{aligned} \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \tfrac{1}{2}\alpha_t \nabla \left[V_\pi(\boldsymbol{x}) - \hat{V}_\pi^{(t)}(\boldsymbol{x}) \right]^2 \\ &= \boldsymbol{w}_t + \alpha_t \left[V_\pi(\boldsymbol{x}) - \hat{V}_\pi^{(t)}(\boldsymbol{x}) \right] \nabla \hat{V}_\pi^{(t)}(\boldsymbol{x}) \end{aligned} \tag{2.23a}$$

where $\nabla \hat{V}_\pi^{(t)}(\boldsymbol{x}) = \frac{\partial \hat{V}_\pi^{(t)}(\boldsymbol{x})}{\partial \boldsymbol{w}_t}$. In the case of linearly parametrized functionals, i.e. when $F(\boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})$, the above expression reduces to a particular simple form:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_t \left[V_\pi(\boldsymbol{x}) - \boldsymbol{w}_t^T \boldsymbol{\phi}(\boldsymbol{x}) \right] \boldsymbol{\phi}(\boldsymbol{x}) \tag{2.23b}$$

However, since the true value function is unknown, this formulation as is has little use. Other update targets need to be used instead, such as the observed return $G(\boldsymbol{x})$ in MC learning, the one-step TD error in TD(0) learning, or the λ-return for n-step TD methods. In the case of TD(0) learning for example, value prediction with function approximation comes down to the following:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_t \left[r_{t+1} + \gamma \hat{V}_\pi^{(t)}(\boldsymbol{x}_{t+1}) - \hat{V}_\pi^{(t)}(\boldsymbol{x}_t) \right] \nabla \hat{V}_\pi^{(t)}(\boldsymbol{x}_t) \tag{2.23c}$$

This update law is in fact a biased approximation to stochastic gradient descent (SGD), due to the neglected contribution of the bootstrapped target in the computation of the gradient and the fact that the target itself is biased. As a result, convergence tends to be less robust compared to tabular value prediction [98]. Nevertheless, Equation 2.23c has a strikingly similar form as Equation 2.15a. Note that global convergence of the SGD update law to the optimal weight vector $\boldsymbol{w}_*$ can only be assured for convex value functions approximated by convex functionals. Otherwise, the value update step may end up in a local optimum for $\boldsymbol{w}$. For the non-convex case, this can be resolved by meaningful initializations for the weight vector close to the global optimum. Curriculum learning forms a considerably more efficient alternative, as explained in Chapter 4.

For convex learning tasks, one can do significantly better in terms of sample complexity by using least-squares regression to find the weight vector instead of SGD. Recall from Subsection 2.3.2 that the Bellman expectation equation in the policy evaluation step of PI is completely linear, and that it may be solved directly from the set of $|\mathscr{X}|$ linear equations if the MDP has a discrete state space. A key insight here is that linear function approximation allows for a stochastic, sample-based approximation to this approach for high-cardinality and continuous MDPs. This can be illustrated as follows. Given a training distribution $\mathscr{D} : \left\{ \left\langle \boldsymbol{x}_t, \hat{V}_\pi^{(t_0)}(\boldsymbol{x}_t) \right\rangle \right\}_{t=t_0}^{t_0+T}$, Silver explains that the expected weight update at convergence is zero, i.e. $\mathbb{E}_{\mathscr{D}}[\Delta \boldsymbol{w}] = 0$. If the update step is written in matrix form, an explicit expression for $\boldsymbol{w}$ can be obtained. By writing $\Phi = [\boldsymbol{\phi}(\boldsymbol{x}_{t_0}), \boldsymbol{\phi}(\boldsymbol{x}_{t_0+1}), \ldots, \boldsymbol{\phi}(\boldsymbol{x}_{t_0+T-1})]$ and $\Phi' = [\boldsymbol{\phi}(\boldsymbol{x}_{t_0+1}), \boldsymbol{\phi}(\boldsymbol{x}_{t_0+2}), \ldots, \boldsymbol{\phi}(\boldsymbol{x}_{t_0+T})]$, the following can be derived:

$$\begin{aligned} \mathbf{0} = \Phi \left[\boldsymbol{r} + \gamma \boldsymbol{w}^T \Phi' - \boldsymbol{w}^T \Phi \right]^T &\rightarrow \mathbf{0} = \Phi \boldsymbol{r}^T + \gamma \Phi \Phi'^T \boldsymbol{w} - \Phi \Phi^T \boldsymbol{w} \rightarrow \Phi \boldsymbol{r}^T = \left[\Phi \Phi^T - \gamma \Phi \Phi'^T \right] \boldsymbol{w} \\ &\rightarrow \boldsymbol{w} = \left[\Phi \left(\Phi - \gamma \Phi' \right)^T \right]^{-1} \Phi \boldsymbol{r}^T \end{aligned} \tag{2.24}$$

where it should be $T \geq d$ for the existence of the matrix inversion. This is known as least-squares TD (LSTD) learning [16, 53], a one-shot algorithm that returns the true value function related to a given policy from a single computation step. Although the sample complexity is reduced to a minimum with LSTD, the computational complexity that is involved is higher than for SGD methods. Incremental forms known as RLSTD [16] and LSPE [18, 75] also exist, which are not bound to batch operations and have lower computational complexities as a result. Note that these methods can also be directly applied for the evaluation of state-action value functions. Extended versions involving eligibility traces have been developed as well, which has become known as the LSTD(λ) and LSPE(λ) class of algorithms.

The compact approximation methods for value function prediction can be applied in a relatively straightforward manner to the on-policy control methods presented for the tabular case. Algorithm 2.11 presents SARSA in the case of approximate learning based on SGD. Similar algorithms can be given for approximate MC learning and SARSA(λ) as well. In this case, every update will adjust the value function over a complete subset of the state-action space, and the effect may even be global if a global functional is used. A greedy policy update is performed on every intermediate estimation of the approximate value function as part of the general process of GPI. In the case of sufficiently small action spaces, the policy improvement step

Algorithm 2.11: SARSA with parametric value function approximation [18, 98]

input : Discount rate γ, learning schedule $\{\alpha_t\}_{t=0}^{\infty}$, exploration schedule $\{\epsilon_t\}_{t=0}^{\infty}$, functional $F(\boldsymbol{w})$
output: Near-optimal state-action value function $Q_*(\boldsymbol{x},\boldsymbol{u}) \leftarrow Q^{(t+1)}(\boldsymbol{x},\boldsymbol{u})\ \forall \boldsymbol{x} \in \mathscr{X}, \forall \boldsymbol{u} \in \mathscr{U}$

initialize weight vector $\boldsymbol{w}_0$;
$\hat{Q}^{(0)}(\boldsymbol{x},\boldsymbol{u}) \leftarrow [F(\boldsymbol{w}_0)](\boldsymbol{x},\boldsymbol{u})$;
measure $\boldsymbol{x}_0$;
$\boldsymbol{u}_0 \leftarrow \begin{cases} \arg\max_{\mathscr{U}} \hat{Q}^{(0)}(\boldsymbol{x}_0,\boldsymbol{u}) & \text{with probability } 1-\epsilon \\ \text{random } \boldsymbol{u} \in \mathscr{U} & \text{with probability } \epsilon \end{cases}$;
for $t = 0, 1, 2, \ldots$ **do**
  apply $\boldsymbol{u}_t$, observe $\boldsymbol{x}_{t+1}$, r_{t+1};
  $\boldsymbol{u}_{t+1} \leftarrow \begin{cases} \arg\max_{\mathscr{U}} \hat{Q}^{(t)}(\boldsymbol{x}_{t+1},\boldsymbol{u}) & \text{with probability } 1-\epsilon \\ \text{random } \boldsymbol{u} \in \mathscr{U} & \text{with probability } \epsilon \end{cases}$;
  $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \alpha_t \left[r_{t+1} + \gamma \hat{Q}^{(t)}(\boldsymbol{x}_{t+1},\boldsymbol{u}_{t+1}) - \hat{Q}^{(t)}(\boldsymbol{x}_t,\boldsymbol{u}_t) \right] \nabla \hat{Q}^{(t)}(\boldsymbol{x}_t,\boldsymbol{u}_t)$;
end

can be straightforwardly implemented through enumeration of the available inputs. However, in the case of high-cardinality or continuous action spaces, enumeration suffers from the same curse of dimensionality as discussed earlier, which makes the policy improvement step nontrivial. In these cases, a separate, explicit representation for the policy is required.

Policy approximation can be done along similar lines as value function approximation, i.e. through a given approximation mapping such as a linear combination of basis function or an ANN. Approximately greedy policy improvement can then be performed by updating the parametrized policy towards actions that maximize the value function [18]. Note that if the policy is updated incrementally towards the greedy one (e.g., via SGD), this has the advantage that the policy is not driven too far off in case of inadequate intermediate estimate of the value function [99]. This combination of separate memory structures for the value function and policy is also known as actor-critic schemes [99] or *adaptive-critic designs* (ACDs) [81, 112]. Here, the policy is often referred to as the *actor*, whereas the value function is known as the *critic*. Chapter 3 presents a more elaborate treatment of ACDs and its various forms. For now, it is assumed that the policy improvement step results in a policy that is (to a large extent) greedy in the latest estimate of the value function.

Likewise, approximate on-policy control can be easily combined with least-squares policy evaluation methods as well. In the model-free control case, this calls for variants of LSTD (or LSPE) that evaluate state-action value functions, known as LSTD-Q or LSQ [18, 53]. The resulting algorithm is known as least-squares policy iteration (LSPI), which can be implemented in different forms. For example, one can apply LSPI in an off-line batch setting, but also in episodic and non-episodic processes that are interacted with online. Algorithm 2.12 shows an online LSPI algorithm with fixed policy improvement intervals. As opposed to SGD methods, the computational complexity of the policy evaluation step may prevent the algorithm from running in real-time in case d is high. Incremental methods such as RLSTD and LSPE may therefore be better candidates in this respect. However, in case learning does not result in unstable behavior[5], real-time policy updates are not strictly necessary, which reduces the computational requirements in this respect.

From here, it is a relatively small step to approximate learning for off-policy control methods. However, despite their widespread popularity, these methods can suffer from considerable convergence issues as a result of the "deadly triad" of function approximation, bootstrapping, and off-policy target distributions [98]. These problems are in fact not unique to the control case, but already appear in the policy prediction step. Nevertheless, two off-policy algorithms will be illustrated here for completeness. Algorithm 2.13 presents Q-learning in the case of approximate learning based on SGD. In principle, a follow-up algorithm based on eligibility traces can be derived as well. Similarly, a least-squares implementation for linearly parametrized functionals is presented as Algorithm 2.14. This algorithm is known as least-squares Q-learning (LSQL). Contrary to LSPI, LSQL makes use of bootstrapping again to evaluate the policy. This introduces a bias in the policy evaluation step, the reason being that the previous estimate of the value function is used to approximate the estimated return for the updated policy [53]. Note that LSQL can be regarded as an online implementation of least-squares fitted Q-iteration [18].

[5] This comes down to the assumption that the learning process is *safe*, as is discussed in Chapter 5

Algorithm 2.12: Online LSPI with fixed policy improvement intervals [18, 53]

input : Discount rate γ, exploration schedule $\{\epsilon_t\}_{t=0}^{\infty}$, basis function vector $\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{u})$, policy update interval T_π
output: Near-optimal state-action value function $Q_*(\boldsymbol{x},\boldsymbol{u}) \leftarrow Q_{\pi_{j+1}}(\boldsymbol{x},\boldsymbol{u})$ and policy $\pi_*(\boldsymbol{x}) \leftarrow \pi_{j+1}(\boldsymbol{x})$, $\forall \boldsymbol{x} \in \mathscr{X}, \forall \boldsymbol{u} \in \mathscr{U}$

initialize $\pi_0(\boldsymbol{x})$ arbitrarily;
initialize empty arrays for Φ_1, Φ'_1, $\boldsymbol{r}_1$;
$j \leftarrow 0$;
measure $\boldsymbol{x}_0$;
$\boldsymbol{u}_0 \leftarrow \begin{cases} \pi_j(\boldsymbol{x}_0) & \text{with probability } 1-\epsilon \\ \text{random } \boldsymbol{u} \in \mathscr{U} & \text{with probability } \epsilon \end{cases}$;
for $t = 0, 1, 2, \ldots$ **do**
  apply $\boldsymbol{u}_t$, observe $\boldsymbol{x}_{t+1}$, r_{t+1};
  $\Phi_{j+1} \leftarrow \text{append}\left(\Phi_{j+1}, \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{u}_t)\right)$;
  $\boldsymbol{r}_{j+1} \leftarrow \text{append}\left(\boldsymbol{r}_{j+1}, r_{t+1}\right)$;
  $\boldsymbol{u}_{t+1} \leftarrow \begin{cases} \pi_j(\boldsymbol{x}_{t+1}) & \text{with probability } 1-\epsilon \\ \text{random } \boldsymbol{u} \in \mathscr{U} & \text{with probability } \epsilon \end{cases}$;
  $\Phi'_{j+1} \leftarrow \text{append}\left(\Phi'_{j+1}, \boldsymbol{\phi}(\boldsymbol{x}_{t+1}, \boldsymbol{u}_{t+1})\right)$;
  if $t = (j+1)T_\pi$ **then**
    $\boldsymbol{w}_{j+1} = \left[\Phi_{j+1}\left(\Phi_{j+1} - \gamma\Phi'_{j+1}\right)^T\right]^{-1} \Phi_{j+1}\boldsymbol{r}_{j+1}^T$;
    $\pi_{j+1}(\boldsymbol{x}) \leftarrow \text{arg max}_{\mathscr{U}}\, \boldsymbol{w}_{j+1}^T \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{u})\ \forall \boldsymbol{x} \in \mathscr{X}$;
    $j \leftarrow j+1$;
  end
end

So far, the discussion on DP and RL methods has been limited to the case where the nature of the policy is deterministic. Stochastic behavior to ensure exploration was encoded separately. In general however, policies can as well be stochastic, as was seen in Section 2.2. Silver demonstrates that there are cases where it is beneficial to explicitly parametrize a stochastic policy through a parameter vector $\boldsymbol{\theta}$, denoted as $\tilde{\pi}_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) \in (0,1)$ [98]. For example, in some MDPs, the optimal policy is a stochastic one, which means that acting greedily on an estimated value function will not lead to an adequate solution (as the resulting policy will always be deterministic). Similarly, for some MDPs, learning a parametrized policy directly and thereby bypassing a value function may considerably reduce sample complexity [98]. This has resulted in direct *policy-gradient* learning methods, which directly optimizes the policy according to some performance measure $J(\boldsymbol{\theta})$ [98]. An example of such an algorithm is REINFORCE [113], which directly performs stochastic gradient descent (or

Algorithm 2.13: Q-learning with parametric value function approximation [18]

input : Discount rate γ, learning schedule $\{\alpha_t\}_{t=0}^{\infty}$, exploration schedule $\{\epsilon_t\}_{t=0}^{\infty}$, functional $F(\boldsymbol{w})$
output: Near-optimal state-action value function $Q_*(\boldsymbol{x},\boldsymbol{u}) \leftarrow Q^{(t+1)}(\boldsymbol{x},\boldsymbol{u})\ \forall \boldsymbol{x} \in \mathscr{X}, \forall \boldsymbol{u} \in \mathscr{U}$

initialize weight vector $\boldsymbol{w}_0$;
$\hat{Q}^{(0)}(\boldsymbol{x},\boldsymbol{u}) \leftarrow [F(\boldsymbol{w}_0)](\boldsymbol{x},\boldsymbol{u})$;
measure $\boldsymbol{x}_0$;
for $t = 0, 1, 2, \ldots$ **do**
  $\boldsymbol{u}_t \leftarrow \begin{cases} \text{arg max}_{\mathscr{U}}\, \hat{Q}^{(t)}(\boldsymbol{x}_t,\boldsymbol{u}) & \text{with probability } 1-\epsilon \\ \text{random } \boldsymbol{u} \in \mathscr{U} & \text{with probability } \epsilon \end{cases}$;
  apply $\boldsymbol{u}_t$, observe $\boldsymbol{x}_{t+1}$, r_{t+1};
  $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_t\left[r_{t+1} + \gamma \text{max}_{\mathscr{U}}\, \hat{Q}^{(t)}(\boldsymbol{x}_{t+1},\boldsymbol{u}') - \hat{Q}^{(t)}(\boldsymbol{x}_t,\boldsymbol{u}_t)\right]\nabla\hat{Q}^{(t)}(\boldsymbol{x}_t,\boldsymbol{u}_t)$;
end

Algorithm 2.14: Online LSQL with fixed policy improvement intervals [52, 53]

input : Discount rate γ, exploration schedule $\{\epsilon_t\}_{t=0}^{\infty}$, basis function vector $\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{u})$, policy update interval T_π
output: Near-optimal state-action value function $Q_*(\boldsymbol{x},\boldsymbol{u}) \leftarrow Q_{\pi_{j+1}}(\boldsymbol{x},\boldsymbol{u})$ and policy $\pi_*(\boldsymbol{x}) \leftarrow \pi_{j+1}(\boldsymbol{x})$, $\forall \boldsymbol{x} \in \mathscr{X}, \forall \boldsymbol{u} \in \mathscr{U}$

initialize $\pi_0(\boldsymbol{x})$ arbitrarily;
initialize empty arrays for Φ_1, $\boldsymbol{q}_1$;
measure $\boldsymbol{x}_0$;
$j \leftarrow 0$;
for $t = 0, 1, 2, \ldots$ **do**
  $\boldsymbol{r}_{j+1} \leftarrow \text{append}\left(\boldsymbol{r}_{j+1}, r_{t+1}\right)$;
  $\boldsymbol{u}_t \leftarrow \begin{cases} \pi_j(\boldsymbol{x})_t & \text{with probability } 1-\epsilon \\ \text{random } \boldsymbol{u} \in \mathscr{U} & \text{with probability } \epsilon \end{cases}$;
  $\Phi_{j+1} \leftarrow \text{append}\left(\Phi_{j+1}, \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{u}_t)\right)$;
  apply $\boldsymbol{u}_t$, observe $\boldsymbol{x}_{t+1}$, r_{t+1};
  $\boldsymbol{q}_{j+1} \leftarrow \text{append}\left(\boldsymbol{q}_{j+1}, \left[r_{t+1} + \gamma \max_{\mathscr{U}} \boldsymbol{w}_j^T \boldsymbol{\phi}(\boldsymbol{x}_{t+1}, \boldsymbol{u}')\right]\right)$;
  if $t = (j+1)T_\pi$ **then**
    $\boldsymbol{w}_{j+1} = \left[\Phi_{j+1}\Phi_{j+1}^T\right]^{-1} \Phi_{j+1}\boldsymbol{q}_{j+1}^T$;
    $\pi_{j+1}(\boldsymbol{x}) \leftarrow \arg\max_{\mathscr{U}} \boldsymbol{w}_{j+1}^T \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{u})\ \forall \boldsymbol{x} \in \mathscr{X}$;
    $j \leftarrow j+1$;
  end
end

ascent, for that matter) based on the return observed during a single episode:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t \gamma G_t \nabla \ln \tilde{\pi}_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) \tag{2.25}$$

Where the latter term follows from the well-known policy-gradient theorem [29]. As this type of learning does not involve a critic, it has become known under the class of *actor-only* methods [29]. However, actor-only algorithms are notorious for their high variance during learning, which is reflected by the use of the total return in the gradient update. This variance can be considerably reduced by subtracting a baseline from the observed return in the form of the value function. Similarly, the total return can be replaced again by a bootstrapping estimate. This results in the well-known class of *policy-gradient* actor-critic methods [29]. This is not to be confused with actor-critics in the context of adaptive critic schemes, which can in fact be regarded as a form of critic-only methods for the reason that they are fully deterministic by nature.

So far, the discussion on approximate learning has focused entirely on parametric forms of function approximation. Selecting some functional a-priori for the value function and policy is a strong form of encoding domain knowledge in the learning process. By exploiting prior knowledge of the problem, the learning process can be made considerably more efficient. However, this knowledge can be difficult to obtain in general, which calls for function approximations with high approximation capabilities such as ANNs. Alternatively, one can adopt non-parametric approximations, which derive good representations directly from data stored in memory. These are highly flexible methods, as they return an estimate directly for a given *query point* based on similar data points. As a result, they are not constrained to some specified global functional. However, their complexity tends to grow considerably if the number of data points stored in memory increases [18]. All the ideas presented in this section can essentially be combined with non-parametric FAs as well.

2.5.3. Model-based Learning

The fundamental idea of applying sample-based GPI or policy gradient methods proves to be a very powerful approach for finding optimal control policies for MDPs in a completely model-free fashion. However, the biggest disadvantage of the class of reinforcement learning algorithms considered so far, especially those that cannot take advantage of the opportunities that arise in convex learning tasks, is that they are notoriously sample-inefficient [2, 30, 98]. The primary reason for this is that samples are discarded immediately after they

have been used to update the value function and/or policy. In situations where these algorithms are applied directly on existing data sets, this does not lead to real practical issues other than considerable learning times. In online applications however, using purely model-free RL implies that valuable experience gets lost, which results in much longer interaction times with the system before a minimum level of performance is reached. This is a classical problem in the reinforcement learning domain, as also mentioned in Chapter 1.

A powerful strategy to considerably enhance the sample efficiency of canonical reinforcement learning methods is to use a *model* of the dynamics $\tilde{F}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')$ (and possibly the reward signal $\tilde{\rho}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')$) of the process or system the agent is interacting with [2, 30, 98]. This is also known as *model-based reinforcement learning*. Generally, there are different motivations for using models in direct combination with the traditionally model-free view of RL methods, although they all have in common that they aim to reduce the amount of data required for learning. This has spurred the development of a large variety of different approaches, which, consequently, has also resulted in an overloading of the terminology. First, models can be adopted to generate *simulated experience* [96, 98]. There are multiple ways as to how the model and simulated samples can be used. Let $\hat{\tilde{F}}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')$ be a (generally stochastic) model of the true dynamics $\tilde{F}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')$. A basic approach would be to identify $\hat{\tilde{F}}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')$ fully a-priori, and use this model to learn the optimal control policy in a simulation setting. With this strategy, the learning process would become largely analogous to the concept of dynamic programming, although the model is not used to perform full-width backups. However, this technique becomes particularly troublesome if the a-priori model turns out to be an insufficient representation of reality. This is also related to the problem of *distributional shift* [6, 65]. A more advanced approach would be to learn $\hat{\tilde{F}}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')$ using online experience, and use intermediate model estimates to find new, better policies. This can be regarded as an indirect approach to reinforcement learning, as real experience is not used directly for updating the value function and/or policy [98]. Although the fact that the model is generated online implies that the problem of distribution mismatch is largely resolved, this method may still suffer from large performance issues if the model representation is not able to capture vital real-world phenomena. Therefore, a hybrid approach where both simulated and real experience are used for finding new policies would be preferred. *Dyna* [96] is a famous method that does so exactly. Here, real experience is used to update the model, value function, and policy at the same time, whereas the learned model is used to generate simulated experience in addition. A similar idea is adopted in [2], where an a-priori, inaccurate model is updated (augmented) based on real experience, and this updated model is used to identify approximate directions for improvement of the policy.

Note that the idea of identifying a process model fully online (i.e., without any a-priori knowledge) and using this to expedite learning is still considered a form of model-based reinforcement learning. Nevertheless, some occurrences in literature (e.g., as in [117]) argue that these approaches can be classified under the family of model-free methods. This is one example of how the terminology has become overloaded over time.

A second motivation for adopting models is to enhance the update accuracy of policy search methods. For this class of methods, the model is not used to simulated virtual experience, but to generate a better estimate of the policy gradient. One example here is the model-learning actor-critic (MLAC) [30] framework, which learns a model to get a better estimate of how the policy can be updated to ascend on the performance measure $J(\boldsymbol{\theta})$. Subsection 3.3.3 provides an explanation of how this idea can be taken advantage of in the context of adaptive critic designs. A key idea here is to not learn a global model, which can be highly demanding in terms of training complexity, but use a locally linear model instead. Another example of how policy search methods can be advanced by means of learned process models is presented in [57]. Here, time-varying, locally linear models are fitted on sampled trajectories of a highly nonlinear process that is controlled by a stabilizing, time-varying linear Gaussian controller. An iterative sequence of local model identification and local policy design results stable exploration of the system dynamics, while each intermediate local policy serves as an intermediate 'example policy' for training a global nonlinear policy. This process is also known as *guided policy search*. By virtue of online identification of local models, the process of optimizing a global policy becomes considerably more efficient and effective.

Finally, a third motivation for using model-based reinforcement learning is to explicitly reason about future steps by rolling out imagined trajectories based on $\tilde{F}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')$ and use these to plan a sequence of inputs. This integrates the idea of planning in the reinforcement learning framework [98]. Note that for this class of methods, simulation is not (exclusively) used as an additional source of training data, but serves primarily to plan future actions given the current state of the process. The use of model-based predictions is also often seen in the context of safe learning, as described in Section 5.2.

3

Adaptive Optimal Control

In the previous chapter, the fundamental concepts and theories in reinforcement learning were introduced. In this chapter, a more focused view is adopted where the goal is to find optimal control laws for systems that can be described in terms of a system of (non)-linear difference equations. Reinforcement learning in this context is often referred to as *Approximate/Adaptive Dynamic Programming* (ADP), due to the general need to approximate the value function and policy (see Subsection 2.5.2). In the context of control engineering, the ideas put forth by RL/ADP can be used to unite optimal control theory with adaptive control [43]. Traditionally, these two fields have different theoretical notions when it comes to designing feedback controllers [59]: traditional optimal controllers are often not adaptive, whereas adaptive controllers are often not optimal. Whereas indirect methods based on online identification of the system dynamics can be used to reconfigure the control law by performing model-based optimization of some performance index, RL/ADP methods offer a *direct* alternative. From this perspective, RL/ADP can also be viewed as *direct adaptive optimal control* [95]. Consequently, the concepts from the previous chapter can be translated to the control of linear and nonlinear systems, such as a UAV.

This chapter is structured as follows. First, the problem of optimal regulation and tracking control for affine discrete-time systems is formulated in Section 3.1. This forms a bridge to Section 3.2, where methods for solving the linear quadratic tracking (LQT) problem are presented. Note that LQT essentially arises as a special case. Subsequently, methods for solving the more general problem of optimal tracking for nonlinear systems will be presented in Section 3.3. The chapter ends with a brief discussion on practical issues and implementation guidelines.

3.1. Optimal Control for Affine Discrete-Time Systems

In general, any (deterministic) dynamical system that is affine in the input can be described mathematically as a system of first-order ODEs in the following way:

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u} \tag{3.1a}$$

with $f(\boldsymbol{x})$ representing the internal dynamics, and $g(\boldsymbol{x})$ the input dynamics. With the majority of the work done in RL/ADP based on discrete-time formulations, the continuous-time dynamics need to be sampled or discretized to arrive at an expression that fits with the discrete-time sequential decision-making problem introduced in Section 2.1 [58]. This results in the following standard form for control-affine discrete-time dynamical systems:

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t) + g(\boldsymbol{x}_t)\boldsymbol{u}_t \tag{3.1b}$$

Under the assumption that this system is locally Lipschitz continuous and that is it stabilizable [43, 61], a policy $\hat{\pi}_*(\boldsymbol{x})$ can be designed that is optimal according to some performance objective, represented as the value function $\hat{V}_*(\boldsymbol{x})$ or $\hat{Q}_*(\boldsymbol{x},\boldsymbol{u})$. As will be shown, the DP/RL mechanisms presented in the previous chapter offer the tools to establish such an optimal policy. In optimal control, the two major objectives are formed by optimal regulation and optimal tracking control [43]. These require different formulations of the optimal control problem, as discussed in the following subsections.

3.1.1. The Regulation Problem

In optimal regulation, the goal is to bring all states or outputs of a dynamical system to zero such that some performance index in optimized. In general, this performance index can take any shape, but it is often defined as a utility that is quadratic in the state and input variables [58]:

$$r(\boldsymbol{x}_t, \boldsymbol{u}_t) = \boldsymbol{x}_t^T Q \boldsymbol{x}_t + \boldsymbol{u}_t^T R \boldsymbol{u}_t \tag{3.2a}$$

Note that a similar utility can be given based on observed outputs. Since $\lim_{t\to\infty} r(\boldsymbol{x}_t, \boldsymbol{u}_t) = 0$ under a stabilizing policy, following (undiscounted) definition of the value function applies [43, 58]:

$$V_\pi(\boldsymbol{x}) \doteq \sum_{t=0}^{\infty} r(\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{3.2b}$$

Consequently, the optimal value function can be written according to the Bellman optimality equation specified as Equation 2.8g:

$$V_*(\boldsymbol{x}_t) = \min_{\boldsymbol{u}} \left[r(\boldsymbol{x}_t, \boldsymbol{u}_t) + V_*(\boldsymbol{x}_{t+1}) \right] \tag{3.2c}$$

In optimal control theory and ADP, this equation is also known as the Hamilton-Jacobi-Bellman (HJB) equation [43, 58]. Subsequently, the optimal control policy $\pi_*(\boldsymbol{x})$ can be derived directly from the discrete-time formulation of the system dynamics [5]. Setting $\frac{\partial V_*(\boldsymbol{x}_t)}{\partial \boldsymbol{u}_t} = 0$ and expanding the partial derivative using the chain rule, the following is obtained [5, 58]:

$$\boldsymbol{u}_t^* = -\frac{1}{2} R^{-1} g^T(\boldsymbol{x}_t) \frac{\partial V_*(\boldsymbol{x}_{t+1})}{\partial \boldsymbol{x}_{t+1}} \tag{3.2d}$$

As discussed in Section 2.3, the nonlinear nature of the HJB equation calls for an iterative approach to find $V_*(\boldsymbol{x}_t)$. Note that in this case, the optimal policy is essentially an adaptive feedback control law.

3.1.2. The Tracking Problem

In optimal tracking control, the objective is to let the system state $\boldsymbol{x}_t$ follow a reference trajectory $\boldsymbol{x}_t^r \in \mathbb{R}^n$ in a way that optimizes a performance index. In the standard formulation, the optimal control consists of a feedforward and a feedback term, that together regulate the tracking error $\boldsymbol{e}_t = \boldsymbol{x}_t - \boldsymbol{x}_t^r$ to zero [43]. The feedforward term can be determined through dynamic inversion (NDI) of the system dynamics:

$$\boldsymbol{u}_t^d = g(\boldsymbol{x}_t^r)^+ \left[\boldsymbol{x}_t^r - f(\boldsymbol{x}_t^r) \right] \tag{3.3a}$$

where $g(\boldsymbol{x}_t^r)^+$ represents the generalized inverse of $g(\boldsymbol{x}_t^r)$, i.e. $g(\boldsymbol{x}_t^r)^+ = \left[g^T(\boldsymbol{x}_t^r) g(\boldsymbol{x}_t^r) \right]^{-1} g^T(\boldsymbol{x}_t^r)$ [45]. Consequently, the feedback term can be obtained by minimizing the accumulated utility, with the utility given as [43]:

$$r(\boldsymbol{e}_t, \boldsymbol{u}_t^e) = \boldsymbol{e}_t^T Q_e \boldsymbol{e}_t + \left(\boldsymbol{u}_t^e \right)^T R_e \boldsymbol{u}_t^e \tag{3.3b}$$

Consequently, the HJB equation can be specified as in Equation 3.2c. The optimal feedback control can then be derived along the same lines as before, resulting in [43]:

$$\left(\boldsymbol{u}_t^e \right)^* = -\frac{1}{2} R_e^{-1} g^T(\boldsymbol{x}_t) \frac{\partial V_*(\boldsymbol{e}_{t+1})}{\partial \boldsymbol{e}_{t+1}} \tag{3.3c}$$

The optimal tracking control is then formed by the sum of the feedforward and feedback terms [43]:

$$\boldsymbol{u}_t^* = \boldsymbol{u}_t^d + \left(\boldsymbol{u}_t^e \right)^* \tag{3.3d}$$

However, this standard formulation of the optimal tracking control requires complete knowledge of the system's internal and input dynamics. An alternative formulation is presented in [41, 43–45], which does not require knowledge about the system's internal dynamics. It is based on the concept of augmented system dynamics, where the reference trajectory $\boldsymbol{x}_t^r \in \mathbb{R}^p$ is assumed to be generated by some form of command generator:

$$\boldsymbol{x}_{t+1}^r = \Psi(\boldsymbol{x}_t^r) \tag{3.4a}$$

Based on this equation and Equation 3.1b, an augmented formulation of the error dynamics can be established [43]:

$$\begin{bmatrix} \boldsymbol{e}_{t+1} \\ \boldsymbol{x}^r_{t+1} \end{bmatrix} = \begin{bmatrix} f(\boldsymbol{e}_{t+1} + \boldsymbol{x}^r_{t+1}) - \Psi(\boldsymbol{x}^r_t) \\ \Psi(\boldsymbol{x}^r_t) \end{bmatrix} + \begin{bmatrix} g(\boldsymbol{e}_{t+1} + \boldsymbol{x}^r_{t+1}) \\ \mathbf{0} \end{bmatrix} \boldsymbol{u}_t \tag{3.4b}$$

Which can also be written as:

$$\boldsymbol{X}_{t+1} = F(\boldsymbol{X}_t) + G(\boldsymbol{X}_t)\boldsymbol{u}_t \tag{3.4c}$$

with $\boldsymbol{X}_t \doteq \begin{bmatrix} \boldsymbol{e}_t^T & \left(\boldsymbol{x}^r_t\right)^T \end{bmatrix}^T$. Subsequently, the utility can be redefined as [43]:

$$r(\boldsymbol{X}_t, \boldsymbol{u}_t) = \boldsymbol{X}_t^T Q_T \boldsymbol{X}_t + \boldsymbol{u}_t^T R \boldsymbol{u}_t \tag{3.4d}$$

where $Q_T \doteq \begin{bmatrix} Q & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$. Since $\lim_{t\to\infty} r(\boldsymbol{X}_t, \boldsymbol{u}_t) \neq 0$ for an optimal tracking policy, the (discounted) HJB equation comes down to the following [43]:

$$V_*(\boldsymbol{X}_t) = \min_{\boldsymbol{u}} \left[r(\boldsymbol{X}_t, \boldsymbol{u}_t) + \gamma V_*(\boldsymbol{X}_{t+1}) \right] \tag{3.4e}$$

which results in the following expression for the optimal tracking control [43]:

$$\boldsymbol{u}^*_t = -\frac{1}{2}\gamma R^{-1} G^T(\boldsymbol{X}_t) \frac{\partial V_*(\boldsymbol{X}_{t+1})}{\partial \boldsymbol{X}_{t+1}} \tag{3.4f}$$

This alternative formulation of the optimal tracking problem only requires knowledge about the augmented input dynamics $G^T(\boldsymbol{X}_t)$, as well as the dimensionality of the state space and reference trajectory dynamics. As explained in [41], the latter assumption is generally not too restrictive, as most types of reference dynamics can be generated according to some command generator model. Note that the resulting optimal control does not involve any feedforward terms, in contrast to the standard formulation.

3.2. Linear Quadratic Tracking

If the system dynamics are completely linear, a special case arises for solving the optimal control policy. Assume that the system dynamics can be formulated as:

$$\boldsymbol{x}_{t+1} = A\boldsymbol{x}_t + B\boldsymbol{u}_t \tag{3.5a}$$

$$\boldsymbol{y}_t = C\boldsymbol{x}_t \tag{3.5b}$$

For this special case, it is known that for optimal regulation, the shape of the value function is quadratic in the state for any fixed policy $\pi(\boldsymbol{x}_t)$, i.e. [41, 43, 58]:

$$V_\pi(\boldsymbol{x}_t) = \boldsymbol{x}_t^T P \boldsymbol{x}_t \tag{3.5c}$$

With P known as the kernel matrix. This is known as the Linear Quadratic Regulation (LQR) case. This representation of the value function forms a good example of how prior knowledge can be used in establishing powerful forms of function approximation (see Subsection 2.5.2). By substituting this equation in the HJB equation 3.2c together with the system dynamics, one obtains the discrete-time Algebraic Ricatti Equation (ARE; without discounting), as demonstrated in e.g. [43, 58]:

$$Q - P + A^T PA - A^T PB(R + B^T PB)^{-1} B^T PA = 0 \tag{3.5d}$$

Solving this equation for P results in the optimal value function $V_*(\boldsymbol{x}_t)\ \forall \boldsymbol{x} \in \mathscr{X}$. This results in the following optimal control, which is linear in the state [43, 58]:

$$\boldsymbol{u}^*_t = -(R + B^T PB)^{-1} B^T PA\boldsymbol{x}_t \tag{3.5e}$$

This is a well-known result, that requires complete knowledge of the internal dynamics and input dynamics, and can only be solved off-line. A similar result can be found for the case of Linear Quadratic Tracking (LQT), as demonstrated in [44] and repeated in Subsection 3.2.1. The remainder of this section will discuss how the LQT ARE equation can be solved on-line with decreasing knowledge of the system dynamics.

3.2.1. The Discrete-time Algebraic Riccati Equation

Following the general formulation of the tracking problem in 3.1.2, the LQT problem can be written in augmented state form. In [44], it is assumed that the command generator dynamics are fully linear, i.e. that $\Psi\left(\boldsymbol{x}_t^r\right) = F\boldsymbol{x}_t^r$. Consequently, the following augmented system description is proposed:

$$\begin{bmatrix}\boldsymbol{x}_{t+1}\\ \boldsymbol{x}_{t+1}^r\end{bmatrix} = \begin{bmatrix}A & \mathbf{0}\\ \mathbf{0} & F\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_t\\ \boldsymbol{x}_t^r\end{bmatrix} + \begin{bmatrix}B\\ \mathbf{0}\end{bmatrix}\boldsymbol{u}_t \tag{3.6a}$$

Which can be concisely written as [44]:

$$\boldsymbol{X}_{t+1} = T\boldsymbol{X}_t + B_1\boldsymbol{u}_t \tag{3.6b}$$

Given that the dimensionality of the state space is generally different from the dimensionality of the reference dynamics, the utility can be written as [44]:

$$r(\boldsymbol{X}_t, \boldsymbol{u}_t) = \left(C\boldsymbol{x}_t - \boldsymbol{x}_t^r\right)^T Q\left(C\boldsymbol{x}_t - \boldsymbol{x}_t^r\right) + \boldsymbol{u}_t^T R\boldsymbol{u}_t \tag{3.6c}$$

which can be written more concisely as:

$$r(\boldsymbol{X}_t, \boldsymbol{u}_t) = \boldsymbol{X}_t^T Q_1 \boldsymbol{X}_t + \boldsymbol{u}_t^T R\boldsymbol{u}_t \tag{3.6d}$$

where $Q_1 \doteq \begin{bmatrix}C_T QC & -C^T Q\\ -QC & C\end{bmatrix}$. It is demonstrated in [44] that in this case, the shape of the value function is quadratic in the augmented state:

$$V(\boldsymbol{X}_t) = \boldsymbol{X}_t^T P\boldsymbol{X}_t \tag{3.6e}$$

This makes the problem largely analogous to the LQR case. Consequently, the equivalent ARE for the LQT problem can be derived to be [44]:

$$Q_1 - P + \gamma T^T PT - \gamma^2 T^T PB_1(R + B_1^T PB_1)^{-1}B_1^T PT = 0 \tag{3.6f}$$

With the optimal tracking control following as:

$$\boldsymbol{u}_t^* = -(R + B_1^T PB_1)^{-1}\gamma B_1^T PT\boldsymbol{X}_t \tag{3.6g}$$

It is explained in [44] that the presence of the discount rate γ prevents the tracking error from becoming truly zero in the limit. However, it can be driven to arbitrarily small values if γ is set to small values and large values for Q are selected.

3.2.2. Off-line Lyapunov Iteration

Due to its implicit form, the ARE is generally difficult to solve. An iterative approach can be used instead, as proposed in [42, 44]. Here, the control policy from Equation 3.6g is rewritten in compact form as $\boldsymbol{u} = -K_1\boldsymbol{x}_t$. The optimal control can be then found by computing the kernel matrix P for intermediate policies, which is subsequently used to update the policy in the policy improvement step. This is a process known as off-line Lyapunov iteration [46], which is essentially a form of dynamic programming for optimal control (see Section 2.3). For policy evaluation, the ARE is rewritten as [44]:

$$Q_1 - P + K_1^T RK_1 + \gamma(T - B_1K_1)^T P(T - B_1K_1) = 0 \tag{3.7}$$

As explained in [46], solving the LQT ARE through DP may be done via either policy iteration (PI) or value iteration (VI). For PI, the Lyapunov Equation 3.7 is solved in full, as shown in Algorithm 3.1. For this algorithm, it is generally required that the initial control policy is stabilizing [42, 58]. Under the principles of GPI however, it is generally not required to find the true value function for each of the intermediate policies. As explained in Subsection 2.3.2, establishing the general trend towards the true value function and acting greedily on this intermediate estimate is sufficient for convergence to the optimal policy. The extreme case is formed by VI. The VI iteration process for the LQT ARE is illustrated in Algorithm 3.2. Contrary to PI, this approach does not require the initial feedback matrix K_1^0 to be stabilizing [42, 58]. Note that this solution method is also known as Lyapunov recursion [58].

Algorithm 3.1: Off-line Lyapunov policy iteration (PI) for LQT ARE [42, 44]

input : Internal dynamics A, input dynamics B, command generator dynamics F, augmented state weight matrix Q_1, input weight matrix R, discount rate γ, threshold η
output: Near-optimal kernel matrix P, near-optimal control feedback matrix K_1

initialize K_1^0 as arbitrary stabilizing control feedback matrix;
$j \leftarrow 0$;
repeat
  solve $P^{j+1} = Q_1 + \left(K_1^j\right)^T R K_1^j + \gamma(T - B_1 K_1^j)^T P^{j+1}(T - B_1 K_1^j)$;
  $K_1^{j+1} \leftarrow (R + B_1^T P^{j+1} B_1)^{-1} \gamma B_1^T P^{j+1} T$;
  $j \leftarrow j + 1$
until $\left\| P^{j+1} - P^j \right\|_\infty < \eta$;

Algorithm 3.2: Off-line Lyapunov value iteration (VI) for LQT ARE [42]

input : Internal dynamics A, input dynamics B, command generator dynamics F, augmented state weight matrix Q_1, input weight matrix R, discount rate γ, threshold η
output: Near-optimal kernel matrix P, near-optimal control feedback matrix K_1

initialize K_1^0 as arbitrary control feedback matrix;
$j \leftarrow 0$;
repeat
  solve $P^{j+1} = Q_1 + \left(K_1^j\right)^T R K_1^j + \gamma(T - B_1 K_1^j)^T P^j(T - B_1 K_1^j)$;
  $K_1^{j+1} \leftarrow (R + B_1^T P^{j+1} B_1)^{-1} \gamma B_1^T P^{j+1} T$;
  $j \leftarrow j + 1$
until $\left\| P^{j+1} - P^j \right\|_\infty < \eta$;

3.2.3. On-line Generalized Policy Iteration

In the previous section, DP was used to find the solution to the LQT ARE. An on-line alternative can be specified that taps into the data-driven nature of RL solution methods by directly solving the Bellman equation using samples. This calls for sample-based GPI methods that are similar in nature as the RL methods introduced in Subsection 2.5.1. In [44, 46], sample-based GPI is used in the policy evaluation step to find the kernel matrix P. The resulting on-line PI and VI algorithms are given as Algorithms 3.3 and 3.4, respectively. Note that both algorithms still require full knowledge about the system dynamics in the policy improvement step.

There are multiple ways to find the kernel matrix in the policy evaluation step. As explained in [58], it is convenient to write the value function in a form that is affine in the augmented state:

$$\boldsymbol{X}_t^T P \boldsymbol{X}_t = \mathrm{vec}(P)^T (\boldsymbol{X}_t \otimes \boldsymbol{X}_t) = \bar{P}^T \bar{\boldsymbol{X}}_t \tag{3.8}$$

with $\bar{P} \in \mathbb{R}^{(n+m)^2}$, $\bar{\boldsymbol{X}} \in \mathbb{R}^{(n+m)^2}$ and $\otimes$ representing the Kronecker product. However, the redundant (symmetric) terms can be removed, which reduces the dimensionality to $(n+m)(n+m+1)/2$ [58]. Subsequently, the kernel matrix can be found through either batch LS or RLS, which makes the policy evaluation step essentially equivalent to LSTD and RLSTD, respectively (see Subsection 2.5.2). Alternatively, one can employ stochastic gradient descent. However, this can significantly slow down learning due to the increased sample complexity.

3.2.4. On-line Q-learning

Due to their dependency on the state value function, the online RL algorithms presented in the previous section still require a full model description to perform the one-step look-ahead in the policy update step. It was seen in Section 2.2 that this model dependency can be eliminated by adopting the state-action value function (see Equations 2.9a and 2.9b). In [44], the state-action value function is exploited in the LQT problem definition, which gives rise to an iterative solution method that finds the optimal control policy without any knowledge of the system dynamics or command generator dynamics. Their findings are briefly summarized

Algorithm 3.3: On-line (data-driven) policy iteration (PI) for LQT ARE [44, 46]

input : Internal dynamics A, input dynamics B, command generator dynamics F, augmented state weight matrix Q_1, input weight matrix R, discount rate γ, threshold η
output: Near-optimal kernel matrix P, near-optimal control feedback matrix K_1

initialize K_1^0 as arbitrary stabilizing control feedback matrix;
$j \leftarrow 0$;
repeat
  solve $\boldsymbol{X}_t^T P^{j+1} \boldsymbol{X}_t = \boldsymbol{X}_t^T \left(Q_1 + \left(K_1^j\right)^T R K_1^j \right) + \gamma \boldsymbol{X}_{t+1}^T P^{j+1} \boldsymbol{X}_{t+1}$;
  $K_1^{j+1} \leftarrow (R + B_1^T P^{j+1} B_1)^{-1} \gamma B_1^T P^{j+1} T$;
  $j \leftarrow j+1$
until $\left\| P^{j+1} - P^j \right\|_\infty < \eta$;

Algorithm 3.4: On-line (data-driven) value iteration (VI) for LQT ARE; based on [42, 44, 46]

input : Internal dynamics A, input dynamics B, command generator dynamics F, augmented state weight matrix Q_1, input weight matrix R, discount rate γ, threshold η
output: Near-optimal kernel matrix P, near-optimal control feedback matrix K_1

initialize K_1^0 as arbitrary control feedback matrix;
$j \leftarrow 0$;
repeat
  solve $\boldsymbol{X}_t^T P^{j+1} \boldsymbol{X}_t = \boldsymbol{X}_t^T \left(Q_1 + \left(K_1^j\right)^T R K_1^j \right) + \gamma \boldsymbol{X}_{t+1}^T P^{j} \boldsymbol{X}_{t+1}$;
  $K_1^{j+1} \leftarrow (R + B_1^T P^{j+1} B_1)^{-1} \gamma B_1^T P^{j+1} T$;
  $j \leftarrow j+1$
until $\left\| P^{j+1} - P^j \right\|_\infty < \eta$;

here. Based on the definition of the utility in Equation 3.6d, the Bellman equation for state-action values can be written as follows:

$$Q(\boldsymbol{X}_t, \boldsymbol{u}_t) = \boldsymbol{X}_t^T Q_1 \boldsymbol{X}_t + \boldsymbol{u}_t^T R \boldsymbol{u}_t + \gamma \boldsymbol{X}_t^T P \boldsymbol{X}_t \tag{3.9a}$$

It can be shown that an equivalent form of this expression can be written as follows:

$$Q(\boldsymbol{X}_t, \boldsymbol{u}_t) = \begin{bmatrix} \boldsymbol{X}_t \\ \boldsymbol{u}_t \end{bmatrix}^T \begin{bmatrix} Q_1 + \gamma T^T P T & \gamma T^T P B_1 \\ \gamma B_1^T P T & R + \gamma B_1^T P B_1 \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_t \\ \boldsymbol{u}_t \end{bmatrix} \tag{3.9b}$$

This results in a new definition of the kernel matrix, denoted as H:

$$Q(\boldsymbol{X}_t, \boldsymbol{u}_t) = \begin{bmatrix} \boldsymbol{X}_t \\ \boldsymbol{u}_t \end{bmatrix}^T \begin{bmatrix} H_{XX} & H_{Xu} \\ H_{uX} & H_{uu} \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_t \\ \boldsymbol{u}_t \end{bmatrix} \tag{3.9c}$$

Consequently, the optimal tracking control is again defined as the one that maximizes the optimal kernel matrix. By setting $\frac{\partial Q_*(\boldsymbol{X}_t, \boldsymbol{u}_t)}{\partial \boldsymbol{u}_t} = 0$, the following expression is obtained:

$$\boldsymbol{u}_t^* = -H_{uu}^{-1} H_{uX} \boldsymbol{X}_t \tag{3.9d}$$

which does not involve any model terms. Writing $\boldsymbol{Z}_t = \begin{bmatrix} \boldsymbol{X}_t^T & \boldsymbol{u}_t^T \end{bmatrix}^T$, two completely model-free algorithms for solving the LQT problem are given in Algorithms 3.5 and 3.6. If the policy evaluation step is done via btach least-squares, these algorithms are equivalent to the LSPI-Q and LSQL algorithms presented in Algorithms 2.12 and 2.14, respectively. Note that the intermediate feedback matrix K_1^j is not directly used in the Bellman equation, as opposed to the algorithms presented earlier in this section. This is to accommodate for an additional exploration input, which is required to ensure persistence of excitation (PE) of the system dynamics

Algorithm 3.5: On-line (data-driven) policy iteration (PI) for LQT based on Q-functions [44]

input : Dimension $n+m+p$ of the augmented state-action space, augmented state weight matrix Q_1, input weight matrix R, discount rate γ, threshold η
output: Near-optimal kernel matrix H, near-optimal control feedback matrix K_1

initialize K_1^0 as arbitrary stabilizing control feedback matrix;
$j \leftarrow 0$;
repeat
  solve $\boldsymbol{Z}_t^T H^{j+1} \boldsymbol{Z}_t = \boldsymbol{X}_t^T Q_1 \boldsymbol{X}_t + \boldsymbol{u}_t^T R \boldsymbol{u}_t + \gamma \boldsymbol{Z}_{t+1}^T H^{j+1} \boldsymbol{Z}_{t+1}$;
  $K_1^{j+1} \leftarrow H_{uu}^{-1} H_{uX}$;
  $j \leftarrow j+1$
until $\left\| H^{j+1} - H^j \right\|_\infty < \eta$;

Algorithm 3.6: On-line (data-driven) value iteration (VI) for LQT based on Q-functions; based on [42, 44]

input : Dimension $n+m+p$ of the augmented state-action space, augmented state weight matrix Q_1, input weight matrix R, discount rate γ, threshold η
output: Near-optimal kernel matrix H, near-optimal control feedback matrix K_1

initialize K_1^0 as arbitrary stabilizing control feedback matrix;
$j \leftarrow 0$;
repeat
  solve $\boldsymbol{Z}_t^T H^{j+1} \boldsymbol{Z}_t = \boldsymbol{X}_t^T Q_1 \boldsymbol{X}_t + \boldsymbol{u}_t^T R \boldsymbol{u}_t + \gamma \boldsymbol{Z}_{t+1}^T H^{j} \boldsymbol{Z}_{t+1}$;
  $K_1^{j+1} \leftarrow H_{uu}^{-1} H_{uX}$;
  $j \leftarrow j+1$
until $\left\| H^{j+1} - H^j \right\|_\infty < \eta$;

[43, 44]. Without PE, these algorithms will not converge to the optimal kernel matrix. The exploration signal can be implemented as any input that disturbs the system, such as a sum of sinusoids. For aerospace systems, doublets or 3211s are more common for system identification purposes [47].

3.3. Optimal Tracking for Nonlinear Systems

Unfortunately, for general nonlinear systems, the useful simplifications that appear in the linear case do not apply. For this type of systems, the value function is not necessarily quadratic, and the optimal policy is commonly not a simple feedback control law that is linear in the (augmented) states. Therefore, one has to resort to more general, complex function approximations for the value function and policy. However, the main ideas of solving the HJB equation via iterative ADP methods still apply in this case. As illustrated in this section, these concepts can be implemented in a variety of ways.

Many approaches for solving the case of optimal nonlinear tracking control have been reported in literature (see e.g.,[41, 60, 81, 115, 116]). There are a few essential distinctions between the different lines of work. Broadly speaking, these are mostly related to the use of discounting and the dependency of the utility on the control input. In most of the optimal control literature, the standard formulation of the tracking problem as presented in Subsection 3.1.2 is adopted, as in e.g. [60, 115]. However, this calls for a control feedforward term that is obtained from dynamic inversion of the system dynamics. The alternative, discounted formulation of the tracking problem does not require this feedforward term, but does not lead to the 'true' optimal control as explained in Subsection 3.2.1. Nevertheless, this approach cannot do without discounting, as otherwise the value function would grow to infinity in the infinite-horizon tracking problem (recall Section 2.2). Alternatively, the input could be left out of the utility altogether, but this eliminates the possibility to penalize the controls. The latter is especially relevant in case the available control authority is bounded, as reported in e.g. [41].

Based on these insights, this section presents a family of parametric adaptive critic designs (ACDs, see also Subsection 2.5.2) for the case of optimal nonlinear tracking control. Both the alternative discounted

formulation as well as the case of input-independent utilities will be considered. Most of these ACD designs were initially proposed by Werbos [112], and can be categorized as Heuristic Dynamic Programming (HDP), Dual Heuristic Dynamic Programming (DHP), Globalized DHP (GDHP), and their so-called action-dependent versions ADHDP, ADDHP, and ADGDHP [81]. Of these, HDP will be presented first, to be followed by its action-dependent equivalent ADHDP. For the remaining variants, one can refer to e.g. [81, 89, 112] Additionally, recently proposed incremental alternatives will be briefly discussed, as presented in [116].

3.3.1. Heuristic Dynamic Programming

The first class of methods that will be discussed is formed by those that fall under the category of Heuristic Dynamic Programming (HDP). As these are based on the use of state value functions, they require a model for performing the policy improvement step. In [41], a method that is essentially HDP is presented for the nonlinear tracking problem based on the augmented state space formulation. Here, the nonlinear HJB equation for the augmented state value function is solved via iterative ADP, which has been illustrated multiple times before for the LQT problem. The algorithmic templates for solving the augmented tracking problem with HDP PI and VI are given in Algorithms 3.7 and 3.8, respectively. Note that the linear methods presented in sections 3.2.2 and 3.2.3 are in fact also examples of HDP.

In general, convergence of the iterative ADP method depends strongly on the functionals used for the actor and the critic. Selecting these functionals a-priori is often a non-trivial task. For this reason, multi-layer ANNs are often used as function approximators: these functionals are known for their high approximation capabilities, whereas they require only little memory because of their compact form. Feedforward ANNs are particularly popular in this context, but there is nothing that prevents one from using other forms such as recurrent neural networks (RNNs) [85]. In HDP, ANNs are used to model the critic, the actor, and the plant

Algorithm 3.7: Heuristic Dynamic Programming (HDP) Policy Iteration (PI) for discounted nonlinear optimal tracking control [41]

input : Input dynamics $g(\boldsymbol{x})$, dimension p of the command generator dynamics, augmented state weight matrix Q_1, input weight matrix R, discount rate γ, threshold η
output: Near-optimal value function $V_*(\boldsymbol{X}) \approx \hat{V}_*(\boldsymbol{X}) \leftarrow \hat{V}^{(j+1)}(\boldsymbol{X})$, near-optimal policy $\pi_*(\boldsymbol{X}) \approx \hat{\pi}_*(\boldsymbol{X}) \leftarrow \hat{\pi}_{j+1}(\boldsymbol{X})\ \forall \boldsymbol{X} \in \mathscr{X}'$

initialize $\hat{\pi}_0(\boldsymbol{X})$ as arbitrary stabilizing control policy $\forall \boldsymbol{X} \in \mathscr{X}'$;
$j \leftarrow 0$;
repeat
  solve $\hat{V}^{(j+1)}(\boldsymbol{X}_t) = r(\boldsymbol{X}_t, \boldsymbol{u}_t) + \gamma \hat{V}^{(j+1)}(\boldsymbol{X}_{t+1})$;
  $\hat{\pi}_{j+1}(\boldsymbol{X}_t) \leftarrow -\frac{1}{2}\gamma R^{-1} G^T(\boldsymbol{X}_t) \frac{\partial \hat{V}^{(j+1)}(\boldsymbol{X}_{t+1})}{\partial \boldsymbol{X}_{t+1}}$;
  $j \leftarrow j+1$
until $\left\| \hat{V}^{(j+1)}(\boldsymbol{X}) - \hat{V}^{(j)}(\boldsymbol{X}) \right\|_\infty < \eta$;

Algorithm 3.8: Heuristic Dynamic Programming (HDP) Value Iteration (VI) for discounted nonlinear optimal tracking control; based on [41]

input : Input dynamics $g(\boldsymbol{x})$, dimension p of the command generator dynamics, augmented state weight matrix Q_1, input weight matrix R, discount rate γ, threshold η
output: Near-optimal value function $V_*(\boldsymbol{X}) \approx \hat{V}_*(\boldsymbol{X}) \leftarrow \hat{V}^{(j+1)}(\boldsymbol{X})$, near-optimal policy $\pi_*(\boldsymbol{X}) \approx \hat{\pi}_*(\boldsymbol{X}) \leftarrow \hat{\pi}_{j+1}(\boldsymbol{X})\ \forall \boldsymbol{X} \in \mathscr{X}'$

initialize $\hat{\pi}_0(\boldsymbol{X})$ as arbitrary control policy $\forall \boldsymbol{X} \in \mathscr{X}'$;
$j \leftarrow 0$;
repeat
  solve $\hat{V}^{(j+1)}(\boldsymbol{X}_t) = r(\boldsymbol{X}_t, \boldsymbol{u}_t) + \gamma \hat{V}^{(j)}(\boldsymbol{X}_{t+1})$;
  $\hat{\pi}_{j+1}(\boldsymbol{X}_t) \leftarrow -\frac{1}{2}\gamma R^{-1} G^T(\boldsymbol{X}_t) \frac{\partial \hat{V}^{(j+1)}(\boldsymbol{X}_{t+1})}{\partial \boldsymbol{X}_{t+1}}$;
  $j \leftarrow j+1$
until $\left\| \hat{V}^{(j+1)}(\boldsymbol{X}) - \hat{V}^{(j)}(\boldsymbol{X}) \right\|_\infty < \eta$;

model [112]. For the critic network, the quadratic TD-error is often taken as the loss function, i.e. in the case of HDP PI:

$$E_c(t) = \tfrac{1}{2} e_c(t)^2 \tag{3.10a}$$

with the TD error similar as before, i.e. for PI:

$$e_c(t) = r(\boldsymbol{X}_t, \boldsymbol{u}_t) + \gamma \hat{V}^{(j+1)}(\boldsymbol{X}_{t+1}) - \hat{V}^{(j+1)}(\boldsymbol{X}_t) \tag{3.10b}$$

However, the backward-view TD-error may also be used in this context, as in [41, 88]. This loss is used to update the weights for every neuron via SGD, a process also known as backpropagation [85, 98]. Writing $\hat{V}^\bullet(t) = \hat{V}^\bullet(\boldsymbol{X}_t)$, the following weight update rule applies [41, 88, 108]:

$$\boldsymbol{w}_c(t+1) = \boldsymbol{w}_c(t) - \alpha_c(t) \frac{\partial E_c(t)}{\partial \boldsymbol{w}_c(t)} \tag{3.10c}$$

$$\frac{\partial E_c(t)}{\partial \boldsymbol{w}_c(t)} = \frac{\partial E_c(t)}{\partial e_c(t)} \frac{\partial e_c(t)}{\partial \hat{V}^{(j+1)}(t)} \frac{\partial \hat{V}^{(j+1)}(t)}{\partial \boldsymbol{w}_c(t)} \tag{3.10d}$$

In principle, it is possible to derive the optimal control directly from the critic network by acting greedily on its output (see Algorithms 3.7 and 3.8). However, due the dependency of HDP on the state value function, this requires full knowledge of the system dynamics. This was already seen in Subsection 3.2.3 for LQT [89]. However, knowledge about the internal and command generator dynamics can be circumvented by using a separate actor network that approximates the optimal control law [58]. An example of this is reported in [41] for optimal tracking with input-dependent utilities. They define the following loss function for the actor update law:

$$E_a(t) = \tfrac{1}{2} e_a(t)^2 \tag{3.11a}$$

$$e_a(t) = \tilde{\boldsymbol{u}}^{(j+1)}(\boldsymbol{X}_{t-1}) - \hat{\boldsymbol{u}}^{(t-1)}(\boldsymbol{X}_{t-1}) \tag{3.11b}$$

where the superscript $t-1$ for the actor input $\hat{\boldsymbol{u}}$ is used to indicate that the current prediction of the optimal control is based on the weights calculated in the previous time step, and $\tilde{\boldsymbol{u}}^{(j+1)}(\boldsymbol{X}_{t-1})$ represents the greedy optimal control input obtained from $\hat{\pi}_{j+1}(\boldsymbol{X}_{t-1})$ as in Algorithms 3.7 and 3.8. Consequently, the actor weight update rule is defined as [41, 45]:

$$\boldsymbol{w}_a(t+1) = \boldsymbol{w}_a(t) - \alpha_a(t) \frac{\partial E_a(t)}{\partial \boldsymbol{w}_a(t)} \tag{3.11c}$$

$$\frac{\partial E_a(t)}{\partial \boldsymbol{w}_a(t)} = \frac{\partial E_a(t)}{\partial e_a(t)} \frac{\partial e_a(t)}{\partial \hat{\boldsymbol{u}}^{(t-1)}(t)} \frac{\partial \hat{\boldsymbol{u}}^{(t-1)}(t)}{\partial \boldsymbol{w}_a(t)} \tag{3.11d}$$

where the notation $\hat{\boldsymbol{u}}^\bullet(t) = \hat{\boldsymbol{u}}^\bullet(\boldsymbol{X}_t)$ is used again. Note that knowledge of the input dynamics is still required to calculate $\tilde{\boldsymbol{u}}^{(j+1)}(\boldsymbol{X}_{t-1})$. The policy improvement and evaluation steps can be performed sequentially, synchronously or something in-between, according to the governing principles of GPI [41, 89, 108]. In the fully synchronous case, it can be seen that $j+1 = t$, which makes HDP similar to the RL methods presented in Section 2.5 such as approximate SARSA and Q-learning (i.e. on-policy and off-policy synchronous HDP, respectively). Additionally, it must be mentioned here that the HDP training method will only converge in case persistence of excitation (PE) is ensured [81].

Alternatively, HDP is often used to solve the optimal tracking problem where the utility is not dependent on the input. This has a considerable impact on the formulation of the problem, which does not require augmentation of the system state space anymore to ensure admissibility of the optimal control policy. In this case, the TD error is simply defined in terms of the system state:

$$e_c(t) = r(\boldsymbol{x}_t) + \gamma \hat{V}^{(j+1)}(\boldsymbol{x}_{t+1}) - \hat{V}^{(j+1)}(\boldsymbol{x}_t) \tag{3.12a}$$

With the loss function for the critic network defined similarly as in Equation 3.10a, the critic weight update law is again equivalent to Equations 3.10c and 3.10d. The update law for the actor network is defined quite differently, as there is not an explicit expression for the optimal control input in this case. However, what is known for this case is the output of the optimal value function, which equals 0 in case of perfect tracking. Therefore, the loss function can be defined as [88, 108]:

$$E_a(t) = \frac{1}{2} e_a(t)^2 \tag{3.13a}$$

$$e_a(t) = \hat{V}^{(j+1)}(\boldsymbol{x}_t) - V_*(\boldsymbol{x}_t) \tag{3.13b}$$

with $V_*(\boldsymbol{x}_t) = 0 \; \forall \boldsymbol{x} \in \mathscr{X}$. However, the fact that HDP is based on state value functions implies that the backpropagation path for the actor update law goes through the model network (recall Subsection 2.5.3). This model network can be learned a-piori, i.e. fully off-line, or it can be identified online. For the latter, a distinction can be made between learning a global model, as in [108], or an incremental model, as discussed in Subsection 3.3.3. For now, a fixed model network is assumed. Then, for optimal tracking with input-dependent utilities, the following update law applies [108]:

$$\boldsymbol{w}_a(t+1) = \boldsymbol{w}_a(t) - \alpha_a(t) \frac{\partial E_a(t+1)}{\partial \boldsymbol{w}_a(t)} \tag{3.13c}$$

$$\frac{\partial E_a(t+1)}{\partial \boldsymbol{w}_a(t)} = \frac{\partial E_a(t+1)}{\partial e_a(t+1)} \frac{\partial e_a(t+1)}{\partial \hat{V}^{(j+1)}(t+1)} \frac{\partial \hat{V}^{(j+1)}(t+1)}{\partial \boldsymbol{x}(t+1)} \frac{\partial \boldsymbol{x}(t+1)}{\partial \hat{\boldsymbol{u}}^{(t-1)}(t)} \frac{\partial \hat{\boldsymbol{u}}^{(t-1)}(t)}{\partial \boldsymbol{w}_a(t)} \tag{3.13d}$$

Note that similar as in the case of input-dependent utilities, one can delay the actor update by a single time step to eliminate the need for predicting the next state through the model (i.e., as in [88]). This implies that only the input dynamics need to be modeled.

3.3.2. Action-Dependent Heuristic Dynamic Programming

The basic ACDs presented in the previous section are all based on the state value function. It was seen multiple times before that using state-action values instead reduces or even eliminates the dependency of RL/ADP methods on the availability of a model. This idea is exploited in so-called action-dependent ACD techniques, which assume a direct connection between the actor and the critic networks [81]. For action-dependent HDP (ADHDP), this implies that the update law for the actor network is not dependent on backpropagation through a model network anymore. ADHDP is in fact equivalent to Q-learning with function approximation of the critic and the actor [81]. Training of the critic is similar for ADHDP as explained earlier for HDP, and consists of minimizing the squared TD error (Equation 3.12a) through stochastic gradient descent:

$$e_c(t) = r(\boldsymbol{x}_t) + \gamma \hat{Q}^{(j+1)}(\boldsymbol{x}_{t+1}, \boldsymbol{u}_{t+1}) - \hat{Q}^{(j+1)}(\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{3.14a}$$

$$\boldsymbol{w}_c(t+1) = \boldsymbol{w}_c(t) - \alpha_c(t) \frac{\partial E_c(t)}{\partial \boldsymbol{w}_c(t)} \tag{3.14b}$$

$$\frac{\partial E_c(t)}{\partial \boldsymbol{w}_c(t)} = \frac{\partial E_c(t)}{\partial e_c(t)} \frac{\partial e_c(t)}{\partial \hat{Q}^{(j+1)}(t)} \frac{\partial \hat{Q}^{(j+1)}(t)}{\partial \boldsymbol{w}_c(t)} \tag{3.14c}$$

The update law for the actor, which still consists of Equations 3.13a-3.13c, is much simpler compared to HDP however, as a result of the shortened backpropagation path [81, 88, 108]:

$$\frac{\partial E_a(t+1)}{\partial \boldsymbol{w}_a(t)} = \frac{\partial E_a(t+1)}{\partial e_a(t+1)} \frac{\partial e_a(t+1)}{\partial \hat{Q}^{(j+1)}(t+1)} \frac{\partial \hat{Q}^{(j+1)}(t+1)}{\partial \hat{\boldsymbol{u}}^{(t-1)}(t)} \frac{\partial \hat{\boldsymbol{u}}^{(t-1)}(t)}{\partial \boldsymbol{w}_a(t)} \tag{3.15}$$

Similarly as for HDP, in case the utility is not independent of the input signal, AHDHP needs to be adopted in combination with an augmented description of the state space. This implies that a different actor update law is needed again, since $V_*(\boldsymbol{x}_t) = \min_{\mathscr{U}} Q_*(\boldsymbol{x}_t, \boldsymbol{u}) \neq 0 \; \forall \boldsymbol{x} \in \mathscr{X}$ in this case. Thanks to the definition of the Q-function however, calculation of the optimal tracking control law can again be done in a completely model-free fashion. This in contrast to HDP, for which full knowledge of the augmented input dynamics was required in the policy improvement step. A critic-only approach to model-free optimal tracking control is proposed in [63, 64], where the optimal control input is directly derived from the converged critic network by requiring that $\frac{\partial \hat{Q}_*(\boldsymbol{X}_{t+1}, \hat{\boldsymbol{u}}_t)}{\partial \hat{\boldsymbol{u}}_t} = 0$ and applying SGD:

$$\hat{\boldsymbol{u}}(t+1) = \hat{\boldsymbol{u}}(t) - \alpha_a(t) \frac{\partial \hat{Q}_*(t+1)}{\partial \hat{\boldsymbol{u}}(t)} \tag{3.16}$$

Here, no superscript has been used for the input vector to reflect the fact that no separate actor network is used. Note that the ADHDP formulation requires a more involved training procedure, due to the fact the Q-function needs to approximate both the value function and the input dynamics model at the same time

[108]. Moreover, it has to do this with less information compared to HDP, as the backpropagation path is less 'precise' in the gradients. Additionally, it must be mentioned that ADHDP suffers in particular from problems with persistence of excitation (PE) [112]. Note that the LQT Q-learning approach described in Subsection 3.2.4 is in fact an example of ADHDP.

3.3.3. Incremental Approaches

The HDP and ADHDP frameworks discussed in the preceding subsections each have distinct advantages and disadvantages: whereas HDP generally enjoys better convergence properties over its action-dependent equivalent, the latter requires considerably less a-priori knowledge since the policy evaluation and improvement steps can be done in a completely model-free fashion. It appears that a balance between these characteristics can be found if an external model network is learned in parallel to the actor and the critic [30, 108]. In this way, an a-priori model of the dynamics will not be required, whereas the complexity of the critic network is considerably reduced and a better estimate of the gradient used in the actor update step will be available by virtue of $\frac{\partial \boldsymbol{x}_{t+1}}{\partial \hat{\boldsymbol{u}}_t}$. This taps into the fundamental advantages of model-based reinforcement learning, as discussed in Subsection 2.5.3. However, learning a global model of the system dynamics is often a non-trivial task, especially when the dynamics are highly nonlinear. In [116, 117, 119], a new framework known as Incremental model-based Heuristic Dynamic Programming (IHDP) is proposed that circumvents this problem by learning an incremental model instead. Here, it is assumed that for sufficiently high sample rates, the global model can replaced by a local approximation that is linear around a given state $\boldsymbol{x}_0$. These locally linear representations are not stored for every operating point; instead, a single time-varying linear model is used, which is updated in parallel with the actor and the critic in a completely online fashion.

An incremental form of the system dynamics can be readily obtained by applying the first-order Taylor expansion. In [116, 117, 119], it is demonstrated how this can be done for discrete-time dynamic systems. Their findings will be briefly repeated here. In general, a non-affine discrete-time dynamical system can be described as follows:

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{3.17a}$$

Note that this formulation transcends the more limited class of input-affine dynamics considered earlier in Equation 3.1b. Consequently, the system can be linearized around a given state $\boldsymbol{x}_0$:

$$\boldsymbol{x}_{t+1} \simeq f(\boldsymbol{x}_0, \boldsymbol{u}_0) + \left.\frac{\partial f(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_0, \boldsymbol{u}_0} (\boldsymbol{x}_t - \boldsymbol{x}_0) + \left.\frac{\partial f(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{u}}\right|_{\boldsymbol{x}_0, \boldsymbol{u}_0} (\boldsymbol{u}_t - \boldsymbol{u}_0) \tag{3.17b}$$

Consequently, if Δt is sufficiently small, the linearization point $\boldsymbol{x}_0$ can be directly replaced by the state at time $t-1$; recognizing that $\boldsymbol{x}_t = f(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1})$, the following is obtained:

$$\boldsymbol{x}_{t+1} - \boldsymbol{x}_t \simeq \left.\frac{\partial f(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}} (\boldsymbol{x}_t - \boldsymbol{x}_{t-1}) + \left.\frac{\partial f(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{u}}\right|_{\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}} (\boldsymbol{u}_t - \boldsymbol{u}_{t-1}) \tag{3.17c}$$

Which can be rewritten in the following concise form:

$$\Delta \boldsymbol{x}_{t+1} \simeq F(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}) \Delta \boldsymbol{x}_t + G(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}) \Delta \boldsymbol{u}_t \tag{3.17d}$$

Here, $F(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1})$ and $G(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1})$ are known as the state transition and control effectiveness matrix, respectively. These matrices can be identified online from collected sensor measurements using least-squares parameter estimation methods. Consequently, the control effectiveness matrix can be directly adopted for the actor update step, since $G(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}) \simeq \frac{\partial \boldsymbol{x}_{t+1}}{\partial \hat{\boldsymbol{u}}_t}$. In [117], the IHDP approach is demonstrated in an online optimal tracking task for a representative air vehicle model, which illustrates its applicability for online learning tasks. However, care must be taken to ensure persistent excitation of the system.

Finally, this section started with the statement that for general nonlinear systems, the simplifications that appeared to be very useful for the linear case do not apply. Although this is true indeed, it turns out that there is an exception for nonlinear systems for which the value function is approximately quadratic in the tracking error. By adopting an incremental model-based approach, the possibility to use the kernel matrix P is restored, which makes solving the HJB equation particularly simple even for nonlinear systems. This idea is adopted in the incremental Approximate Dynamic Programming (iADP) framework, which was developed by [116, 118]. In their work, the iADP approach is applied to a spacecraft with liquid sloshing, a highly nonlinear dynamic phenomenon which is generally very hard to model. This demonstrates the potential of this method.

4

Curriculum Learning

Reinforcement learning algorithms tend to consume considerable amounts of data before reaching near-optimality of the policy. This becomes problematic when this data is expensive to obtain, e.g. because of long running times, delicate hardware, or high operational (monetary) costs. The ability of the RL algorithm to find the optimal policy for a given MDP is inherently dependent on the complexity of the task at hand. Task complexity covers multiple aspects of a task such as dimensionality of the state and action space or richness of the reward signal. However, a key insight is that this dependency does not exclusively define learning performance, but that there is a strong link with the abilities of the agent itself. This understanding gives rise to the idea of adjusting the complexity of a given learning task to the agent's competencies, in order for it to learn the desired behavior more effectively. To a certain extent, this can be seen as the reverse counterpart of finding a good agent representation for a certain MDP, as discussed in Subsection 2.5.2. This notion of solving a given task by breaking down its complexity in a sequence of auxiliary learning tasks is known as *curriculum learning* (CurL).

The idea of presenting a learning entity with a meaningful learning curriculum has been frequently exploited in animal training studies, where it also known as *shaping* [10, 91]. It also forms the basis of virtually every human education system, where students learn to master 'simpler' concepts first before they get presented with more complex material. This allows them to exploit the knowledge they already have to grasp new concepts more effectively. The idea to use this strategy in supervised learning[1] with deep neural networks was first formally introduced in [10]. Here, the authors successfully demonstrated effective learning curricula for shape recognition and language modeling. Curriculum learning in reinforcement learning was conceived independently in a completely different branch of research, known as *transfer learning* (TL), and was first introduced in [100]. Here, the idea is that knowledge learned in one MDP (the source) can be reused in a different, but related MDP (the target). This concept can be extended to a well-designed learning sequence, where at each stage knowledge is transferred between at least one pair of tasks.

In this chapter, curriculum learning is introduced as a new dimension to RL as direct adaptive optimal control. First, the fundamentals and taxonomy as derived from literature are presented in Section 4.1. Based on the concepts introduced here, Section 4.2 presents a few approaches reported in literature that are illustrative for how intermediate training distributions can be managed in terms of learning complexity. Finally, Section 4.3 gives a brief introduction to the field of transfer learning.

4.1. Fundamentals

The concept of curriculum learning is based on the idea that for non-convex optimization tasks, the order in which training samples are presented is of vital importance to avoid bad local minima [10, 51]. In practice, finding the global optimum in such a problem is often facilitated by randomly initializing a learning algorithm over multiple runs, after which the best performing solution is selected based on its performance on a validation set [51]. However, such an approach requires a vast number of seeds if the function image contains multiple local minima over its domain, which is computationally expensive. Curriculum learning provides an efficient alternative to this practice, by virtue of its commonality with continuation methods [10]. Within this

[1] However, the principle idea of presenting the agent with a sequence of auxiliary control tasks before solving a required end task was already coined much earlier in the field of reinforcement learning in [87].

class of methods, the non-convex optimization criterion is smoothened by a series of lower-order criteria, where in each stage of the resulting training sequence the solution is drawn towards a region that is in the vicinity of the global optimum. In a way, curriculum learning is based on a similar conception, but relies on an appropriate sequence of training samples rather than the optimization objective. Or, as Bengio et al. put it, on a "reweighing of the training distribution" [10]. In this regard, a curriculum can be formed by multiple stages where in each stage samples of a certain complexity level are presented to the learner according to some probability.

The ideas behind curriculum learning for supervised learning are formalized in [10]. This framework is repeated here, but has been adapted in accordance with the reinforcement learning notation. Consider a certain transition tuple $z_{t+1} : (\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}, r_{t+1})$ corresponding to a given MDP $M : \langle \mathscr{X}, \mathscr{U}, \tilde{F}, \tilde{\rho}, \gamma \rangle$; then, let $\mathscr{D}(z_{t+1})$ be the target distribution determining the collection of samples that the agent will be presented with ultimately. Similarly, a weight $0 \leq W_\lambda(z_{t+1}) \leq 1$ can be defined that adjusts the target distribution $\mathscr{D}(z_{t+1})$, with λ representing the step in the training sequence. Then, at each step λ, an intermediate training distribution is formed as:

$$\mathscr{D}'_\lambda(z_{t+1}) \propto W_\lambda(z_{t+1})\mathscr{D}(z_{t+1}) \;\; \forall z_{t+1} : t \in [0, T] \tag{4.1}$$

with T the total number of transitions in the target distribution. Bengio et al. then summarize the definition of curriculum learning based on an increasing *entropy of distributions*, as repeated in Definition 4.1.

Definition 4.1: Curriculum Learning with Fixed Internal Representation; adapted from [10]

A learning curriculum is formed by a series of intermediate training distributions $\mathscr{D}'_\lambda(z_{t+1})$, where at each stage the entropy of this distribution increases according to

$$H(\mathscr{D}'_{\lambda+\epsilon}) > H(\mathscr{D}'_\lambda) \;\; \forall \epsilon > 0 \tag{4.2}$$

and where the weighting function $W_\lambda(z_{t+1})$ is monotonically increasing, i.e.

$$W_{\lambda+\epsilon}(z_{t+1}) \geq W_\lambda(z_{t+1}) \;\; \forall z_{t+1} : t \in [0, T], \forall \epsilon > 0 \tag{4.3}$$

In the context of RL, the entropy of the training distribution can be increased by adding new (unseen) transition samples to the training set, or by substituting 'easy' transitions by others of higher complexity. Then, by increasing the weighting function accordingly, the probability that this sample is used for training is increased. Considering the continuation analogy, the knowledge gained in each stage λ draws the parameter set describing the agent's internal representation towards the productive region for the next stage $\lambda+\epsilon$. In this view, the intermediate training distributions can be derived from a single MDP M only, or can be modeled as intermediate MDPs themselves. This adds another perspective to the definitions of the distribution entropy $H(\mathscr{D}'_\lambda)$ and weighting function $W_\lambda(z_{t+1})$, as they effectively *project* the target MDP M to an auxiliary MDP M'_λ of lower complexity.

As mentioned before, curriculum learning has also been conceived in the field of transfer learning where it is defined as a sequence of training tasks that allow an agent to learn a different, but related task M more effectively by leveraging knowledge gained from another task M'. Here, the aim is to improve learning speed or performance on a single complex task [74, 100] by making use of TL techniques[2]. This conception extends the motivation for curriculum learning, as it does not merely consider the case of finding the global minimum in a non-convex optimization task, but adds the dimension of finding the optimum *more quickly*. This has two implications. First, curriculum learning can also be beneficial for convex learning tasks in case fast learning is desired; and second, it allows the agent's internal representation to change at each stage λ. Hence, the continuation aspect of a learning curriculum is not limited to the training distribution only, but also resides in its representation by the agent. For example, whereas performing optimally in a complex task requires a representation with considerable approximation power, it can be beneficial to draw it towards the productive parts of the parameter space based on earlier knowledge captured by a lower-level representation. This means that a distinction can be made between the *actor* or *agent domain* and the *task* or *training domain*[3]. As a result, Definition 4.1 can be extended by relaxing the condition on strictly increasing entropy at

[2] A discussion of transfer learning is postponed until Section 4.3

[3] Definitions for these terms were presented in Subsection 1.3.2.

Definition 4.2: Curriculum Learning with Variable Internal Representation; based on [10]

A learning curriculum is formed by a series of intermediate training distributions $\mathscr{D}'_\lambda(z_{t+1})$ and internal representations. At each stage of the curriculum, the entropy of the training distribution does not decrease, i.e.

$$H(\mathscr{D}'_{\lambda+\epsilon}) \geq H(\mathscr{D}'_\lambda) \;\; \forall \epsilon > 0 \tag{4.4}$$

while the weighting function $W_\lambda(z_{t+1})$ is monotonically increasing, i.e.

$$W_{\lambda+\epsilon}(z_{t+1}) \geq W_\lambda z_{t+1}) \;\; \forall z_{t+1} : t \in [0,T], \forall \epsilon > 0 \tag{4.5}$$

and the internal representation *can* be adjusted to reduce the approximation residual, according to

$$\Delta_{\lambda+\epsilon}(\mathscr{D}'_\lambda) \leq \Delta_\lambda(\mathscr{D}'_\lambda) \tag{4.6}$$

every stage and adding the option to adopt a new representation resulting in a lower approximation residual Δ (e.g., network loss). This results in Definition 4.2, which now also incorporates a related paradigm known as *representation learning*[4] [101].

4.1.1. Learning Complexity in Reinforcement Learning

A complete formulation of the curriculum learning paradigm requires an accurate account of learning complexity. Although the term is frequently coined in literature, an adequate description is often not given. Advanced classifications of complex systems exist (e.g., as in [107]), but these are outside the scope of the current discussion. Therefore, an attempt is made here to summarize those aspects of an MDP that make finding the optimal solution difficult. First, the same distinction between the task domain and the agent domain is made again. This makes learning complexity the combination of task complexity and agent complexity. It is important to recognize that the complexity of the agent domain is often related to the complexity of the task domain, but not always. This depends on the aspects of the task that are seen to be complex. Assuming that the learning task is an MDP $M : \langle \mathscr{X}, \mathscr{U}, \tilde{F}, \tilde{\rho}, \gamma \rangle$ that is well-observable, task complexity can arise in three main areas: (1) the dimensionality of the state space $\mathscr{X} \subset \mathbb{R}^n$ and/or action space $\mathscr{U} \subset \mathbb{R}^m$; (2) the nature of the dynamics $\tilde{F}$; and (3) the richness of the reward function $\tilde{\rho}$. Considering the dimensionality of the task, a high-dimensional state space and and/or action space requires a large number of samples and therefore significant exploration. This dimensionality must also be captured by the internal representation of the agent, often resulting in a high-dimensional function approximation for the value function and/or policy. This is related to the curse of dimensionality (see Subsection 2.5.2). Second, the nature of the dynamics plays a less trivial role. Complexity often arises in terms of strongly coupled dynamics, nonlinear dependencies, discontinuities, instabilities, badly or non-controllable dynamics, and/or strong stochastic effects. The first three aspects call for complex function approximators such as ANNs which have sufficient approximation power to enable adequate representation of the optimal value function and/or policy. Nonlinearities make the optimization problem non-convex. The combined problem of learning complexity is enlarged if the internal representation is inadequate. For example, a smooth representation is not capable of capturing the optimal value function if the dynamics contain significant discontinuities spread over the domain. By contrast, the impact of stochastic effects is unrelated to complexity of the agent domain, but requires more interactions with the system according to the law of larger numbers. The effect of instabilities is also not related to representation complexity, but has a strong impact on the time available to learn a stable policy in an online setting before reaching a fatal region of the state space[5]. Finally, a sparse reward function can lead to considerable difficulties in discovering the desired behavior (see e.g. [76] for an elaborate treatment of this topic). A well-designed reward signal may speed up the learning process considerably, effectively reducing the complexity of learning the task. However, this is generally unrelated to the agent domain.

Based on this account of learning complexity, increasing the entropy of the distribution set during every stage λ of the learning curriculum then refers to a complexification or diversification of learning samples based on heuristics that capture any of the three main areas responsible for task complexity. However, it is

[4] One may also recognize elements from *active learning* in this framework (see e.g. [106]). However, active learning does generally not account for learning based on multiple levels of complexity, and will therefore not be considered here.

[5] A more elaborate discussion of fatal transitions and the fatal state space is provided in Subsection 5.2.1.

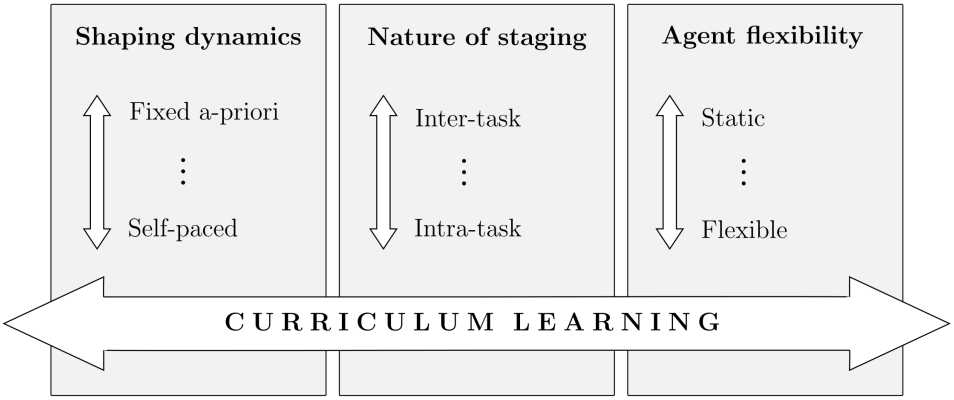


Figure 4.1: Visual representation of the curriculum learning taxonomy

often unclear how such a heuristic can be obtained in the first place. This is related to the general taxonomy seen in curriculum learning, which will be subject of the next section.

4.1.2. Taxonomy

The framework introduced in the previous subsection provides for a structured taxonomy of techniques that have emerged in the loosely organized field of curriculum learning for RL. In broad terms, the curriculum learning paradigm can be decomposed along three dimensions. The first element is referred to as the *shaping dynamics*. In general, a learning curriculum can be fully determined a-priori, or may arise from online feedback of the capabilities of the agent during the learning process. A combination of these is also possible in some hybrid form. Curriculum learning of the first form is based on a fixed sequence of training distributions selected heuristically by some oracle, commonly a domain expert. An illustrative example is presented in [10], where a nonlinear classifier is trained on a sequence of images of shapes such as triangles, ellipses and rectangles. A learning curriculum is established that presents the classifier first with images of shapes that have high contrast and little variability, i.e. with straight orientations, before commencing with more complex examples. This ought to be a very intuitive way of curriculum design and allows a domain expert to embed prior knowledge in the learning process. However, such a ranking of training distributions can be very difficult to provide a-priori if it is not clear what determines the complexity in learning. Moreover, it does not take into account any feedback about the learning progress, meaning that the curriculum is static. The philosophy of generating a learning curriculum dynamically by the learner itself [51] or some agent-specific teacher [74] is known as *self-paced* learning (SPL). Here, the learner selects training samples autonomously based on a heuristic that indicates the complexity of learning a sample, e.g. the prediction loss [51] or temporal difference error [82]. However, like static curriculum learning, SPL suffers from some problems that need to be adequately addressed. In its basic form, SPL does not provide for any possibility to embed prior knowledge in the learning process. A hybrid method that allows for prior knowledge from static curriculum learning while also taking account of the advantages of SPL is known as self-paced curriculum learning [37]. Moreover, SPL does not take into account the diversity of the training samples, whereas this can be of vital importance to develop a comprehensive solution. This has given rise to the concept of self-paced learning with diversity (SPLD) [36]. The concept of SPL is discussed in more detail in Subsection 4.2.2.

The second dimension that defines the curriculum learning framework is the *nature of staging*. The main categories considered here will be denominated as *inter-task* and *intra-task* learning, referring to the notion of learning within or across (a set of) MDP(s). In intra-task learning, the environment or world associated with the target task are fully comprised within the boundaries of the MDP. That is, the boundaries and the inherent nature of the MDP will not change, meaning that the task domain is static. In the context of dynamical systems, this can be considered as an MDP that captures the complete system dynamics. It must be mentioned that varying levels of augmentation by some external controller, such as an autopilot or human collaborator, do not affect this definition[6]. This contrasts with inter-task staging, where either the boundary of the world or its inherent nature can be altered. This results in a new, distinct MDP, known as a source task.

[6] Although this controller does in fact affect the dynamic nature of the task and therefore the MDP that needs to be solved, the definition only refers to the boundary and the *inherent* (i.e. open-loop) nature of the dynamics.

In this case, one must adopt transfer learning techniques to carry knowledge from the source to the target MDP. Here, some heuristic is again required to regulate the timing of transfer. For example, this heuristic is based on some performance measure on the source task. The nature of staging directly follows from the nature of the intermediate training distributions, as formalized by Definitions 4.1 and 4.2.

The final dimension in curriculum learning is formed by the flexibility of the agent domain, in accordance with Definition 4.2. In broad terms, one can make a distinction between static and dynamic (or flexible) agent representations. With dynamic agent representations, one can opt to 1) switch between different representations altogether or 2) apply a mapping between function approximations. With option 1), one essentially performs representation learning [101]. In this context, functions approximators can be replaced during the learning curriculum by counterparts with higher representational power (complexificiation). An even more radical approach is also possible, by completely changing the learning algorithm (i.e. from learning based on value functions to policy search). Option 2) counts on the availability of a mapping across functions, i.e. those related to the value function, the policy, or an internal model. Subsection 4.3.3 gives a small introduction to transferring knowledge using mappings. Note that this is largely equivalent to the concept of *flexible function approximation* developed by [35].

The proposed taxonomy is visualized in Figure 4.1. Note that in general, a learning curriculum can be employed for different purposes. Here, the case of learning a single complex target MDP is considered. However, it can also serve in the context of better generalization capabilities, also known as *multi-task* learning [105].

4.2. Training Distribution Entropy Management

In this section, various techniques for ranking the entropy of intermediate training distributions as reported in literature will be discussed. Based on the taxonomy presented in the previous section, this relates primarily to the shaping dynamics. Hand-engineered learning curricula will be considered first, to be followed by a more extensive examination of self-paced learning (SPL).

4.2.1. Ordering through Domain Knowledge

As mentioned before, the dependency of the curriculum learning paradigm on a properly defined measure of learning complexity gives the impression that the efficacy of the method is quite problem-specific. Nevertheless, in cases where the complexity of training samples can be established in a clear-cut manner, such an *ad hoc* approach to curriculum learning can leave impressive results. In supervised learning, there are many successful examples of this. In [17] for instance, a deep neural network for automatic speech recognition is trained by using a curriculum learning strategy where the signal-to-noise ratio (SNR) is the leading complexity factor. At the onset of training, the training distribution consists almost solely of samples with (very) low SNR values. As learning proceeds, samples with higher SNRs are added progressively, thereby increasing the entropy of the training distribution. The result is that the so-called word error rate (WER) decreases considerably compared to conventional training methods. Here, it is very clear how the SNR can be use as a guide to constructing effective learning curricula.

In reinforcement learning, similar results have been presented in a recent publication on learning complex locomotion behaviors in challenging environments through simple reward signals [33]. Here, agents with various embodiments are trained with deep reinforcement learning methods to traverse as long as possible distances on various terrains types. Each terrain has its own distinct characteristics that require the agent to adopt certain behaviors. For example, the terrain may be filled with slalom walls, hurdles, and platforms, which means that the only way the agent can make progress is to adopt specific gaits such as jumping and crouching. Accordingly, the entropy of the training distribution resides in the complexity (and diversity) of the terrain type, which can be specified heuristically in terms of distance between obstacles, steepness, and homogeneity. By increasing the complexity of the terrain gradually over the course of learning, a curriculum is formed that results in considerably better learning performance. Not only is the agent able to adopt the necessary gaits more quickly, the learned behavior also appears to be more robust to changing dynamics of the environment. For this problem, it can be easily seen how the training distribution can be adjusted appropriately to match the intermediate capabilities of the learning agent.

4.2.2. Autonomous Complexity Indexing

In learning tasks where it is not clear how an appropriate learning curriculum can be designed a-priori, one will have to adopt self-paced learning (SPL) techniques that rank training samples directly from the data collected by the agent. In SPL, curriculum design is tied directly to the learning objective by adding a con-

vex regularization term to the loss function[7] L [36, 37, 51]. That is, for a given supervised learning method, the learning objective is augmented by a separate term that controls the importance of each sample in the training set. Most commonly, the regularization term is taken as the ℓ_1-norm, which leads to the following optimization objective for SPL [36, 37, 51]:

$$\min_{\boldsymbol{w}\in\mathbb{R}^d,\boldsymbol{v}\in[0,1]^n}\left[\sum_{i=1}^{n} v_i L(y_i,[F(\boldsymbol{w})](\boldsymbol{x}_i))-\kappa\sum_{i=1}^{n} v_i\right] \tag{4.7a}$$

with $[F(\boldsymbol{w})](\boldsymbol{x})$ defined as some functional that estimates the true output y_i over the complete training distribution $\mathscr{D}=\{(\boldsymbol{x}_i,y_i)\}_{i=1}^n$, $\boldsymbol{v}=\begin{bmatrix}v_1 & v_2 & \dots & v_n\end{bmatrix}^T$ a binary weight vector that is similar in nature as the weighting function introduced in Definition 4.1, and κ a parameter that quantifies learning complexity. Optimizing according to this objective in one go is not possible. A feasible alternative is to make the optimization of $\boldsymbol{w}$ and $\boldsymbol{v}$ disjoint; then, for fixed $\boldsymbol{w}$, one can optimize $\boldsymbol{v}$ straightforwardly by applying the following condition [36, 37, 51]:

$$v_i^*=\begin{cases}1, & L(y_i,[F(\boldsymbol{w})](\boldsymbol{x}_i))<\kappa\\ 0, & \text{otherwise.}\end{cases} \tag{4.7b}$$

The general principles of curriculum learning specified in Definition 4.1 can be recognized in the SPL optimization problem. At any moment during the learning process, only those samples for which the prediction loss is smaller than an a-priori set threshold κ will be included in the intermediate training distribution $\mathscr{D}'_\lambda$. SPL assumes that the prediction loss is a sufficiently representative heuristic for quantifying the complexity of learning a sample. The binary weight vector $\boldsymbol{v}$ is then used to fill $\mathscr{D}'_\lambda$ with those samples that are considered 'easy' under the current capabilities of the learner [36, 37, 51]. Upon convergence of of the learner, all samples are included in the training set, which completes the learning curriculum.

Note that SPL is not the only method for autonomous curriculum design: in [28] for example, a learning curriculum is generated by a multi-armed bandit algorithm that shapes the so-termed 'task syllabus' based on rewards that indicate learning progress. Here, learning progress can be measured through loss-driven progress, as in SPL, or complexity-based progress, where it is assumed that prediction capability of a neural network can be quantified based on the network complexity. Further discussion of the details is outside of the scope of this section, but it is important to recognize here that using prediction loss is not necessarily the only measure of learning progress.

SPL in the form presented above can lead to data overfitting quite rapidly, as the learning curriculum tends to re-select 'easy' samples over the entire learning process [37]. Moreover, as argued in [37], the entropy of intermediate training distributions does not only reside in the complexity of the samples considered, but also in their diversity. This leads to the concept of self-paced learning with diversity (SPLD), which employs a slightly adjusted formulation of the SPL performance objective (Equation 4.7a). Here, it is assumed that the complete training set can be divided in distinct groups, which often appears natural for supervised machine learning tasks. The diversity of training then resides in the fact that samples need to be selected from each of these groups. In this case, the sample weight vector $\boldsymbol{v}$ appears as a matrix, where each column corresponds to a distinct group. However, this makes the training process more complex as compared to SPL, as optimizing $\boldsymbol{v}$ becomes non-convex. Further details are out of the scope of this section, but the basic idea of ensuring diversity of the intermediate training set is important to consider.

The basic ideas of SPLD have recently been explored in reinforcement learning in [82], where an agent is trained to play Atari 2600 games. Here, a complexity index (CI) function is defined in terms of *transition complexity* as well as sample diversity. This approach, dominated Deep Curriculum Reinforcement Learning (DCRL), makes use of the same augmented learning objective of SPL, assuming the same disjoint optimization approach to optimize the agent and the curriculum design at the same time. Sample batches in the form of transitions stored in replay memory serve as intermediate training distributions:

$$\mathscr{D}'_\lambda=\left\{\left(\left[\hat{Q}(\boldsymbol{x}_t,\boldsymbol{u}_t)\right]_i,\left[r_{t+1}+\gamma\max_{\mathscr{U}}\hat{Q}(\boldsymbol{x}_{t+1},\boldsymbol{u}')\right]_i\right)\right\}_{i=1}^{n} \tag{4.8a}$$

Here, the bootstrapping nature of RL is emphasized again. Subsequently, it may become clear that the TD-error $(\delta_{t+1})_i$ is the leading factor for autonomous curriculum design in reinforcement learning. In [82], three reasons are given as to why it is important to put an upper limit on the TD errors of the intermediate training distributions: (1) high TD errors may very well be the result of noise in the reward signal or utility; (2)

[7] Example loss functions were given in Section 3.3 for the case of nonlinear optimal tracking control.

bootstrapping itself may lead to erratic learning; and (3) large TD errors violate the assumptions of first-order SGD. On the other hand, the TD error is the single factor that drives learning in approximate RL, which means that some balance between the two should be found. For this reason, ranking transition complexity requires something more than the binary evaluation proposed in SPL (Equation 4.7b). Therefore, the authors propose the 'self-paced prioritized function' $\mathrm{SP}(\delta_{t+1}, \kappa) \mapsto [0,1]$, a piecewise continuous function that describes how the TD error relates to transition complexity.

Similarly, sample diversity can be accounted for through another function, denominated the coverage penalty (CP) function, which reduces the probability that a sample is selected based on the number of times it was selected before. Then, the complexity index function consists of the sum of SP and CP, where the latter can be multiplied by some weighting factor to indicate the importance of transition complexity and diversity. By managing the entropy of the sample batches from experience replay in this way, the authors demonstrate that an agent trained by DCRL performs considerably betters compared to others trained by other state-of-the-art deep RL techniques. Moreover, both learning efficiency and robustness are shown to be improved.

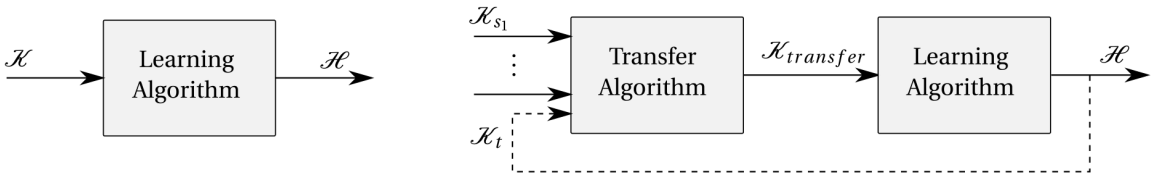
4.3. Transfer Learning

By incorporating the principles of transfer learning (TL), the scope of the curriculum learning paradigm can be considerably extended. Like the methods presented earlier, transfer learning has its roots in area of supervised learning [62]. In this section, the fundamentals of TL in the reinforcement learning domain will be broadly introduced. Being a separate research area in itself, the presentation will be limited to a general framework and taxonomy, which is followed by a brief discussion on so-called knowledge mappings. The latter is especially relevant when knowledge needs to be transferred between two related MDPs with dissimilar state-action spaces. For a more detailed overview of TL techniques, the reader is referred to e.g. [105]. Referring to the curriculum learning taxonomy presented in Section 4.1.2, the concepts presented here are mostly related to inter-task staging and the consequences thereof, as well as the flexibility of the agent domain.

4.3.1. Framework

The many views and approaches that together constitute the field of transfer learning limit the extent to which a comprehensive framework can be established. A broad definition is presented in [55], where the learner and the transfer learning algorithm are considered as mappings between the knowledge space $\mathcal{K}$ and the hypothesis space $\mathcal{H}$. This concept is illustrated in Figure 4.2. In a conventional learning process - i.e., where the target task is learned completely from scratch -, $\mathcal{K}$ typically consists of collected transition samples $z_{t+1} : (\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}, r_{t+1})$ and the agent representation specified a-priori by the designer (see also Subsection 2.5.2). In the transfer learning setting however, the proposition is that one can reduce the dependency on the knowledge space of the target $\mathcal{K}_t$ by providing a knowledge basis based on experience from different, but related source tasks. The knowledge space of each source task $\mathcal{K}_s$ may again consist of samples, but also low-level information such as value functions and even high-level information like learning rules or options [105]. Then, a transfer learning algorithm is used to map this knowledge base to a coherent representation $\mathcal{K}_{transfer}$ that enables the learner to exploit this knowledge.

In general, the TL paradigm can be exploited for a range of different objectives. This relates to the many evaluation metrics that can be used to measure the performance of a given transfer strategy, as discussed in [55, 105]. Broadly speaking, there are two leading aspects here: time-to-learn (TTL), and learning per-



(a) Conventional learning process **(b)** Transfer learning setting

Figure 4.2: Transfer learning framework as presented in [55]. In a conventional learning process (a), task-specific knowledge $\mathcal{K}$ is the only input provided to the learning algorithm to form a hypothesis $H \in \mathcal{H}$; (b) in the transfer learning setting, knowledge from one or more source tasks is used (potentially) together with knowledge specific for the target task, which is processed by a transfer learning algorithm to form a combined knowledge space $\mathcal{K}_{transfer}$. Adapted from [55].

formance. In terms of learning time, one can express transfer performance as either the total training time, which also takes into account the time spent in learning the source tasks, or the target training time, where only training in the target task is considered. In [105], the total training time is used as the preferred metric. For learning performance, there is a wider range of possibilities that can be considered. In [105], five categories are established. These are briefly repeated below:

1. *Jumpstart*: here, transfer performance is solely measured in terms of the performance gain achieved at the onset of learning. Progress during learning of the target task is of no relevance here.

2. *Asymptotic performance*: with this metric, the transfer algorithm is evaluated in terms of how it increases the performance of the learner in terms of asymptotic reward or cost incurred once the learning process has converged.

3. *Total reward*: here, the total area under the learning curve (integral) with transfer is compared to the total area without transfer. This essentially combines the concepts of target training time and learning performance in a single metric.

4. *Transfer ratio*: essentially the same as total reward, but expressed in terms of a ratio between the areas.

5. *Time-to-threshold*: with this metric, one assumes that there is some threshold that can be assigned to quantify adequate learning performance. The time to reach this threshold is then used to evaluate transfer performance.

It may be clear that any of these metrics may yield a different picture of transfer performance. For example, a transfer strategy may show no immediate improvement over random policies at the onset of learning but make steady progress thereafter, as reported in [102]. In this case, using the jumpstart metric to evaluate performance would not be appropriate. Therefore, one should carefully select an adequate metric for the problem at hand.

4.3.2. Taxonomy

With the surveys reported in [55, 105], a clear taxonomy has become available that structures the TL paradigm along a range of key dimensions. These focus on the variations in (1) allowed task differences, (2) source task selection, (3) transferred knowledge, (4) knowledge mappings, and (5) allowed learners. Each of these aspects will be briefly discussed in this Section. Here, it must be emphasized that transfer learning can be adopted in different settings. That is, one can consider single pairs of source and task MDPs only, but the same principles generally apply to the multi-task setting as well.

In general, the target MDP $M_t : \langle \mathcal{X}_t, \mathcal{U}_t, \tilde{F}_t, \tilde{\rho}_t, \gamma \rangle$ can differ (1) in multiple ways from a source MDP $M_s : \langle \mathcal{X}_s, \mathcal{U}_s, \tilde{F}_s, \tilde{\rho}_s, \gamma \rangle$ that is used for transfer. This may have a strong impact on the transfer strategy that is to be adopted, as well as the extend to which transfer can be successful. It is emphasized here that any pair of tasks considered for transfer requires them to be related in some sense, or transfer will be to no avail. Even worse, generating an inappropriate knowledge base may even hurt learning, an effect known as negative transfer [105]. This also relates to the source task selection process (2), which can be done by a domain expert or autonomously by the learning agent itself. This is the part where the learning curriculum is designed. The main ideas here are essentially equivalent to those discussed in Section 4.2. Most of the work in TL assumes that such a curriculum is specified a-priori, but recent publications also present autonomous curriculum generation [73, 74]. It may be clear that a well-designed learning curriculum ties the source and target tasks in such a way that the target knowledge base $\mathcal{K}_{target}$ results in beneficial transfer performance.

Assuming that the selected task sequence allows for positive transfer, the key distinctive factor in TL is (3) the transferred knowledge, i.e. how the source task knowledge space $\mathcal{K}_s$ materializes. Some options were mentioned already in the TL framework, including transition samples, low-level knowledge such as value functions, or other types of high-level knowledge. In general, one can distinguish between instance transfer, which directly re-uses observed experience from the source task, representation transfer, which derive representations for the target task through some abstract process, and parameter transfer, which directly initializes a target learning with knowledge gained from the source task [55]. The question of how this knowledge is to be mapped from the source to the target task is then addressed by (4) the knowledge mapping. These may again be determined a-priori by the domain expert, but can also be generated autonomously by the learner itself by identifying how the source task is related to the target task [105]. On the other hand, there are also

methods that circumvent any type of mapping by exploiting so-called task-invariant knowledge. More details on knowledge mappings are presented in the next section.

Lastly, there is the question of what learning paradigms (5) are compatible with the selected transfer strategy. For example, an approach that maps individual state-action values will not be compatible with policy search methods. This shows that the transfer algorithm is often inherently tied to the learning algorithm itself.

4.3.3. Knowledge Mappings

Having established the type of knowledge used that is to be transferred between a pair of tasks, the key towards ensuring positive transfer is to apply an adequate mapping $\mathscr{A}_{map} : \mathscr{K}_{s_1} \times \cdots \times \mathscr{K}_{s_N} \times \mathscr{K}_t \mapsto \mathscr{K}_{target}$. This is especially relevant in situations where the state-action spaces of the task pair under consideration differ in terms of dimensionality and semantic meaning, which often calls for a manual design of $\mathscr{A}_{map}$ [105]. By specifying these mappings a-priori, one adds another source of domain knowledge that enables the learner to increase its learning performance. This is also known as the concept of *inter-task mappings* [105]. In this subsection, this class of methods is further scrutinized.

The concept of inter-task mappings was introduced in [102], where low-level knowledge in the form of Q-values is used to initialize learning in a larger MDP based on a similar, but smaller source MDP. Based on how the learning domains of these MDPs semantically compare to each other, an action variable mapping is combined with a state variable mapping to form a so-called transfer functional. This transfer functional is essentially a dedicated algorithm that assigns Q-values to those state-action pairs of the target task that show a direct connection to the source. The specific way this functional is implemented depends on the type of function approximator considered, but the general idea is evident. In [102], this method is investigated for three approximators, including CMACs, RBFs, and a feedforward ANN. For the CMACs and RBFs, the transfer functional loops over the complete range of basis functions, whereas for an ANN, it loops over each of the links that connect a given pair of neurons. This approach is in fact equivalent to the concept of flexible function approximation [35], where a given approximator can be expanded (or reduced) to accommodate learning in related, but different MDP. However, here one does not explicitly make use of a transfer functional to seed the target approximator with the parameters from a source, but modifies the same approximator while keeping the existing parametrization largely intact. Note that inter-task mappings can also be used for instance transfer, which makes the procedure also compatible with model-based reinforcement learning. An example of this is the TIMBREL algorithm, presented in [103].

Although strongly dependent on how two MDPs relate to each other, a one-to-one assignment or expansion procedure is generally relatively straightforward. However, the question that arises immediately is how to initialize those parts of the Q-function that are outside of the scope of this transfer functional, i.e. for regions that go beyond the scope of the source task. In [102], it is opted to initialize the weights of those basis functions or neurons that fall under this category with the average of all other weights, or to copy the weights of a related region that shows high correspondence. In [35] however, it is suggested to either keep the information from the source MDP on the zero subspace of the newly added state or action variable, or to clone existing knowledge and project this randomly on the unknown dimension. Independent of what strategy is chosen, it may be clear that the way these unknown subsets are initialize can have a strong influence on the performance of the TL algorithm. In [102], this effect is briefly investigated.

Although hand-engineered mappings represent a major category, there are also other approaches reported in literature that depend less on explicit knowledge of the target task. One class of methods takes advantage of task-invariant knowledge, which often use some form of high-level information that generalizes over a large distribution of tasks that share common features [102]. An example of this approach is presented in [49], where a framework for learning shaping functions is investigated that allows an agent to carry over internal knowledge in the form of *sensations* between goal-directed tasks that have dissimilar state space representations. The assumption here is that both the reward signal $\tilde{\rho}$ as well as the input space $\mathscr{U}$ are invariant across the task distribution. However, this also illustrates that this class of methods can be quite limiting.

Another way to reduce the need for a-priori information on how a pair of tasks is related is to have the transfer learning algorithm generate $\mathscr{A}_{map}$ autonomously. One example of this is presented in [104], where instances from both the source and the target task are used to identify adequate knowledge mappings. In order to reduce the total amount of data required from the target task, a transition model is generated that enables prediction over a large distribution of state-input pairs observed from the source. Then, an exhaustive evaluation algorithm is run to identify the most appropriate action and state variable mappings. However, this kind of approaches normally suffers from large computational complexity.

5

Safe Learning

One of the main open problems with reinforcement learning methods and artificial intelligence in general is their potential for catastrophe on safety-critical systems [6]. With this kind of systems, there is a nonzero probability that the agent takes an action that will cause it to damage itself or its environment. In general, the safety concept can take many forms [23]. For example, the agent can exhibit negative behaviors during deployment that were not anticipated for during design and training [6]. This means that, in case of an inadequate or inappropriate objective function, even an optimal policy can lead to dangerous decisions [23]. Or, the agent may find itself in a situation that it has not encountered before, leading to the concept of robustness to distributional shift [6, 65]. An agent that cannot sufficiently generalize across different conditions cannot be expected to perform well in this kind of situations. Safety during on-line exploration forms another major item [6, 65]. When learning online, states and actions that have not been encountered or considered before cannot be adequately evaluated in terms of their safety properties. Methods that address these issues in the concept of learning-based control systems are generally collectively referred to as *Safe Learning* (SL) methods [23, 78].

The dependency of safety and its companion, risk, on the context of the learning problem has resulted in many research directions and taxonomies for safe learning algorithms. For example, one may augment the definition of the reward signal with a specification for risk aversion [25], an approach that is coherent with the idea of reward shaping [76]. However, such an approach does not take account of potentially fatal transitions, but uses the definition of forbidden states to guide the agent through the learning process, resulting in more effective training runs. Other approaches alter the optimization objective altogether, thereby restricting the space of available policies to those that are guaranteed to be safe [72]. Garcia [23] has categorized these methods under the optimization criterion of safety.

Another major class is formed by algorithms that explicitly modify the exploration process [23]. Common approaches here include directed exploration based on some heuristic related to risk [24], and methods that exploit a certain form of external knowledge. Examples of this last category include (1) learning from demonstration (also known as apprenticeship learning) [1], (2) productive initializations[1], and (3) learning in a supervisory framework. In this last category, learning typically takes place under supervision of a knowledgeable sub-optimal controller or teacher. The teacher can have various levels of autonomy: Garcia [23] makes the distinction between *ask-for-help* and *overriding* approaches. It is important to recognize that virtually any control theoretic approach can be adopted in such a supervisory framework. An example that has seen successful integration with machine learning and learning-based control is *Reachability Analysis* [4, 27]. Another class of methods that will be listed here is formed by approaches that exploit tools from control theory to limit the attainable policy space, for example by switching between selected safe baseline controllers [79] or by guiding exploration under Lyapunov stability verification [11]. This last category enables direct integration of control-theoretic tools in the framework of learning-based control, which is still considered a relatively open area in RL research [6].

In this work, the main concern is to ensure adherence to strict safety constraints during online learning of the optimal control law, i.e. during *exploration*. This calls for methods that can *guarantee* the boundedness of the system state during learning under mild assumptions of initial knowledge, i.e. that can stabilize

[1] Transfer learning methods are also listed under this category

the learning process [6]. This excludes methods that adopt a weak notion of risk (i.e. in a manner that can be considered open-loop), as e.g. the PI-SRL algorithm proposed in [22] which depends on the addition of Gaussian noise around an existing policy. In general, some prior knowledge is necessary to ensure safe exploration [65, 78]. In the literature, methods for ensuring boundedness of the system during exploration have taken multiple forms: variations have been proposed that employ safe (a-priori) backup policies, propagate a(n) (approximate) model of the system forward in time, assume locally stable initial policies, and many more. This shows that research in safe learning is quite fragmented [78]. As a result, a general framework or taxonomy is hard to identify. For this reason, this chapter scrutinizes methods and approaches that have been influential in the field of safe learning, or that have been proposed recently. Three main areas have been identified. In Subsection 5.1, safe learning methods are surveyed that employ a certain global reference that is available throughout the lifetime of the agent, most importantly at the start of exploration. This refers to approaches that have direct access to a-priori information that allows them to correct inputs generated by the learning agent, i.e. without the need for predicting trajectories on-line. Predictive methods are discussed in Subsection 5.2. This class of methods guarantee safety by propagating a(n) approximate model of the system on-line, thereby restricting the agent from entering certain regions of the state space. A third category directly constraints the attainable policy space, which is surveyed in Subsection 5.3. Here, the safe policy space can be given a-priori, or expanded on-line. Note that this is not a fully clear-cut division, as there generally is some overlap between the different categories.

5.1. A-priori Global Reference

The methods surveyed in this section have in common that they support the learning process with information that is instantly available, i.e. without the need for table look-up or input search. Typical for this kind of approaches is the presence of real backup capabilities, assisted by some form of risk detection. Subsection 5.1.1 discusses global safety functions as derived from observed minimum rewards. A less strict methodology based on supervised reinforcement learning is presented in Subsection 5.1.2. Another technique, based on artificial potential fields, is discussed in Subsection 5.1.3.

5.1.1. Safety Functions

By having access to global safety functions that are specified a-priori, an agent can be prevented from taking actions that may lead to fatal transitions. Hans et al. [32] argue that the existence of such a global safety function, in combination with the availability of a safe backup policy, is sufficient to solve the challenge of safe exploration. In their study, a transition is considered fatal if the observed reward r is below a given threshold τ. This quantification of safety gives rise to the concept of min-reward function, which returns an estimate of the minimum reward that would be observed given the current state-action pair and following the fixed backup policy $\pi(\boldsymbol{x})$ thereafter. In this way, the agent is restricted from providing inputs that can lead it to so-called *super-critical states*, which are defined as those states for which fatal transitions cannot be prevented with certainty. Such a min-reward safety function can be specified by a designer, or it may be learned from a-priori collected min-reward samples of the system. In addition to forming the primary reference for the min-reward function, the back-up policy can also be activated online in case an unknown state is visited by the agent.

Hans et al. [32] describe that obtaining an adequate global safety function is not a trivial task. In the presented case study, the min-reward function is determined by means of least-squares regression of the observed min-rewards using quadratic state features as the regression terms. In the case of state-action pairs that fall outside the regression domain, it is assumed that extrapolation is sufficient to calculate the risk level associated with a certain state. Additionally, the presented method does not take into account uncertainty of the min-reward estimate, as denoted in [72]. These points indicate that finding a good global safety function for (stochastic) MDPs with high-dimensional state-action spaces can be very challenging.

5.1.2. Supervision by a Stable Controller

As an alternative to formulating safety in terms of fatal transitions, it is often seen that a looser definition is adopted in terms of constraints. Then, safety is ensured if the system remains within the limits set a-priori by the designer. However, violation of these constraints does not strictly lead to a fatal region of the state space, if timely correction is assured. This can be achieved in the framework of supervision by a stable controller, as first proposed by [9] and later adopted by [83, 84]. In this framework, reinforcement learning is essentially combined with supervised learning (SL) of a sub-optimal, but stable policy. This gives rise to

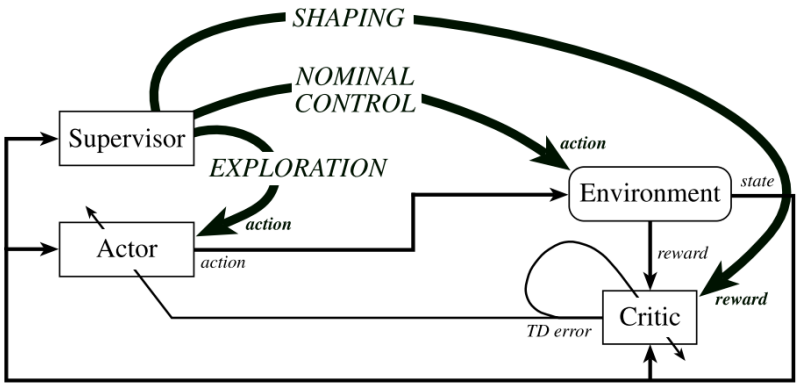


Figure 5.1: Illustration of the supervised actor-critic architecture as presented in [84]. Taken from [84].

the concept of a *supervised actor-critic* structure, as illustrated in Figure 5.1. Here, a supervisor policy is responsible for guiding the learning process in three areas: 1) it can augment the utility signal generated by an additional shaping term, making the learning process more tractable; 2) it may support the actor in selecting good actions during the exploration process; and 3) it can override the actor completely and take direct control of the system if considered necessary. In this way, the learning process is sped up considerably, while minimum safety and performance levels are ensured [84].

In the work reported in [83, 84], the supervised actor-critic framework is established by means of a composite actor consisting of the supervisor, the RL actor, and a "gain scheduling" element. The gain scheduler essentially returns an input calculated as the weighted average between the exploratory actor input $\boldsymbol{u}_E \sim \tilde{\pi}(\boldsymbol{u}|\boldsymbol{x})$ and the baseline input $\boldsymbol{u}_S = \pi_S(\boldsymbol{x})$, using an interpolation parameter $k \in [0,1]$:

$$\boldsymbol{u} = k\boldsymbol{u}_E + (1-k)\boldsymbol{u}_S \tag{5.1a}$$

At the start of exploration, the interpolation parameter is intialized as zero, representing zero autonomy for the learning controller. During this phase, learning is essentially equivalent to learning by demonstration [1]. The interpolation parameter can be updated incrementally using a *state-visitation histogram*, which can be implemented in a tabular setting or using a function approximator. The learning agent reaches full autonomy if $k = 1$. The actor update makes use of the same weighted average:

$$\boldsymbol{w}_a(t+1) = \boldsymbol{w}_a(t) + k\Delta\boldsymbol{w}_a^{RL}(t) + (1-k)\Delta\boldsymbol{w}_a^{SL}(t) \tag{5.1b}$$

Here, $\Delta\boldsymbol{w}_a^{RL}(t)$ is calculated as in e.g. Equation 3.11c, whereas $\Delta\boldsymbol{w}_a^{SL}(t)$ results from minimizing the supervisory error (i.e., the difference between the supervisor and agent input). This algorithm is successfully demonstrated by the authors in a ship steering task [83] and goal tracking using an acrobot system, as well as on a real seven degree-of-freedom robot arm manipulator [84]. However, safety constraints are not taken explicitly into account in these experiments. In fact, with this approach the assumption is made that starting from a stable policy and transferring control gradually to the learning agent is sufficient to prevent risky behaviors.

The implementation reported in [9] is slightly different, as the weighted input is activated only if one of the system constraints is violated. This is achieved by means of *guardians* or *peripheral controllers*. Based on the relative importance of these guardians and the central controller, superimposition of inputs can steer the system state away from its limits. This essentially represents a scaled backup input. Moreover, an additional punishment is added to the reward signal to shape the critic. In this way, the supervised learning framework significantly adds to safety during exploration.

5.1.3. Potential Field Techniques

As a less demanding alternative to a-priori designed backup controllers, potential field techniques using *Artificial Force Fields* (AFFs) can be considered. Artificially designed potential fields have seen applications in obstacle avoidance for robots and real-time manipulator control [40], and have also gained interest by the haptic control community [54]. In the field of reinforcement learning, AFFs have been used in solving path planning problems [114], as well as in the context of *potential-based reward shaping* [25, 76]. Artificial potential fields in their basic form as suggested by [40] consist of a collection of attractive and repulsive poles that

generate a potential field, in analogy with e.g. gravitational or electrical potential energy. Taking the gradient of the potential function $\Phi(\boldsymbol{x})$, which is scalar, results in a conservative force field $\boldsymbol{F}(\boldsymbol{x}) = -\nabla\Phi(\boldsymbol{x})$ [25]. In a control-theoretical context, this implies that a kinodynamic entity can repelled from or attracted to certain regions of its environment by selecting $\boldsymbol{u}$ proportional to $-\nabla\Phi(\boldsymbol{x})$.

Multiple forms of AFFs have been suggested in literature [54]. The basic form proposed by [40] defines potential as a function of distance between a robot manipulator and an object. The attractive potential field is then defined as the square of this distance:

$$\Phi^{+}(\boldsymbol{x}) = \tfrac{1}{2}k(\boldsymbol{x}-\boldsymbol{x}_r)^2 \tag{5.2a}$$

with k representing a tuning gain. The repulsive field potential is defined as:

$$\Phi^{-}(\boldsymbol{x}) = \begin{cases} \tfrac{1}{2}\eta\left(\tfrac{1}{\rho}-\tfrac{1}{\rho_0}\right)^2 & \text{if } \rho \le \rho_0 \\ 0 & \text{else,} \end{cases} \tag{5.2b}$$

with $\rho = \|\boldsymbol{x}-\boldsymbol{x}_0\|^2$, ρ_0 representing a measure of minimum distance, and η again a tuning gain. These fields can e.g. be superimposed on a grid using $\Phi(\boldsymbol{x}) = \Phi^{+}(\boldsymbol{x}) + \Phi^{-}(\boldsymbol{x})$. With this definition, the resultant force is nonzero most of the time, aside from regions where a (local) minimum or maximum is formed. A more sophisticated alternative by taking account of the relative velocity, giving rise to the concept of generalized potential field (GPF) proposed by [50]. In the work by [15], GPFs were further extended to make them better compatible with UAV applications, resulting in the concept of a *risk field*. The risk field returns a value between 0 and 1, indicating the risk of hitting an obstacle. This concept was further developed into *parameterized risk fields* [15], giving the designer more freedom in selecting the shape of the risk field.

As a side note, potential functions play a remarkable role in the context of reward shaping as they do not change the optimal policy when solving an MDP [76]. Here, the reward function can be augmented by a shaping term in the following way:

$$\tau(\boldsymbol{x},\boldsymbol{u},\boldsymbol{x}') = \rho(\boldsymbol{x},\boldsymbol{u},\boldsymbol{x}') + \sigma(\boldsymbol{x},\boldsymbol{u},\boldsymbol{x}') \tag{5.3a}$$

Then, if the shaping reward is selected as:

$$\sigma(\boldsymbol{x},\boldsymbol{u},\boldsymbol{x}') = \Phi(\boldsymbol{x}) - \gamma\Phi(\boldsymbol{x}') \tag{5.3b}$$

it can be shown that the optimal value function and policy will remain unaffected [76]. Note that γ represents the usual discount rate in infinite-horizon MDPs. However, the definition of the potential function $\Phi(\boldsymbol{x})$ is not fully equivalent to what has been described before. It is better described as a *descent function*, as clarified by [25]. Note that a mere alteration of the reward signal is not sufficient to guarantee safety during exploration, as argued before.

Although AFFs have been primarily applied in obstacle avoidance tasks, their use is in not limited to this kind of applications. When it comes to safe exploration, they might as well be implemented as some form of global risk function, as introduced in Subsection 5.1.1. Essentially, the risk 'force' can then be interpreted as an integrated combination of safety function and backup policy. However, it is not trivial to map this force to real low-level inputs for every risk mode, especially when the system dynamics are uncertain. This will be further discussed in Section 6.2.

5.2. Model-Predictive Methods

This section will focus on methods that propagate an approximate or bounding model to keep the system state space within a given subset of the state space $\mathscr{X} \subset \mathbb{R}^n$ given a feasible control sequence with inputs taking values in $\mathscr{U} \subset \mathbb{R}^m$. These differ from the methods discussed in the previous section in that safety is ensured by means of online predictions. This class of methods fall in the range of model-based reinforcement learning approaches, which where discussed earlier in Subsection 2.5.3. First, heuristic backup identification methods will be discussed in Subsection 5.2.1, where the aim is to let the agent explore infinitely based on risk detection and already explored states. A strongly related framework based on ranking control options is introduced next in Subsection 5.2.2, which introduces a more practical implementation for backup identification. Another approach is discussed in Subsection 5.2.3, where the computational downsides of online model propagation are further reduced by means of a table look-up. Subsequently, a related but different framework based on forward propagation of statistical models that are updated online is described in Subsection 5.2.4. Finally, again another paradigm based on the approximation of safe sets using reachability analysis is introduced in Subsection 5.2.5.

5.2.1. Heuristic Backup Identification

In applications where knowledge about the system and its environment is very limited, there is typically insufficient a-priori information available to guide the exploration process adequately. Mannucci et al. [66] present an approach that reduces the knowledge required for safe exploration to an overestimate of the system dynamics and experience gained on-line by the agent. This method is denominated as the Safety Handling Exploration with Risk Perception Algorithm (SHERPA), and has been developed in the context of UAV exploration. The fundamental concepts behind the algorithm will be briefly recited here.

SHERPA is essentially a heuristic search algorithm, and is the result of a formal approach to safe learning. It is based on the assumption that fatal occurrences are related to fatal states instead of actions, giving rise to the concept of *Fatal State Space* (FSS) [70]:

$$FSS = \left\{ \boldsymbol{x}' | \forall \boldsymbol{x} \in \mathscr{X}, \forall \boldsymbol{u} \in \mathscr{U}, \tau = \left(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}' \right) \in \mathscr{T}_{fat} \right\} \tag{5.4a}$$

with $\mathscr{T}_{fat}$ the set of transitions that lead to fatal occurrences. Then, the *Safe State Space* (SSS) is defined as the complement of the FSS, i.e.

$$SSS = \mathscr{X} \setminus FSS \tag{5.4b}$$

The risk modes, also referred to as "modes of fatal occurrence", are assumed to be known a-priori and collectively form the *Restricted State Space* (RSS). Then, the intersection of the RSS with the FSS gives rise to the *Restricted Fatal State Space* (RFSS):

$$RFSS = RSS \cap FSS \tag{5.4c}$$

The RFSS is unknown a-priori. However, it is assumed that the algorithm can perceive how close the current system state is with respect to the nearest fatal state. This is essentially a sensor-based approach and is denominated as the *Risk Perception* capability of the algorithm. Risk perception is defined as a mapping $W(\boldsymbol{x})$ that returns 1 if risk is detected in a limited perception region H and 0 otherwise. If no risk is detected, all states in the region H are added to the belief SSS. In this way, the agent gradually expands its safe exploration area. The framework is complemented by the concept of *Lead-to-Fatal* (LTF) states, which are not part of the FSS, but "evolve in the FSS with probability one" [70]. The set of LTF states is represented as L, and is defined as:

$$L = \{ \boldsymbol{x} | \forall \boldsymbol{u}(\boldsymbol{x}(t_0), t),\ \exists t : \boldsymbol{\sigma}(\boldsymbol{x}(t_0), \boldsymbol{u}(t), t) \in FSS \} \tag{5.4d}$$

With $\boldsymbol{\sigma}(\boldsymbol{x}(t_0), \boldsymbol{u}(t), t)$ representing a continuous-time state-trajectory between time t and t_0. LTF states are essentially equivalent to what Hans et al. [32] refer to as supercritical states (see Subsection 5.1.1). The key idea is that from any state $\boldsymbol{x}$, the agent must be able to return to a state visited before that lies within the SSS, i.e. it must avoid both fatal states as well as lead-to-fatal states. The predictive capabilities of the SHERPA algorithm are based on an overestimate of the system dynamics, referred to as a *bounding model*. Given the

Definition 5.1: Control backup in interval form. Taken from [66, 70].

A strictly feasible control sequence $\boldsymbol{U}_b = \{\boldsymbol{u}_k, \ldots, \boldsymbol{u}_{k+m}\}$ is a control backup in interval form if,

$$\forall i \in \{1, 2, \ldots, m\}, [\boldsymbol{x}_{k+i}] \subset SSS_k \tag{5.5}$$

and $[\boldsymbol{x}_{k+m}] \cap L = \varnothing$, i.e.,

$$\forall \boldsymbol{x}_{k+m} \in [\boldsymbol{x}_{k+m}],\ \exists p \leq k : (\boldsymbol{x}_{k+m} - \boldsymbol{x}_p) \in [\boldsymbol{\epsilon}] \tag{5.6}$$

meaning that the bounded uncertainty cannot exceed a reaching interval $[\boldsymbol{\epsilon}]$, *or*, alternatively, if it leads to an equilibrium state $\boldsymbol{x}_E$,

$$\forall \boldsymbol{x}_{k+m} \in [\boldsymbol{x}_{k+m}],\ (\boldsymbol{x}_{k+m} - \boldsymbol{x}_E) \in [\boldsymbol{\delta}] \tag{5.7}$$

with reaching interval $[\boldsymbol{\delta}]$.

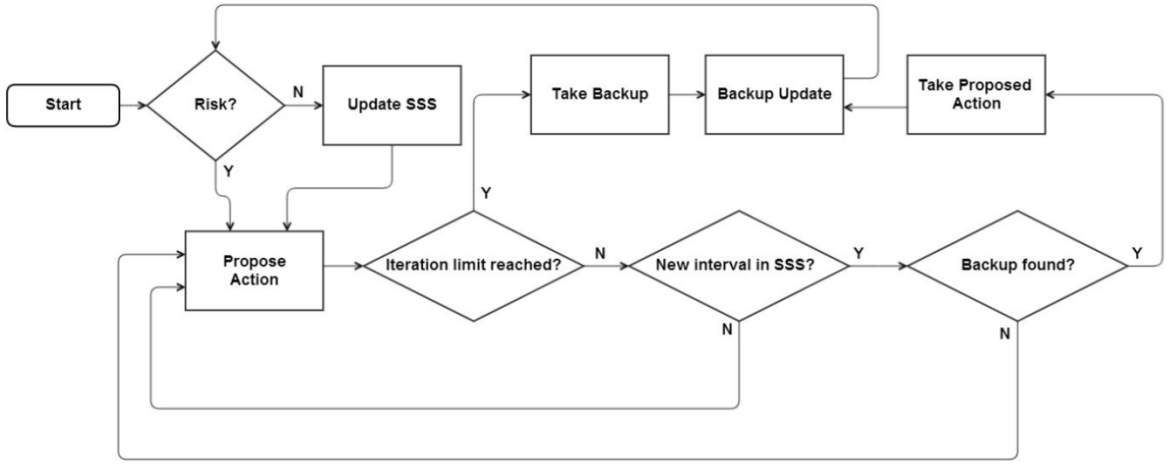


Figure 5.2: Flowchart of the procedure followed by the Safety Handling Exploration with Risk Perception Algorithm (SHERPA) developed by [65, 66, 70]. Taken from [66].

state distribution function $\tilde{F} : \mathscr{X} \times \mathscr{U} \times \mathscr{X} \mapsto [0,1]$ (see Section 2.2) giving rise to the total set of transitions $\mathscr{T}$, the bounding model $\boldsymbol{\Delta}(\boldsymbol{x}, \boldsymbol{u})$ is defined as [70]:

$$\forall \tau = \{\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}'\} \in \mathscr{T} : \boldsymbol{x}' \in \boldsymbol{\Delta}(\boldsymbol{x}, \boldsymbol{u}) \tag{5.8}$$

With this approach, if all trajectories τ predicted by the bounding model lie within the SSS and do not include any LTF states, safety of the system can be guaranteed with probability one. Forward-in-time propagation of the uncertain dynamics is facilitated by Interval Analysis (IA).

The SHERPA algorithm integrates all of these aspects in its search for control backups, which enables the agent to explore indefinitely. Mannucci et al. [66] explain that this is in fact equivalent to the condition of ergodicity [72] for continuous systems (see Subsection 5.3.1). If the predicted distribution of states $[\boldsymbol{x}_{k+1}] = \boldsymbol{\Delta}(\boldsymbol{x}_k, \boldsymbol{u}_k)$ - that is, given the proposed input - is completely in the known SSS, safe return to the neighborhood of a state visited earlier is guaranteed if a control backup sequence $\boldsymbol{U}_b(t)$ can be found for $[\boldsymbol{x}_{k+1}]$. The definition of a control backup sequence is given as Definition 5.1. The reaching intervals $[\boldsymbol{\epsilon}]$ and $[\boldsymbol{\delta}]$ form the *closeness condition* that must be satisfied by the backup sequence, and must be chosen heuristically. Additionally, they can be made dependent of the magnitude of the backup control inputs. The algorithmic logic defining SHERPA is summarized by the flowchart in Figure 5.2.

Although similar to the interpretation of control backup discussed in Subsections 5.1.1 and 5.1.2, this procedure mitigates the need for an a-priori safe policy by an on-line heuristic backup search. However, it can take a considerable number of iterations before a control backup sequence is found, especially for high-dimensional state-action spaces or systems with slow dynamics in the RSS. As a result, the computational complexity of SHERPA can be quite demanding.

5.2.2. Ranking Control Options

Although the online heuristic backup search framework taken advantage of by SHERPA guarantees the possibility for infinite exploration under limited assumptions, the probability that a feasible backup cannot be found in the available of time reduces the practical applicability of the algorithm, especially for high-dimensional systems involving multiple levels of integration. If the algorithm depletes it backup search over multiple time steps, safety of the system cannot be guaranteed in the absence of a correct control backup. A key insight here is that the best available, or 'least bad', control backup will at least perform better than no backup. Also, if the bounding model is very uncertain or prediction horizons are relatively long (as in the case of slow dynamics), it is unlikely that the conditions shown in Definition 5.1 can be met. These intuitions form the basis for SHERPA successor algorithm, denominated optiSHERPA [70]. OptiSHERPA adopts the same framework of state space labeling and bounding models as discussed in the previous section, but replaces the heuristic backup search of Definition 5.1 by a set of metrics to evaluate a finite collection of input sequences, including that proposed by the agent policy. The key features of optiSHERPA will be briefly summarized below. More extensive background can be found in [70].

First, in order to promote control inputs that keep the system close to the belief SSS, a *distance metric* is proposed that quantifies the 'distance' between a given state interval $[\boldsymbol{x}_k]$ and some known state $\boldsymbol{x}_p$:

$$d(\boldsymbol{x}_p, [\boldsymbol{x}_k]) = \left\| \left(\boldsymbol{x}_p - \frac{\sup([\boldsymbol{x}_k]) + \inf([\boldsymbol{x}_k])}{2} \right) \odot \boldsymbol{v}_r \right\| + \rho \cdot \left\| \left(\frac{\sup([\boldsymbol{x}_k]) - \inf([\boldsymbol{x}_k])}{2} \right) \odot \boldsymbol{v}_r \right\| \tag{5.9}$$

with $\boldsymbol{v}_r \in \mathbb{R}^{n+}$ a weighting term representing the level of risk associated with each state (basically a heuristic indication of the RSS), $\rho < 1$ a tuning parameter indicating the relative importance of interval width over center closeness, and $\odot$ the Schur product (entrywise product). By doing this evaluation for every state interval in the bounding trajectory τ and integrating the result over time (summation in discrete time), a single score can be assigned for every input sequence $\boldsymbol{U}$ generated by optiSHERPA and the agent. Note that the agent requires some form of prediction capability, which can be achieved by giving e.g. it a model realization from the bounding model $\boldsymbol{\Delta}(\boldsymbol{x}, \boldsymbol{u})$. Then, the first input $\boldsymbol{u}$ from the sequence $\boldsymbol{U}$ associated with the minimum accumulated distance is applied to the system. Similar as for the reaching distance used for backup identification in SHERPA, the distance metric can be corrected for the magnitude of the control input.

The distance metric is applied if the risk returned by the risk mapping $W(\boldsymbol{x})$ equals zero. If risk is perceived, another metric coined the *evasion metric* is invoked. This metric makes use of an a-priori partitioning of the state space into regions R_i, each associated with increasing levels of risk. This can simply be the SSS and the FSS, but any hierarchy of regions can be considered. Each of these regions is assigned a given weight w_i, with higher weights representing higher risk levels. Then, the intersection volume ρ_{ij} of a given trajectory $\tau_j \in \boldsymbol{\tau}$ is computed for each region R_i. The optimal evasive control sequence $\boldsymbol{U}_e^*$ is then determined using:

$$\tau^* = \arg\min_{\tau_j \in \boldsymbol{\tau}} \sum_{R_j} \rho_{ij} w_i \tag{5.10}$$

Again, the first input $\boldsymbol{u}_e^*$ is applied to the system. This is essentially a heuristic alternative to the closeness condition, although the fact that the resulting evasive control is not necessarily a true control backup means that infinite exploration cannot be assured. Therefore, optiSHERPA achieves computational tractability at the expense of losing some strong safety guarantees provided by SHERPA.

5.2.3. Graph Pruning

Instead of reducing the computational complexity of online backup identification by following a heuristic rating scheme, one can also choose to limit the state-action space by enforcing the condition of ergodicity a-priori. In Mannucci et al. [67, 69], the same overestimate of the system dynamics in the form of the bounding model (Equation 5.8) is used to form a collection of a-priori[2] look-up tables for predicting future trajectory intervals $[\boldsymbol{x}_{k+i}]$ given a control sequence $\boldsymbol{U}$. These look-up tables are generated by partitioning the particular subset of the state space $\mathcal{X}$ that follows from the operational envelope of a dynamic system, which in general does not intersect with the FSS but may contain LTF states. This partitioning is known as a *tiling* $\mathcal{T} \subset \mathbb{R}^n$, where each $\tau \subset \mathcal{T}$ forms an interval $[x_i, x_i + \Delta_i]$ for all $x_i \in \boldsymbol{x}$ within the operational envelope. Here, Δ_i represents the tiling width [65]. Based on this tiling and the bounding model $\boldsymbol{\Delta}(\boldsymbol{x}, \boldsymbol{u})$, one is able to generate a matrix for a given action $\boldsymbol{u} \in \mathcal{U}$ that - given the current bounding model - indicates all transitions to neighbouring tiles for all $\tau \subset \mathcal{T}$. Essentially, this matrix very efficiently summarizes transition intervals over state intervals based on zeroes and ones only. By generating such a matrix for a representative subset of inputs $\mathcal{U}_{sub} \subset \mathcal{U}$, a collection of look-up matrices is obtained that is referred to as a hypergraph $\mathcal{G}$. This forms the central concept for an alternative approach to preserving ergodicity, a process known as *graph pruning* [67].

When the graph is being generated, only those transition intervals that will map the subsequent tile collection within the operational envelope are labelled as feasible. By definition of the operational envelope, this implies that the resulting set of policies will be limited to those actions that will not directly lead to the FSS. However, without any further action, $\mathcal{G}$ will still include lead-to-fatal states in the predicted transition intervals. That is, there are still nodes contained in the hypergraph from which there is no return, as illustrated in Figure 5.3. As explained in [65], these are also known as *sinks* in graph theory. Consequently, the idea of graph pruning is to remove all sinks from $\mathcal{G}$, as well as the links that lead towards it. This is an iterative process, as the removal of these links may potentially lead to new sinks. Once pruning has finished, $\mathcal{G}_p$ essentially represents the policy space for which ergodicity of the system is preserved. Safe learning then comes down to verifying that a given state-action pair $(\boldsymbol{x}, \boldsymbol{u})$ is contained in the pruned hypergraph by means of a table

[2] Note that this subsection could have been included under Section 5.1 as well, but it was decided to position it here due to the large commonality with previous subsections.

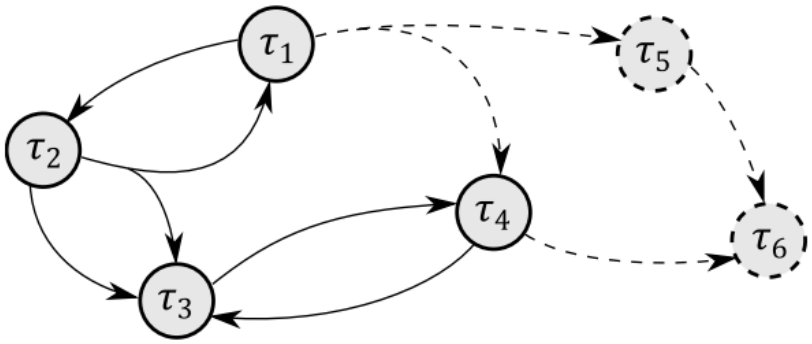


Figure 5.3: Schematic illustrating the key principles behind graph pruning, as proposed by Mannucci et al. [65, 67]. Here, the hypergraph contains a sink in the form of τ_6, which is removed in the pruning process; however, this yields τ_5 as a new sink, which shows the need for additional pruning. Taken from [65].

look-up. Note that this shows strong commonalities with the idea of global safety functions, as discussed in Subsection 5.1.1. However, in this case, uncertainty is explicitly accounted for through the bounding model.

5.2.4. Uncertainty Propagation using Statistical Models

Although the outer approximation properties of bounding models is very appealing, they generally lead to a very conservative estimate of the allowable policy space when the agent is deployed on-line (i.e., the space of policies for which ergodicity is preserved with certainty under the bounding model). The reason for this is that the bounding model is not updated during learning, but remains fixed under the a-priori determined interval bounds. If these bounds would be updated online - that is, narrowed - based on observations made from the system that the agent is interacting with, predicted future trajectory intervals would be much more accurate, given that updated bounding model still contains the true dynamics $\tilde{F}$. A model-learning implementation for bounding models has not been studied yet. However, the central idea of updating predictive models online is investigated in [48] for safe reinforcement learning based on statistical model-predictive control (MPC). Here, an outer approximation model $\tilde{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ consisting of a prior model $h(\boldsymbol{x}_t, \boldsymbol{u}_t)$ and an upper bounded estimate of the unknown model error $g(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is used to make predictions over a finite time horizon, where a control sequence $\boldsymbol{U}$ must be found given an a-priori determined optimization objective and safety constraints. A Gaussian Process (GP) function approximator is used to model $g(\boldsymbol{x}_t, \boldsymbol{u}_t)$, which is updated online based on collected samples of the system. In this way, the predictions made by the MPC algorithm become less uncertain with every update of the GP model. This technique is known as the SAFEMPC algorithm [48], and is based on the key principles of learning-based MPC (LBMPC) that was originally conceived in [7].

The MPC-based approach to achieve safe exploration is conceptually quite similar to the ideas that underlie the SHERPA and optiSHERPA algorithms discussed in Subsections 5.2.1 and 5.2.2. However, instead of performing multi-step predictions using bounding models, SAFEMPC uses ellipsoids to bound the model uncertainty for the reason that transformations can be performed in affine form [48]. Forward predictions are performed based on (1) a contribution $\tilde{f}_\mu(\boldsymbol{x}_t, \boldsymbol{u}_t)$ consisting of a linearization of the prior model $h(\boldsymbol{x}_t, \boldsymbol{u}_t)$ and the estimated mean $\mu_n(\boldsymbol{x}_t, \boldsymbol{u}_t)$ from the GP model uncertainty estimate, and (2) an upper confidence bound of the prediction mismatch $\tilde{d}(\boldsymbol{x}_t, \boldsymbol{u}_t)$. The latter is formed by a combination of the Langrangian remainder due to linearizing $h(\boldsymbol{x}_t, \boldsymbol{u}_t)$, as well as the standard deviation of the GP estimate of $g(\boldsymbol{x}_t, \boldsymbol{u}_t)$ and Lipschitz contuinity bounds. By over-approximating both contributions with ellipsoids, they can be straightforwardly combined by means of an affine transformation. This process is illustrated in Figure 5.4.

Consequently, safe learning is enforced through a similar - but not equivalent - approach as given in Definition 5.1. First, given the prediction window of width T, the trajectory bounds described by the collection of ellipsoids $\{R_{t'}\}_{t'=0}^{T-1}$, with $t' = t - t_{eval}$, must satisfy the safety constraints at all times. These constraints follow from the definition of the SSS (Equation 5.4b), which may given a-piori or identified on-line ([48] does not provide any restrictions in this regard). Second, ergodicity is ensured by a terminal set constraint $R_T \subset \mathcal{X}_{safe}$, where $\mathcal{X}_{safe} \subset \mathcal{X}$ is that part of the state space where the system can be safely controlled by an a-priori safe policy $\pi_{safe}(\boldsymbol{x})$. Now the difference with Definition 5.1 is that $\mathcal{X}_{safe}$ is assumed to be invariant, whereas earlier it was seen that the SSS can be expanded as the state space is further explored. This can significantly limit the extent to which the agent can explore $\{\mathcal{X} \setminus \mathcal{X}_{safe}\} \cap SSS$, i.e. the rate at which the agent can learn

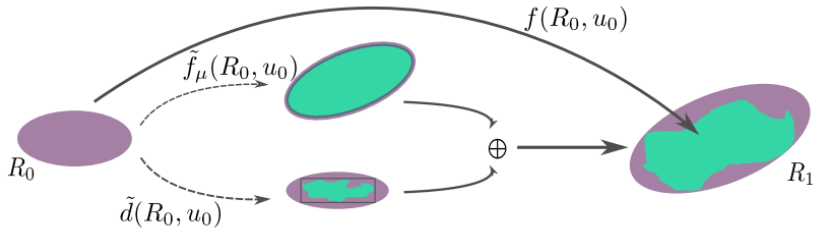


Figure 5.4: Illustration of the key principles behind forward predictions based on ellipsoidal over-approximations of the true dynamics based on an uncertain model as conceived by [48]. By virtue of the affine transformation properties of ellipsoids, a linear model contribution can be straightforwardly combined with an upper estimate of the model mismatch through the Minkowski sum $\oplus$. Taken from [48].

about the environment. In [48], this deficiency is accounted for by planning a so-called performance trajectory in addition. Nevertheless, as learning proceeds, SAFEMPC is progressively able to make better model predictions by virtue of online GP updates.

5.2.5. Approximation of Safe Sets using Reachability Analysis

The approaches discussed so far in this section have a couple of aspects in common. First, they do not alter the reward signal $\tilde{\rho}$ in any way. In case of on-policy solution methods, this becomes problematic in case a control backup is performed by an external supervisor, such as SHERPA or optiSHERPA described in Subsections 5.2.1 and 5.2.2. By contrast, this is perfectly acceptable for off-policy learning, as also described in Subsection 2.5.1. Since the control backup maps the system state $\boldsymbol{x}$ back to the known SSS, there is no way for the agent to distinguish between the open-loop system dynamics and the closed-loop dynamics under the backup policy $\pi_b(\boldsymbol{x})$ if this difference is not reflected in the reward signal. For all the agent knows, the system will transition back into the SSS as soon as it approaches the FSS. This is in contrast to e.g. the supervised actor-critic architecture as described in Subsection 5.1.2, where the reward signal is augmented by an additional shaping term. A second commonality is the dependency on an over-approximation of the uncertain dynamics to be able to make predictions. Even in the case of the SAFEMPC algorithm described in the previous subsection, which accounts for overly conservative uncertainty bounds by updating the model online, predictions are still based on an ellipsoidal over-approximation of the true dynamics. These insights show that there is considerable room for improvement here.

In [4], a model-based safe-learning framework is proposed based on reachability analysis (see also [26, 27]) that addresses these problems simultaneously. Here, system dynamics of the form $f(\boldsymbol{x}, \boldsymbol{u}) = h(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u} + d(\boldsymbol{x})$ are considered, where the (deterministic) state-dependent disturbance function $d(\boldsymbol{x})$ is unknown, but bounded by a compact set $\mathscr{D}(\boldsymbol{x})$. This term must be learned to account for model deficiencies of the system's internal dynamics, and is approximated by means of a Gaussian Process. Note that the input-affine term $g(\boldsymbol{x})$ is assumed to be completely known. Then, the SSS is described as a set $\mathscr{K} \subset \mathbb{R}^n$, where the system must remain during some time horizon τ given all possible realizations of $d(\boldsymbol{x})$ in $\mathscr{D}(\boldsymbol{x})$. That is, there must exist an input sequence $\boldsymbol{U}$ that ensures $\boldsymbol{x} \in \mathscr{K}\ \forall t \in [0, \tau]$. The idea of reachability analysis is then to compute the set of states at time $t = 0$ by solving a form of the so-called Hamilton–Jacobi–Isaacs (HJI) partial differential equation (see also [71]). The solution is found as that set of states for which the function $v : \mathbb{R}^n \times [0, \tau] \mapsto \mathbb{R}$ is positive everywhere on the domain, except on the boundary, where it evaluates to zero. This problem is essentially formulated as a differential game, where the disturbance $d(\boldsymbol{x})$ drives the system towards the FSS $\mathscr{K}^c = \mathbb{R}^n \setminus \mathscr{K}$ and the control input $u(\boldsymbol{x})$ tries to maintain $\boldsymbol{x} \in \mathscr{K}\ \forall t \in [0, \tau]$. Then, there is an optimal (i.e., safe) control strategy $\pi_b^*(\boldsymbol{x})$ that maximizes the set of initial states for which this differential game can be won. For reinforcement learning applications, a key insight is that $\pi_b^*(\boldsymbol{x})$ only needs to be applied once the agent reaches the boundaries of this set, meaning that this strategy essentially serves as a control backup policy.

Now, there are essentially two main contributions in [4] that aim to resolve the aforementioned issues. First, the utility signal is augmented by a safety metric based on $v(\boldsymbol{x})$, which returns a penalty that goes to infinity as the system state reaches the boundary of the initial safe set. An experiment is presented that shows that by incorporating this metric, the agent learns faster and requires less assistance by the backup policy $\pi_b^*(\boldsymbol{x})$. Second, the disturbance set $\mathscr{D}(\boldsymbol{x})$ is estimated and validated online based on a so-called *model-*

reliability margin. This metric allows for a sample-based evaluation of the correctness of the current estimate of $\mathscr{D}(\boldsymbol{x})$. That is, instead of putting a over-approximating bound on $\mathscr{D}(\boldsymbol{x})$ that will lead to pessimistic prospects for the controller to win the differential game, an intermediate (under-)estimate of $\mathscr{D}(\boldsymbol{x})$ will be considered sufficient unless proven otherwise based on data of the real system. This approach is shown to be very successful in a quadrotor tracking experiment in a constrained environment (control room). As a result, the available learning space is much less constrained while safety is still guaranteed.

Note that for this framework, safety of the system is only ensured over the time window $[0,\tau]$ through reachability analysis. Contrary to the other approaches described so far, this does not imply global ergodicity in case this time window is finite. However, one can select $\tau \to \infty$ to get global guarantees. Nevertheless, this comes at the expense of very high (if not infeasible) computational complexity.

5.3. Restricted Policy Search Space

In this last section, the focus will be on a range of methods that directly constrain the available policy space that can be used for finding the optimal control policy. The resulting policy space may be dynamic, meaning that it can be adjusted online, or can be determined fully a-priori. In Subsection 5.3.1, the concept of safety through ergodicity is discussed in the context of model-free safe exploration, where the search for an optimal exploration policy is constrained by the condition that the agent must be able to return to a given initial state. Another view of safety in terms of system stability is introduced in Subsection 5.3.2, where a-priori Lyapunov domain knowledge is used to restrict the available action space. Finally, the concept of Lyapunov functions is also used by the approach discussed in the Subsection 5.3.3, which restricts the policy space to those policies that keep the system within a safe region of attraction.

5.3.1. Safety through Ergodicity

The concept of preserving ergodicity during learning has been mentioned already multiple times in this survey on safe learning. The account presented in [72] has been leading in emphasizing its importance. Here, the authors explain that it forms the main enabler for safe exploration. Ergodicity implies that it is with certainty that some trajectory exists between any arbitrary pair of states in a given subset of the state space, which is often not the case for the complete state space in safety-critical systems such as those often considered in practice. In [72], this principle is used to find an exploration strategy that safely traverses the complete SSS associated with an unknown environment. During this process, the agent builds up a belief β of the dynamics $\tilde{F}$ and the reward structure resulting from $\tilde{\rho}$. It does not do so based on some form of prediction model, but ensures safe exploration only through its belief based on what it has experienced before and limited sensing abilities. Given that the probability of preserving ergodicity during exploration must at least be equal to some threshold δ, safety is ensured by means of a safe backup (or return) policy that can bring the system back to its starting state $\boldsymbol{x}_0$ from another state $\boldsymbol{x}_T$:

$$\mathbb{E}_{\beta}\mathbb{E}_{\tilde{\pi}_e,\tilde{F}}\left[\mathbb{E}_{\pi_b,\tilde{F}}\left[B_{\boldsymbol{x}}|X_T=\boldsymbol{x}'\right]|X_0=\boldsymbol{x}\right]\geq\delta \tag{5.11}$$

where $B_{\boldsymbol{x}}=\mathbb{1}(X_t=\boldsymbol{x})$ for $t\in[T,\infty)$. Note that contrary to the notation in Section 2.2, the expectation operator $\mathbb{E}_{\tilde{\pi}_\bullet,\tilde{F}}$ has been made explicitly dependent on $\tilde{F}$, to reflect that the unknown dynamics appear as a random variable under the belief. Also note that in Definition 5.1, this probabilistic measure of the existence of a backup is replaced by means of reaching intervals. Equation 5.11 now effectively serves as a safety constraint that restricts the available space of exploration policies $\tilde{\pi}_e$. It is shown in [72] that this can be done by solving two separate MDPs at every step during the exploration process: one for finding the optimal backup policy $\tilde{\pi}_b^*$, and one for finding the optimal exploration policy $\tilde{\pi}_e^*$ based on exploration bonuses under the current belief. In this way, the entire SSS can be explored in a safe, but potentially sub-optimal manner without any prior knowledge of the system dynamics or the reward signal.

5.3.2. Controller Switching using Lyapunov Domain Knowledge

In control theory, control Lyapunov functions (CLF) form a fundamental tool in the analysis of stability or tracking performance for dynamic systems [92]. Lyapunov design techniques can be used to design controllers that cause the system state to descend on an adequately chosen CLF, resulting in guaranteed convergence towards a target state or reference. As an example, backstepping (BS) control takes advantage of the fundamental concept of CLFs by designing a recursive control law that stabilizes the complete state space of a system [93]. Perkins and Barto [79] have studied the idea of using Lyapunov design knowledge in reinforcement learning to achieve safety and reliability during online learning. In this work, the authors propose

to restrict the action set available to the agent to a collection of a-priori designed control laws that cause the system state to descend on an appropriate Lyapunov function. In this way, any input selected by the agent is guaranteed to meet basic levels of successful behavior, thereby keeping the system state bounded. The most important findings from this study will be briefly summarized below.

The notion of using CLFs in the RL framework follows from a set of characteristics that must hold to provide basic safety and performance guarantees for a regulated system. Let $T \in \mathscr{X}$ represent a set of target states, corresponding to e.g. the goal state or a set of safe states. Then, any trajectory $\tau = \{\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}'\} \in \mathscr{T}$, with $\mathscr{T}$ the total set of transitions, that starts in T, must remain in T with probability one. This is referred to as the *Remain* property of a closed-loop system. A second property is that from any initial state $\boldsymbol{x}_0 \in T^c$ with $T^c = \mathscr{X} \setminus T$, it is with certainty that $\exists t \geq 0 : \boldsymbol{x}_t \in T$, denominated the *Reach* property. When combined, these properties ensure that the system will reach the set of target state and keep it there. Under some conditions however, such a guarantee cannot be given; for these systems, it is sufficient that the system state will always return to the region T. Lastly, the framework is completed by a convergence condition that ensures that the system state does in fact reach T in the limit, i.e. as $t \to \infty$.

These basic properties form the basis in extending the concept of CLFs to the framework of reinforcement learning. The authors propose two fundamental theorems, one for deterministic systems and other for stochastic systems, that allow an RL agent to satisfy essential safety and reliability conditions. These assume the existence of a Lyapunov function $v : \mathscr{X} \mapsto \mathbb{R}^+$ that is positive on T^c. For deterministic systems, it is that $\forall \boldsymbol{x} \in T^c$, it is that $\boldsymbol{x}' \in T$ or $v(\boldsymbol{x}) - v(\boldsymbol{x}') \geq c$, meaning that $\boldsymbol{x} \in T$ in a finite number of time steps. A similar theorem can be given for stochastic systems, under the provision that the Lyapunov function $v(\boldsymbol{x})$ has an upper bound. If the agent is restricted to switching between a set of control laws that satisfy these theorems, a notion of safety can be guaranteed.

The authors describe the effectiveness of their approach using multiple levels of Lyapunov domain knowledge across a range of different experiments. First, they show in a pendulum swing-up task how constraining the action set exclusively to control laws derived from CLFs results in much less variance during learning. Note that the target set T in this experiment has been extended with respect to the original goal of keeping the pendulum upright, based on the insight that any state that results in the same energy level as the target state results in asymptotic convergence to the goal. In a follow-up experiment, a wider range of control laws is available to the agent, but descending on a Lyapunov function is not guaranteed. However, the extended target set is still in place. Here, the learning process exhibits "intermediate performance". In a third experiment, the agent cannot invoke any CLF-derived control laws, but Luyapunov domain knowledge in terms of the extended target set is still available. This results again in a worsening of performance and safety. Finally, the extended target state is replaced by the original goal state, resulting in learning performance that is again significantly worse. These results shows that any form of Lyapunov domain knowledge adds to learning performance and safety.

A more advanced experiment is also reported. Here, an agent is required to stabilize a 3-link robot arm. Using feedback linearization (also known as nonlinear dynamic inversion - NDI) in combination with an LQR approach, a CLF $v_{arm}(\boldsymbol{x})$ is constructed based on the observation that the objective function used for LQR is in fact similar to a Lyapunov function. The conclusion from this experiment is again that adequate performance and safety levels can be ensured only if all inputs applied to the system descend on an appropriate CLF.

Making RL compatible with the guarantees provided by CLFs extends the relevance of these methods to on-line adaptive or approximate optimal control problems where safety must be guaranteed. However, as also discussed by the authors, the framework in its current form relies heavily on the availability of a high-fidelity description of the system dynamics and the ability to construct adequate CLFs. However, the authors expect that partially unknown dynamics can be accommodated for by using *Robust* CLFs (RCLFs) based on similar propositions.

5.3.3. Exploration in Safe Regions of Attraction

When the system dynamics are uncertain and must be identified through exploration, a stabilizing control law must be sufficiently robust to ensure stability in face of this uncertainty. As stated in the previous section, Robust CLFs (RCLFs) can be used to design fixed control laws that accommodate for bounded uncertainties (or disturbances), thereby yielding the system asymptotically stable in a given region of the state-action space [21]. However, using these control laws to restrict the available state-action space a-priori can result in premature convergence of the policy, resulting in a performance level that is not optimal. Instead, it is desirable to let the agent explore 'at will' in a region of the state-action space that is guaranteed safe and apply

a safe backup policy when the boundaries of this region are reached, as was seen multiple times before. An approach that enables this in the face of uncertain system dynamics is presented by Berkenkamp et al. [11] This framework considers robustness in the context of statistical models of the dynamics, and shows some similarities with robust nonlinear control techniques using RCLFs. With this approach, an agent can safely explore a region of the state space that is asymptotically stable with high probability under a safe backup policy $\pi_b(\boldsymbol{x})$. This region is referred to as the safe region of attraction (ROA). A key advantage of this framework is that exploration also results in guaranteed safe expansion of the ROA itself, resulting in a notion of full exploration under a backup policy that is always available. In particular, full exploration is achieved based on an iterative cycle where each stage n consists of three steps, in which 1) the ROA is identified for the current policy using the latest update of the statistical model; 2) the backup policy $\pi_b(\boldsymbol{x})$ is updated such that the ROA is expanded; and (3) new measurements are collected to further reduce model uncertainty. The most important concepts of this framework will be briefly recited below.

Learning through safe regions of attraction is based on two fundamental assumptions regarding continuity and uncertainty bounds. Specifically, it is assumed that the discrete-time dynamics $\boldsymbol{x}' = f(\boldsymbol{x},\boldsymbol{u}) = h(\boldsymbol{x},\boldsymbol{u}) + g(\boldsymbol{x},\boldsymbol{u})$, with $h(\boldsymbol{x},\boldsymbol{u})$ representing a prior model and $g(\boldsymbol{x},\boldsymbol{u})$ indicating the uncertainty with respect to the true dynamics, and the available policy space are Lipschitz continuous. Moreover, it is assumed that the posterior model has bounded confidence intervals. That is, with $\mu_n(\cdot)$ and $\Sigma_n(\cdot)$ representing the posterior mean and covariance matrix conditioned on n data points, there exists some $\beta_n > 0$ such that $\left\| f(\boldsymbol{x},\boldsymbol{u}) - \mu_n(\boldsymbol{x},\boldsymbol{u}) \right\| \leq \beta_n \sigma_n(\boldsymbol{x},\boldsymbol{u})$ with high probability. The latter assumption hints at the need for statistical models such as Gaussian Processes (GPs), although it is emphasized by the authors that any model class can be used.

The definition of the ROA is based on an a-priori known CLF $v(\boldsymbol{x})$ that allows a given policy to render the system stable. That is, if $v(f(\boldsymbol{x},\pi_b(\boldsymbol{x}))) < v(\boldsymbol{x})\ \forall \boldsymbol{x} \in \mathcal{V}(c)$, with $c > 0$, the *level set* $\mathcal{V}(c) = \{\boldsymbol{x} \in \mathcal{X} \setminus \{\boldsymbol{0}\} \,|\, v(\boldsymbol{x}) < c\}$ is a region of attraction. This set is in fact equivalent to the definition of a safe set introduced in the previous section, and therefore features the same range of properties that were discussed there. As mentioned by the authors, the CLF $v(\boldsymbol{x})$ can be constructed using first principles based on the prior model $h(\boldsymbol{x},\boldsymbol{u})$, or by virtue of the value function itself in case of a strictly positive reward function. However, as a consequence of the uncertainty in the estimate of the true dynamics $f(\boldsymbol{x},\boldsymbol{u})$, the one-step decrease condition that defines the level set must be modified. This is done by leveraging the earlier assumption on bounded confidence intervals, resulting in the following definition of a robust CLF:

$$\mathcal{Q}_n(\boldsymbol{x},\boldsymbol{u}) \doteq \left[v(\mu_{n-1}(\boldsymbol{x},\boldsymbol{u})) \pm L_v \beta_n \sigma_{n-1}(\boldsymbol{x},\boldsymbol{u}) \right] \tag{5.12a}$$

with L_v representing the Lipschitz constant for $v(\cdot)$. Then, to enforce that any safe state-action pair that has been identified before must remain safe, the resulting Lyapunov confidence intervals are intersected after each stage n:

$$C_n(\boldsymbol{x},\boldsymbol{u}) \doteq C_{n-1}(\boldsymbol{x},\boldsymbol{u}) \cap \mathcal{Q}_n(\boldsymbol{x},\boldsymbol{u}) \tag{5.12b}$$

Consequently, robustness can be captured in the Lyapunov descend condition by taking the supremum of the Lyapunov confidence intervals:

$$u_n(\boldsymbol{x},\boldsymbol{u}) \doteq \sup C_n(\boldsymbol{x},\boldsymbol{u}) \tag{5.12c}$$

To make identification of the ROA a tractable procedure, the state space $\mathcal{X}$ is discretized under a tiling $\mathcal{T}$ with tiles τ. This results in a discretized state space $\mathcal{X}_\tau$, with $\|\boldsymbol{x} - \boldsymbol{x}_\tau\| \leq \tau\ \forall \boldsymbol{x} \in \mathcal{X}$. Then, the Lipschitz continuity assumption is leveraged to generalize over all continuous states in $\mathcal{X}$. Using the Lipschitz constant $L_{\Delta v}$ to bound the Lyapunov decrease over the interval τ, the authors show that it is sufficient to check that $u_n(\boldsymbol{x},\pi_b(\boldsymbol{x})) < v(\boldsymbol{x}) - L_{\Delta v}\tau\ \forall \boldsymbol{x} \in \mathcal{V}(c) \cap \mathcal{X}$. However, this procedure is known to suffer from the curse of dimensionality. The set of all state-action pairs (i.e. not related to any policy) that satisfy this condition is denoted as:

$$\mathcal{D}_n \doteq \{(\boldsymbol{x},\boldsymbol{u}) \in \mathcal{X}_\tau \times \mathcal{U} \,|\, u_n(\boldsymbol{x},\boldsymbol{u}) < v(\boldsymbol{x}) - L_{\Delta v}\tau\} \tag{5.12d}$$

Consequently, the backup policy is updated as the policy that has the largest coverage over this parent set, i.e. that maximizes the ROA as the level set $\mathcal{V}(c_n)$:

$$\pi_{b_n}(\boldsymbol{x}), c_n = \arg \max_{\pi \in \Pi_L, c \in R^+} c : (\boldsymbol{x}, \pi_{b_n}(\boldsymbol{x})) \in \mathcal{D}_n\ \forall \boldsymbol{x} \in \mathcal{V}(c) \cap \mathcal{X}_\tau \tag{5.12e}$$

With Π_L representing the set of Lipschitz continuous policies. As a result, exploration can take place in a larger ROA. To collect further measurements, any exploration policy $\tilde{\pi}_e(\boldsymbol{x})$ can be applied, under the condition that the maximum increase on the supremum of the Lyapunov confidence interval $u_n(\boldsymbol{x}, \boldsymbol{u})$ does not cause the system to leave $\mathcal{V}(c_n)$. This verification step is necessary due to the fact that $\mathcal{V}(c_n)$ is only a subset of $\mathcal{D}_n$. The upper bound is again based on the assumption of Lipschitz continuity on the Lyapunuv function and the dynamics. This results in the set $\mathcal{S} \subset \mathcal{X}$ that can be safely explored under the existence of the backup policy. Denoting $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{u})$:

$$\mathcal{S}_n = \left\{ \boldsymbol{z}' \in \mathcal{V}(c_n) \cap \mathcal{X}_\tau \times \mathcal{U} \,|\, u_n(\boldsymbol{z}') + L_v L_f \left\| \boldsymbol{z} - \boldsymbol{z}' \right\| \leq c_n \right\} \tag{5.12f}$$

Essentially, this formulation extends the Lyapunov decrease condition towards states that can possibly transcend $\mathcal{V}(c_n)$. Note that this step is similar to the model-based forward looking approaches discussed in Section 5.2.

As a practical extension to this safe exploration framework, the authors combine the dynamic programming step for the agent policy $\pi(\boldsymbol{x})$ with the (generally intractable) backup policy identification step in Equation 5.12e with the Lyapunov descend constraint from Equation 5.12d, which results in a reformulated optimization problem. This effectively augments the policy update step with a regularization term, resulting in a policy that maximizes performance under robust Lyapunov stability constraints.

In summary, this way of using Lyapunov domain knowledge is very flexible and allows one to safely learn an optimal control law on-line under high probability safety guarantees. It requires relatively limited initial knowledge, including 1) a locally stabilizing initial backup policy $\pi_{b_0}(\boldsymbol{x})$, and 2) information about the Lipschitz constants for the dynamics, the policy, and the Lyapunov function. Moreover, safety is not affected if backup policy updates cannot be identified in a sufficiently short time interval (e.g. real-time), as the system remains guaranteed stable under the known backup policy. Using Lipschitz constants to bound exploration can however pose a serious caveat in this approach. In case of global Lipschitz constants, exploration can be very conservative due to very broad bounds in parts of the state space that do not feature large gradients. On the other hand, local Lipschitz constants can be very hard to identify.

6

Synthesis

In the preceding chapters, the key principles and approaches in the fields of adaptive optimal control, curriculum learning, and safe learning were largely discussed in isolation. It is the aim of this chapter to present a coherent view on how these fields relate to each other, and how an integrated approach leads to new opportunities in the problem of safe and efficient on-line learning of primary flight control laws. First, the case of curriculum learning in adaptive optimal flight control will be discussed. This is to be followed by a general discussion on safety in this context, and how safe learning mechanisms can be used to achieve safety in on-line learning applications. Lastly, these views will again be united to establish a framework for safe curriculum learning.

6.1. On Curriculum Learning for Adaptive Optimal Flight Control

An adequate learning curriculum is expected to speed up the process of learning adaptive optimal rate control considerably, whereas it might also lead to better asymptotic performance. The taxonomy presented in Chapter 4 shows multiple directions as to how such a curriculum can be generated. In case the domain expert has a good feel for the dynamics, constructing a learning curriculum a-priori may be a very effective approach. For example, in case the longitudinal and lateral dynamics are decoupled over a large region of the state space, a curriculum could be designed that devises separate MDPs for learning lateral and longitudinal control. This idea can be taken a step further by dividing learning for each individual axis also over multiple MDPs, where in each stage the allowed state space is extended, thereby letting the agent perform more extreme maneuvers. In each of these stages, the agent representation may be altered to facilitate more effective learning. However, it is expected that care is required to ensure all dynamics are observable on the selected representation, as also discussed in [35]. Moreover, such an inter-task learning strategy requires adequate knowledge mappings, as discussed in Section 4.3.

In case of very exotic vehicle configurations, adequate domain knowledge may not be available, which severely limits the opportunities of the designer to construct an adequate learning curriculum. This implies that the agent will have to generate its own learning curriculum autonomously, by adopting self-paced learning techniques. In this case, the airborne vehicle will be able to explore the full state-action space at will, but training of the agent will be limited to those samples that meet certain standards in terms learning complexity. As explained in Subsection 4.2, this requires some form of complexity index. In practice, this would mean that the vehicle can perform whatever extreme maneuver it likes, but that learning from this maneuver will be postponed to a later time where the agent has become sufficiently capable to incorporate it. This means that the SPL curriculum is likely to be less efficient compared to hand-engineered curricula, as it will waste a considerable amount of time performing maneuvers it has nothing to learn from yet.

However, nothing prevents one from using the principles of SPL in combination with an a-priori designed curriculum. In this case, the aircraft will be limited in its inputs and the parts of the state space it is able to explore and process, but on top of that it will further manage learning entropy on the sample-level. It can be hypothesized that this prevents the agent from incorporating samples of badly observable dynamics when solving intermediate MDPs, which can occur e.g. if the curriculum is not fully adequate. In this way, the strengths of SPL and a-priori learning curricula are reinforced to form a hybrid, better learning curriculum.

6.2. On Safe Learning for Adaptive Optimal Flight Control

In order not to damage the learning agent or its surrounding environment, appropriate safety measures are required in situations where on-line reinforcement learning is applied to safety-critical systems. This requires an adequate specification of the safety constraints, as discussed at length in Chapter 5. In the context of autonomous learning of adaptive optimal flight control laws for an aerial vehicle with uncertain dynamics, these constraints follow from the vehicle's operational envelope, intersected with the available learning space (e.g., a control room). The vehicle's risk profile - that is, the restricted state space (RSS) - forms the governing factor here, and must therefore be fully known a-priori. By specifying the learning environment a-priori, the learning process essentially becomes a closed system since its boundaries are controlled (i.e., remain free from any external factors). Safe learning is then assured if ergodicity is preserved throughout the learning process, meaning that exploration must be restricted to those regions in the state space where the system can be recovered through a feasible control strategy. Note that ergodicity cannot be enforced if there is no clear view on the risk modes of the system. These conclusions were already made in [65].

In general, ergodicity can only be guaranteed if a feasible control strategy is known to exist - that is given the current belief of the system dynamics[1] - that brings the system state $\boldsymbol{x}$ back to a safe region $\mathcal{X}_{safe} \subset \mathcal{X}$ and keeps it there with probability one as $t \to \infty$. This implies that for systems with continuous state and input spaces, the existence of such a control sequence cannot be proven without an internal model. Guaranteed safe autonomous learning of adaptive optimal flight control laws is therefore inherently model-based, although this model may be learned online. In fact, as discussed in Subsections 5.2.4 and 5.2.5, online adaptation of the dynamics model is necessary to prevent overly conservative forward predictions. In case the prediction model is static and suffers from (very) large uncertainties, there may even be no control strategy at all under the belief that provably brings the system state back to $\mathcal{X}_{safe}$, meaning that all safety guarantees are lost. The latter is reflected by the optiSHERPA algorithm discussed in Subsection 5.2.2, which effectively ranks backup sequences based on their likelihood of not succeeding in recovering the system. Reducing these uncertainty bounds by adapting the model online based on observations from the real system may prevent one from having to forfeit ergodicity guarantees. This becomes even more relevant in case the system dynamics contain slow dynamics and intense couplings across states.

6.3. On Safe Curriculum Learning for Adaptive Optimal Flight Control

From the account on safe learning given in the previous section, it can be deduced that curriculum learning alone cannot achieve ergodicity in general. That is, without any form of safeguard that prevents the agent from driving the system into the fatal state space, safety during learning cannot be guaranteed. However, a key hypothesis is that an adequate learning curriculum can have a highly positive impact on dynamic stability during learning, which is insufficient for preserving ergodicity, but may considerably expedite the search for backup control strategies. That is, given an intermediate policy $\pi(\boldsymbol{x})$ that stabilizes the system in some region $\mathcal{X}^{\pi}_{stable}$, a safe backup control sequence $\boldsymbol{U}_b \subset \mathcal{U} \times [0, T]$ that drives the system back to a safe region $\mathcal{X}_{safe}$ from a given state $\boldsymbol{x} \in \{\mathcal{X} \setminus \mathcal{X}_{safe}\} \cap \mathcal{X}^{\pi}_{stable}$ may be found with larger probability. The hypothesis here is that the available space of safe backup policies may be larger if the system remains within a local stable region, given that a the safe backup trajectory could be very complicated in the opposite case. In the case of inter-task curriculum learning, safety simply resides in meaningful initialization of the learning process, which is classified by Garcia et al. [23] as a form of incorporating external knowledge.

Moreover, by leveraging domain knowledge to design an a-priori sequence of intermediate learning tasks, the problems that arise in guaranteeing ergodicity during learning can be further relaxed (see also Subsection 4.2.1). In Section 6.1, it was already proposed that learning complexity can be significantly reduced by devising multiple intermediate MDPs with different state and input spaces $\mathcal{X}_{learn} \subseteq \mathcal{X}$ and $\mathcal{U}_{learn} \subseteq \mathcal{U}$. First, consider the case where $\dim_{\mathbb{R}}(\mathcal{X}_{learn}) = n$ and $\dim_{\mathbb{R}}(\mathcal{U}_{learn}) = m$, meaning that the dimensionality of intermediate learning problems covers the full state-action space of the system under consideration. Then, the first phase of a learning curriculum may constrain $\mathcal{X}_{learn}$ artificially to reduce learning complexity. A key insight is that this has a positive impact on learning safety as well, since constraining $\mathcal{X}_{learn}$ will prevent the agent from driving the system towards the edges of the operational envelope. That is, the system can be recovered more easily by a feasible control strategy when the system approaches the boundaries of $\mathcal{X}_{learn}$. In the second case where $\dim_{\mathbb{R}}(\mathcal{X}_{learn}) < n$ and $\dim_{\mathbb{R}}(\mathcal{U}_{learn}) < m$ - that is, when learning is restricted to particular dimensions of the system only - another favorable ergodicity characteristic arises. By isolating the system dynamics from coupling effects, it is hypothesized that a backup control sequence can be identified more easily

[1] A more formal account of characteristics necessary to ensure safety was given in Subsection 5.3.2

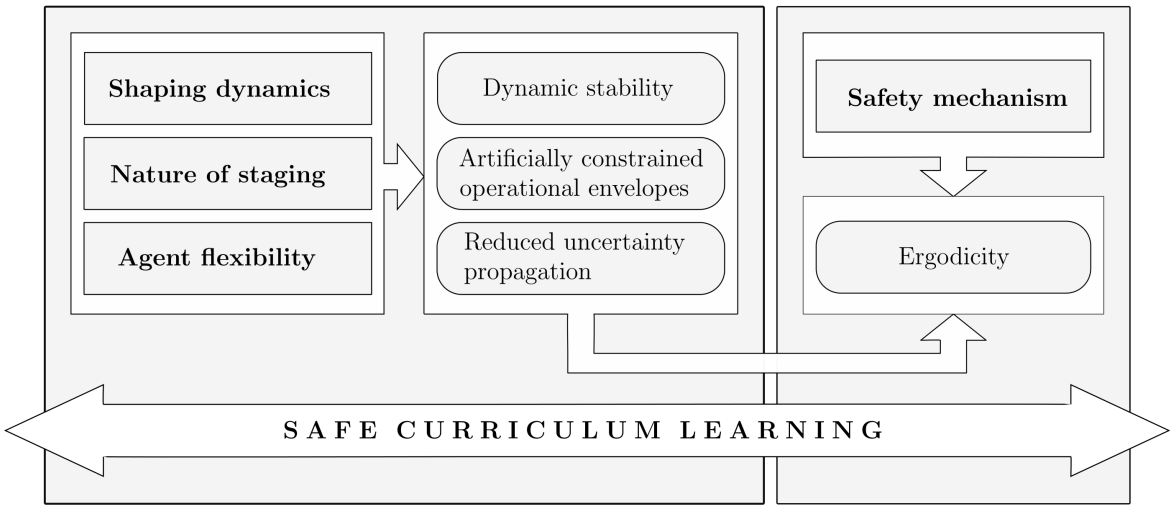


Figure 6.1: Illustration of the safe curriculum learning framework. Curriculum learning may promote safe behavior in multiple ways, but will not be sufficient to achieve ergodicity in general. An external safe learning mechanism for ensuring ergodicity is therefore necessary.

in case the system dynamics are uncertain. That is, uncertainties that are related to coupled dynamics will not propagate trough in the backup computation for the constrained MDP. As a result, the likelihood that a feasible and provably safe backup can be found is increased.

However, this case requires an adequate addressing of controllability and observability, starting with the former. Here it is that $\dim_{\mathbb{R}}(\mathscr{U}_{learn}) < m$, which means that the agent has no authority over a given subset of the available control effectors and will therefore not be able to stabilize the system in a given dimension. This calls for a robust supervisor that regulates those parts of the system that are inaccessible to the learner. This supervisor must be designed a-priori based on the uncertain model. In principle, this requires no additional knowledge about the system dynamics, as the uncertain model is already required for ensuring ergodicity during the online learning phase in the first place. However, this may lead to issues with observability during learning, as non-optimal regulation may introduce dynamics that are not observable under the intermediate MDP. This was already addressed in Section 6.1.

Finally, another safety aspect arises when a learning curriculum is considered where the agent representation may change (flexible agent). In this case, a crucial question is how to initialize the value function and/or policy for those parts of the state space for which no knowledge is available from the previous learning stage. Assuming that equivalent knowledge can be transferred perfectly between two representations (which is already doubtful in general), assigning wrong information to unknown regions may result in the negative transfer phenomenon (see Subsection 4.3.2). This shows again that curriculum learning alone is not sufficient for achieving safe learning in general. However, on the other hand, if the domain expert has a very good feeling for the dynamics, the required learning time using the new representation can be reduced to a minimum, thereby also maximizing learning safety.

A graphic overview of safe curriculum learning for adaptive optimal control is given in Figure 6.1. This diagram can be viewed as an extension to the curriculum learning taxonomy presented in Section 4.1.2.

III

Preliminary Experimental Study

This part describes a preliminary study that served to investigate several curriculum approaches and safe learning techniques on an empirical basis. The results from these experiments were used to identify the main opportunities and challenges related to these fields, and determine how these could be combined into a single, integrated framework. This has lead to the development of the safe curriculum learning paradigm described in Part I.

This part is based on previously graded work.

7

Optimal Tracking of a Linear Cascaded Mass-Spring-Damper System using online Q-learning

This presents chapter the findings from a preliminary experiment based on a cascaded mass-spring-damper (MSD) system that has been performed to investigate the proposed safe curriculum learning framework for adaptive optimal control. The advantage of the MSD system is that its dynamics are fully linear, which means that the true value function is quadratic in the state. Since the preliminary experiment focuses on optimal tracking, this implies that the theory on Linear Quadratic Tracking (LQT) presented in Section 3.2 can be applied. The consequence is that the learning problem becomes particularly simple, which enables us to focus on the safety and efficiency aspects of safe curriculum learning. Another benefit of the cascaded MSD system is that task complexity (see Subsection 4.1.1) can be altered in a straightforward manner: this can e.g. be done by modifying the dimensionality of the (observable) state-space, adjusting the number of actuators to be controlled by the learning agent, or regulating the effect of dynamic couplings.

The experimental setup is illustrated in Figure 7.1. The cascaded MSD system consists of three masses m_i that are connected along a single dimension via springs k_i and dampers c_i. The parameters associated with these springs and dampers can take negative values, which leads to dynamic instability. Each mass features a unique actuator, which implies that the system is fully controllable and there are no hidden dynamics. Safety resides in the fact that the outer mass m_3 must remain within a given region around its equilibrium point, which is indicated as SSS_3 in the figure. Once m_3, which is modeled as a point mass, hits the boundaries of this safe region, the system is said to have crashed and learning terminates.

Since it is desired to learn the optimal policy with as little knowledge about the system dynamics as possible, online Q-learning will serve as the preferred LQT algorithm. With this technique, the value function takes the following form:

$$Q(\boldsymbol{X}_t, \boldsymbol{u}_t) = \begin{bmatrix} \boldsymbol{X}_t \\ \boldsymbol{u}_t \end{bmatrix}^T \begin{bmatrix} H_{XX} & H_{Xu} \\ H_{uX} & H_{uu} \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_t \\ \boldsymbol{u}_t \end{bmatrix} \tag{3.9c}$$

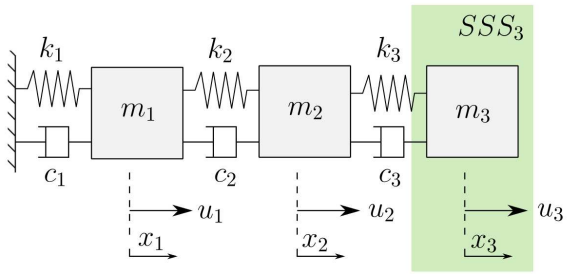


Figure 7.1: Experimental setup of the cascaded mass-spring-damper (MSD-3) tracking experiment

Algorithm 3.6: Online (data-driven) value iteration (VI) for LQT based on Q-functions; based on [42, 44]

input : Dimension $n+m+p$ of the augmented state-action space, augmented state weight matrix Q_1, input weight matrix R, discount rate γ, threshold η
output: Near-optimal kernel matrix H, near-optimal control feedback matrix K_1

initialize K_1^0 as arbitrary stabilizing control feedback matrix;
$j \leftarrow 0$;
repeat
  solve $\boldsymbol{Z}_t^T H^{j+1} \boldsymbol{Z}_t = \boldsymbol{X}_t^T Q_1 \boldsymbol{X}_t + \boldsymbol{u}_t^T R \boldsymbol{u}_t + \gamma \boldsymbol{Z}_{t+1}^T H^j \boldsymbol{Z}_{t+1}$;
  $K_1^{j+1} \leftarrow H_{uu}^{-1} H_{uX}$;
  $j \leftarrow j+1$
until $\left\| H^{j+1} - H^j \right\|_\infty < \eta$;

with H referred to as the state-action kernel matrix, or simply the kernel. The theory behind LQT Q-learning was explained in detail in Subsection 3.2.4. The Hamilton-Jacobi-Bellman (HJB) equation - which in this case amounts to solving the discrete-time Algebraic Ricatti Equation (ARE) - will be solved via Value Iteration (VI), since this option does not require an initial stabilizing control policy (recall Subsection 3.2.2) and there is no need for an additional risk term in the utility signal (recall Section 6.2). This implies that Algorithm 3.6 will be adopted, which has been repeated above for clarity. Policy evaluation will be performed via batch least-squares, which extracts as much information as possible from only a limited number of samples. Curriculum learning will be investigated primarily from the perspective of nature of staging and agent flexibility, which implies that the effect of different shaping dynamics will not be investigated. The reason is that for a linear problem with quadratic representation of the value function, all samples are of equal value to the learning process, which implies that self-paced selection techniques will be of little[1] benefit. Consequently, the entropy of the training distribution will only be managed through a sequence of a-priori designed stages with increasing learning complexity.

The chapter is structured as follows. First, the performance of the LQT Q-learning agent when learning fully from scratch will be presented in Section 7.1. This is to be followed by an investigation of inter-task curriculum learning in Section 7.2, where both learning efficiency and stability will be concerned. An analogous study will be performed for intra-task curriculum learning in Section 7.3. Next, safety through ergodicity will be investigated in Section 7.4, where the SHERPA and optiSHERPA algorithms will be studied. Finally, this is to be followed by a safe curriculum implementation, where safety through ergodicity will be enforced in the learning curriculum to achieve provably safe learning.

7.1. Flat Learning

In this section, it will be examined how the LQT Q-learning agent performs when learning the MSD-3 tracking task fully from scratch. This plain learning variant will serve as the benchmark case, which allows us to better quantify the effectiveness of the (safe) curriculum learning framework. The dynamic characteristics of the system are summarized in Table 7.1, together with a specification of the utility signal and the discount rate that define the MDP. The negative stiffness parameter and damping coefficient for respectively the first and second element indicate that this system is inherently unstable, which means that the system can only be maintained within the SSS_3 region by means of a stabilizing or ergodicity-preserving policy. The utility signal is of the form given in Equation 3.6c, i.e. corresponding to:

$$r(\boldsymbol{X}_t, \boldsymbol{u}_t) = \boldsymbol{X}_t^T Q_1 \boldsymbol{X}_t + \boldsymbol{u}_t^T R \boldsymbol{u}_t \tag{3.6c}$$

and only penalizes the position error and the actuator input for each mass element. Note that all states and inputs are assumed to be fully observable and free of noise and disturbances. Each actuator has symmetric saturation limits at ± 5 N, which results in the existence of lead-to-fatal (LTF) states which must be prevented at all times. In order to comply with the discrete-time MDP framework, the continuous-time dynamics are

[1] In Section 6.1, it was hypothesized that SPL techniques could be used to address observability issues. However, this hypothesis will not be further considered here.

Table 7.1: Benchmark parameters for the MSD-3 MDP

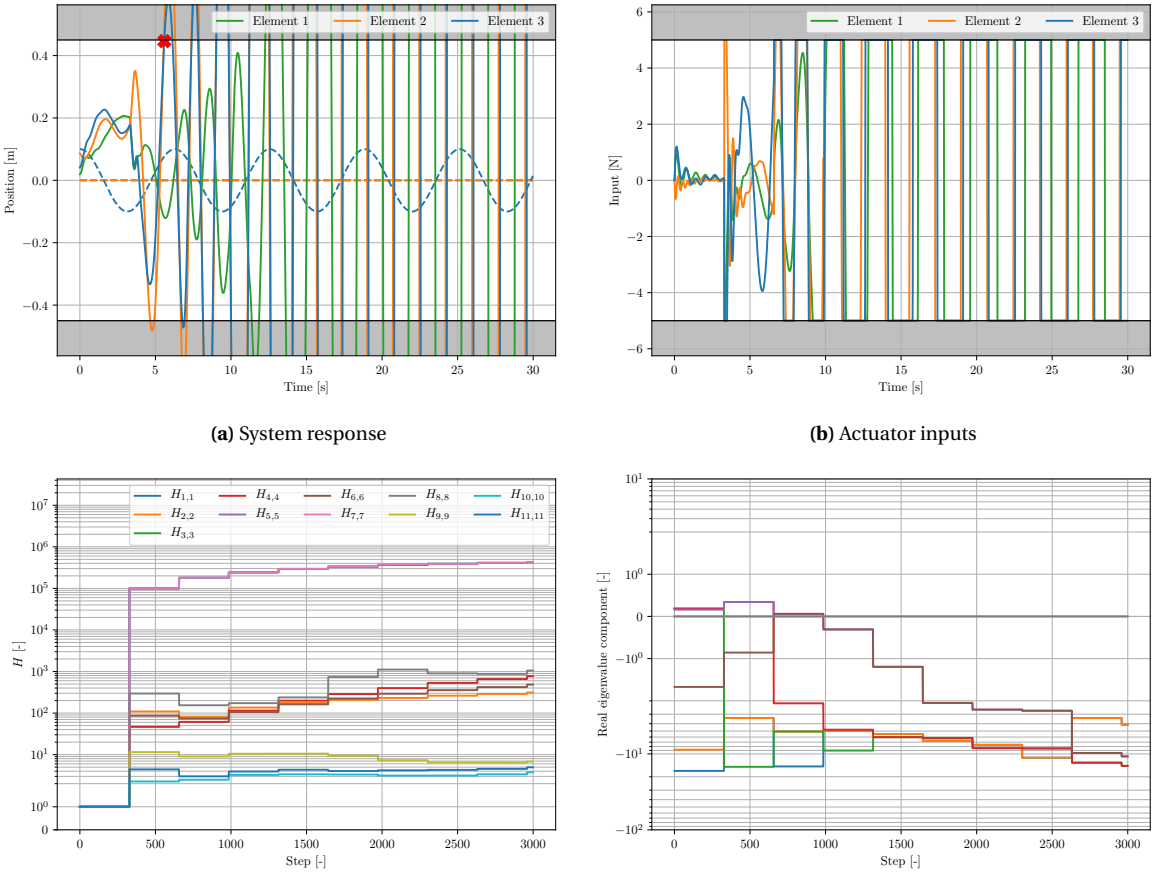
	m [kg]	k [N/m]	c [N / ms^{-1}]
Elem. 1	0.3	−1	6
Elem. 2	0.8	3	−1
Elem. 3	0.4	4	3
$\mathscr{U}_i[N]$	$[-5,5]$	Q [-]	1e5I_3
γ [-]	0.8	R [-]	1e0I_3

Table 7.2: Exploration inputs

	Excitation signal [N]
Elem. 1	$\frac{2}{0.1t+1}\left(\sin\frac{1}{10}t+\sin t+\sin 10t\right)$
Elem. 2	$\frac{-1}{0.1t+1}\left(\sin\frac{2}{10}t+\sin 2t+\sin 20t\right)$
Elem. 3	$\frac{2}{0.1t+1}\left(\sin\frac{1}{30}t+\sin 3t+\sin 30t\right)$

discretized by means of a Taylor series expansion that is truncated after the 10th order term. The sampling time is taken as $\Delta t = 0.01$.

Since policy evaluation will be performed using batch least-squares, the minimum number of independent samples required equals $(n+m+p)\cdot(n+m+p+1)/2$ [44]. However, to ensure that the regression matrix is always properly conditioned, a multiple of this minimum will be used for the pseudo-inverse (see also Equation 3.8). Taking this multiple equal to 5 resulted in good numerical robustness properties in practice. The command generator $\Psi(\boldsymbol{x}_t^r)$ generates sinusoidal reference signals, which means that its dynamics are of second-order (i.e., $p = 2$). With $n = 6$ and $m = 3$, this implies that the policy will be updated every 330 time steps. The system dynamics have been tailored such that, despite its inherent instability, the system state will remain within SSS_3 before the first policy update has taken place. In this way, a fair comparison can be made. Throughout the experiment, the position of the outer mass m_3 must follow a reference sinusoid of amplitude 0.1 m and frequency $\frac{1}{2\pi}$ rad/s, whereas the position of the inner masses must be regulated towards zero. Per-



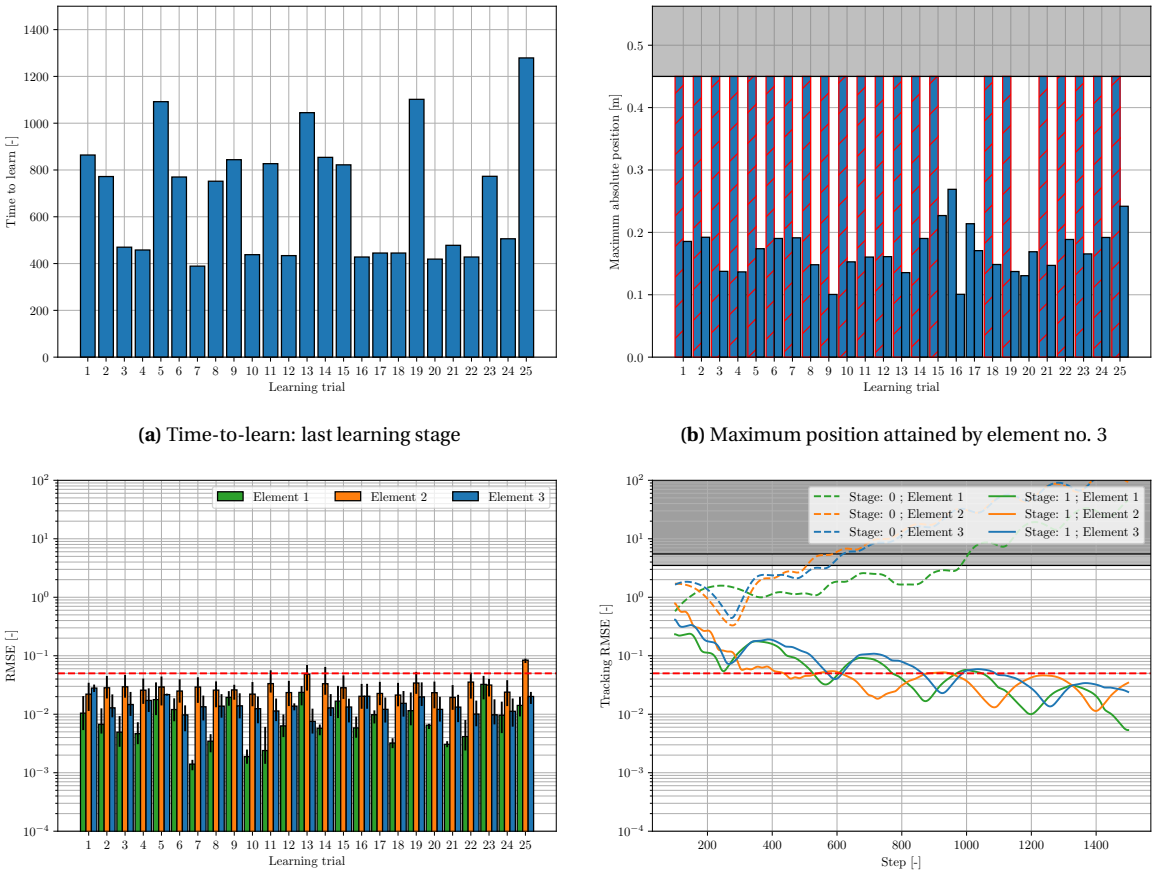
(a) System response

(b) Actuator inputs

(c) Kernel evolution (diagonal only)

(d) Real eigenvalue components of closed-loop system

Figure 7.2: Example learning trial for the MSD-3 tracking task when learning from scratch (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)



(a) Time-to-learn: last learning stage

(b) Maximum position attained by element no. 3

(c) RMSE at end of last learning stage

(d) RMSE evolution (example trial)

Figure 7.3: Learning performance statistics over 25 independent trial runs for the MSD-3 tracking task when learning from scratch (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; no safety mechanism; random init. with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

sistence of excitation (PE) is ensured by means of external inputs consisting of sums of sinusoids that decay over time. The excitation signals used for each element are summarized in Table 7.2.

Figure 7.2 illustrates typical learning performance when learning from scratch and each element is initialized with random positions and velocities $x_i \in [-0.2, 0.2]$ and $\dot{x}_i \in [-0.25, 0.25]$, respectively. The results show that the first policy update does not lead to closed-loop stability and that the system is driven out of the available learning space almost immediately, which is indicated by the red marker as a crash. Normally, this would imply that the learning trial terminates, but here the simulation continues to find out whether a stabilizing policy can be found that may even succeed in bringing the tracking error below a 5% threshold. Figure 7.2d shows how the real eigenvalue components of the closed-loop system evolve over time. It can be observed that the first policy update in fact destabilizes the system even more than it already is, which qualifies as unsafe learning behavior. However, a stabilizing control law is found after two more policy updates, meaning that the system would in principle be recoverable. Nonetheless, Figure 7.2b shows that the agent has too little control authority, which means that is no longer possible to dissipate sufficient momentum at this stage. Therefore, a full system reset would be necessary to make the learning process continue.

In order to have a benchmark for learning time (i.e., apart from safety), it is assumed here that the system is reset after 15 seconds of learning. This implies that learning takes place over two stages, which can also be considered as episodes (which is the terminology often used for off-line optimization, as may also be recalled from Chapter 2). However, it must be born in mind that this is not very realistic, as the system was seen to have crashed already during the first phase. Figure 7.3 presents the results for this two-stage learning process over 25 independent trial runs with random initialization of the system. Evaluation of learning performance is based on the following metrics:

1. *Time-to-learn*: time passed by before learning performance reaches a certain threshold. Here, the

threshold is set at 5% tracking or regulation root-mean-square-error (RMSE) relative to the maximum value of the reference signal. Learning is only considered successful if all elements meet this condition;

2. *Performance at the end of learning*: measured as the tracking or regulation RMSE during the last 100 time steps of the last learning stage;

3. *Maximum absolute position*; max. absolute position attained by an element during a learning stage.

First, it can be seen from Figures 7.3a and 7.3c that the policy does ultimately lead to tracking errors of 5% RMSE or less for all learning trials. On average, this point is reached after about 6-8 seconds of learning in the second stage, indicating that the stabilizing control law that was found from the first stage gives the agent the possibility to perform meaningful learning. This can also be observed from Figure 7.3d, in which a typical evolution of the RMSE during each learning stage is visualized. Second, Figure 7.3b shows that the system crashes almost consistently during the first stage for the majority of the learning trials, apart from a few exceptions. This shows that learning optimal control for the MSD-3 system from scratch without any safety mechanism is inherently unsafe.

7.2. Inter-task Curriculum Learning

With the flat learning benchmark in place, it can now be investigated to what extent curriculum learning can advance the learning process in terms of safety and learning speed. In particular, the ability of curriculum learning to stabilize the system dynamics will be scrutinized. The absence of any safeguard implies that preservation of ergodicity cannot be guaranteed, which means that all one can hope for in terms of safety is that learning stabilizes sufficiently quickly. However, it must be noted that the MSD-3 learning task presents a special case in terms of safety through stability: if the agent manages to keep the energy in the system bounded to a certain safe limit, the system state will always remain in the safe region SSS_3. In this case, ergodicity will be preserved implicitly.

Figure 7.4 illustrates the inter-task curriculum design. The learning process has been split in three stages, where at each stage the system complexity increases. In stage 1, the system will only consist of the outer element m_3, meaning that $\mathscr{X}_1 \subset \mathbb{R}^2$ and $\mathscr{U}_1 \subset \mathbb{R}$. This is followed by stage 2, where the inner mass m_2 and corresponding actuator are added, and finally stage 3, which corresponds to the original MSD-3 learning task with $\mathscr{X}_3 \subset \mathbb{R}^6$ and $\mathscr{U}_3 \subset \mathbb{R}^3$. At each stage, the system parameters listed Table 7.1 will be maintained. Consequently, the learning curriculum consists of three related, but different MDPs, with the agent representation expanding with every subsequent MDP. Recall from Section 4.1 that for such a curriculum design, knowledge mappings are required to transfer experience across every MDP. In this case, a low-level transfer approach is used based on *kernel mappings* in order to bias learning in every stage (recall Subsection 4.3.3). This means that the target kernel H_{target} will be mapped explicitly based on the source kernel H_{source}, where H_{source} corresponds to the converged kernel from the previous stage. Each mapping corresponds semantically to the curriculum design shown in Figure 7.4. However, since there are multiple options regarding the initialization of the unknown parts of the state-action space, a range of different strategies is conceivable. In this Section, the effect of two mapping strategies will be scrutinized.

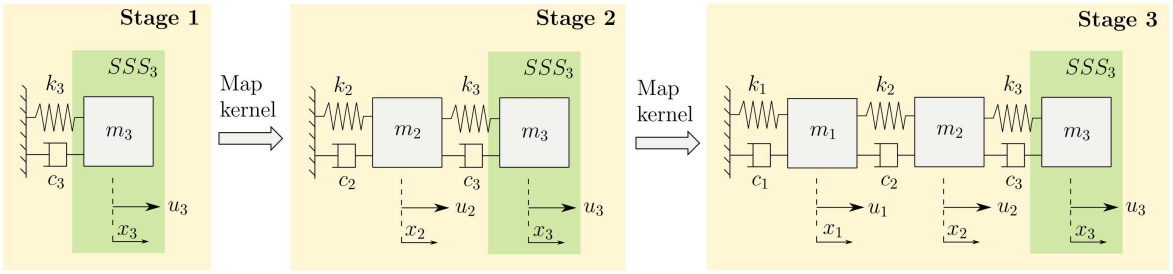
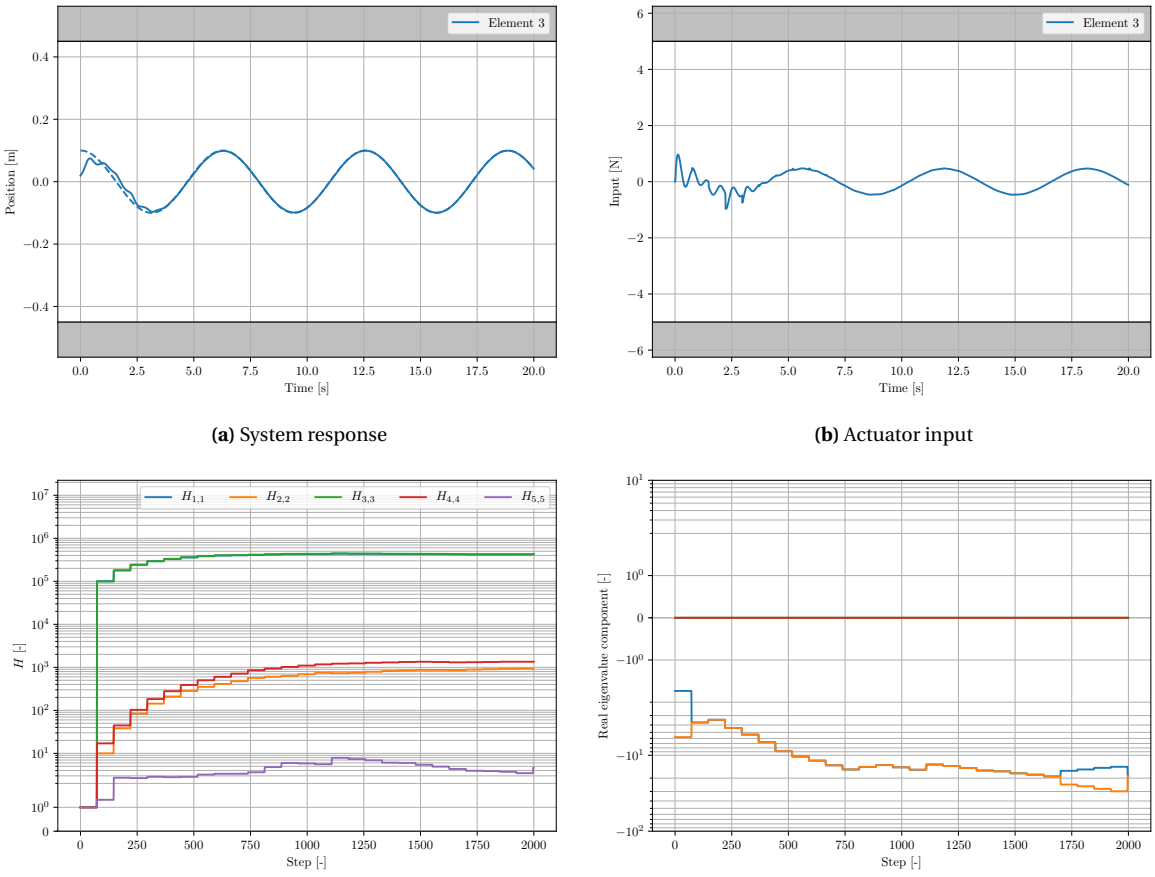


Figure 7.4: Inter-task curriculum setup for the cascaded mass-spring-damper (MSD-3) tracking experiment

7.2.1. Basic kernel mapping strategy

Figure 7.5 visualizes learning performance during the first stage of the learning curriculum. Since the MSD-1 system is inherently stable and learning remains free from destabilizing policy updates, the system always



(a) System response

(b) Actuator input

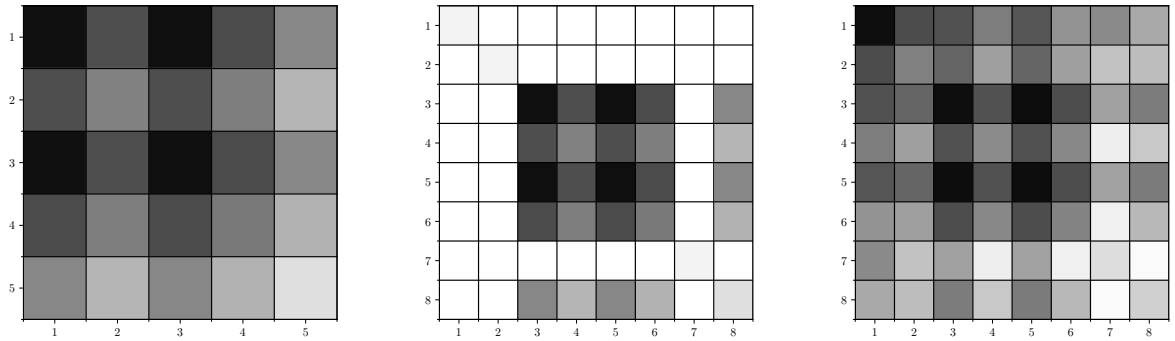
(c) Kernel evolution (diagonal only)

(d) Real eigenvalue components of closed-loop system

Figure 7.5: Example learning trial for the MSD-3 stage 1 tracking task as part of the inter-task learning curriculum (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

stays within the safe learning space. The low complexity and dimensionality of the problem leads to short learning times, as can also be observed from Figure 7.10d. Consequently, the learned policy is of sufficient quality to seed the kernel for stage 2.

A basic kernel mapping strategy will be investigated first. This mapping is visualized in Figure 7.6 for transfer between stage 1 (the source) and stage 2 (the target). Figure 7.6a shows the converged kernel matrix

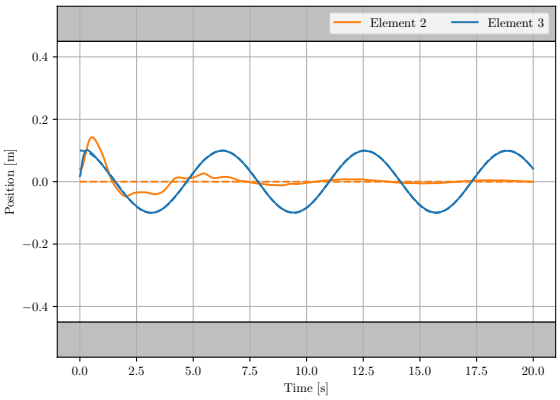


(a) Source kernel H_{source}

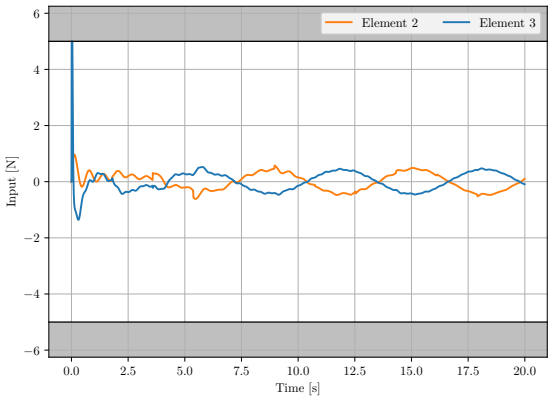
(b) Mapped kernel H_{target}

(c) Converged kernel H_{target}

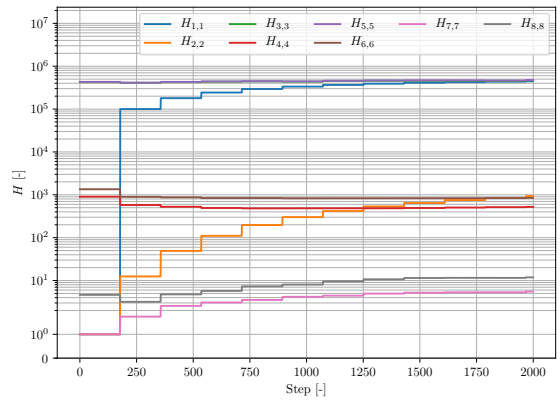
Figure 7.6: Basic kernel mapping for the MSD-3 stage 2 tracking task as part of the inter-task learning curriculum; grayscale depth is based on logarithmic scale, equivalent levels of intensity implies same order of magnitude.



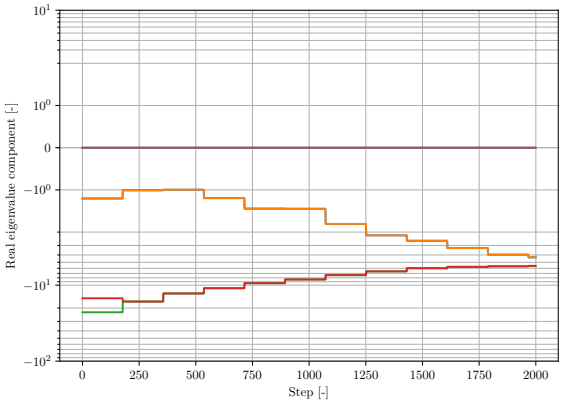
(a) System response



(b) Actuator input



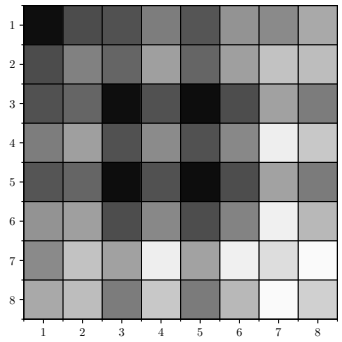
(c) Kernel evolution (diagonal only)



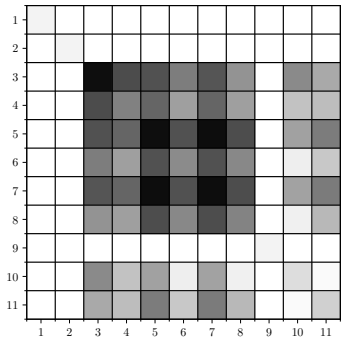
(d) Real eigenvalue components of closed-loop system

Figure 7.7: Example learning trial for the MSD-3 stage 2 tracking task as part of the inter-task learning curriculum, with basic initial kernel derived from stage 1 (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

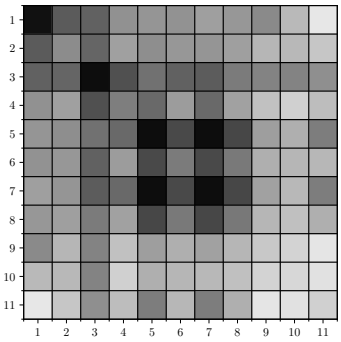
from stage 1, which is to be mapped to the target kernel directly based on symantic correspondences. The target kernel is first initialized as the identity matrix, to ensure that the matrix is invertible. Subsequently, the basic kernel mapping is applied. The result is shown in Figure 7.6b. Note that apart from the ones on the diagonal, there is no effort made to seed the value function for the unknown state-action space, which



(a) Source kernel H_{source}

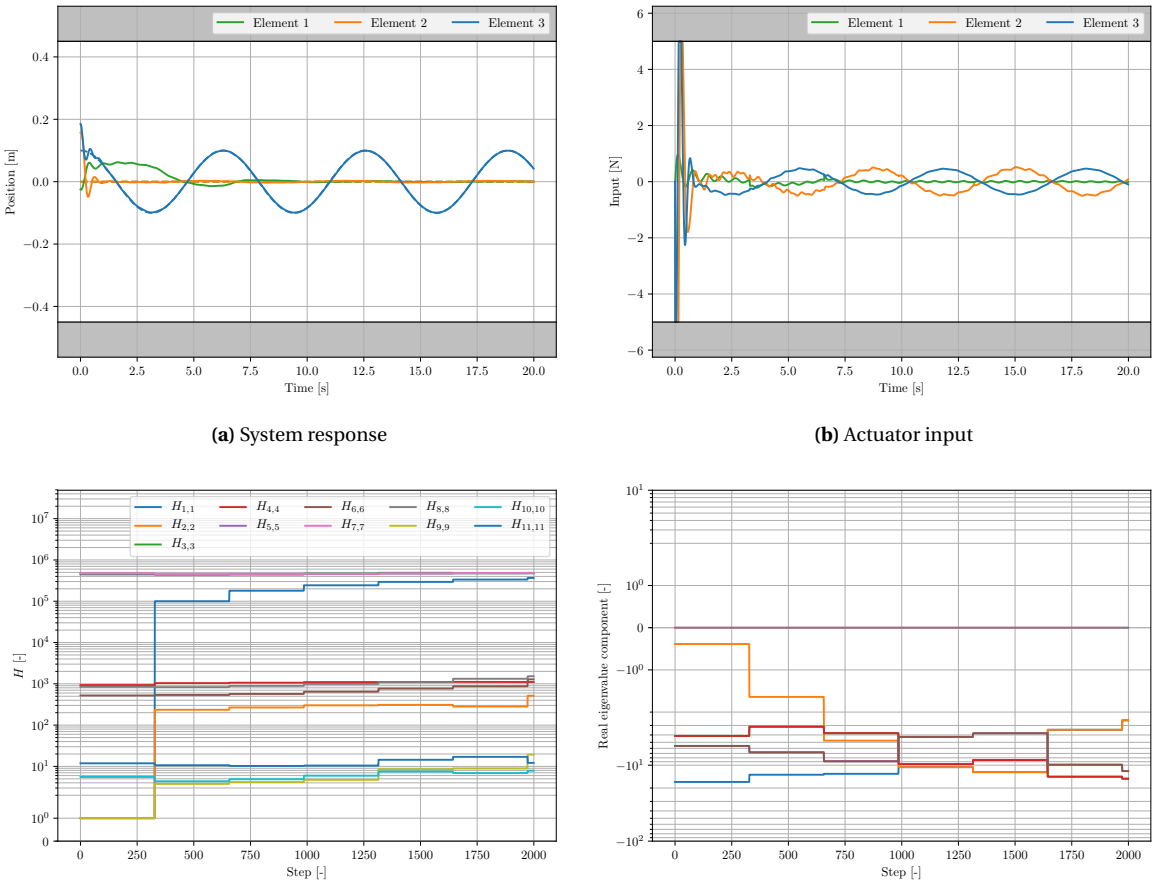


(b) Mapped kernel H_{target}



(c) Converged kernel H_{target}

Figure 7.8: Basic kernel mapping for the MSD-3 stage 3 tracking task as part of the inter-task learning curriculum; grayscale depth is based on logarithmic scale, equivalent levels of intensity implies same order of magnitude. Note that Figure (a) is equivalent to Figure 7.6c.



(a) System response

(b) Actuator input

(c) Kernel evolution (diagonal only)

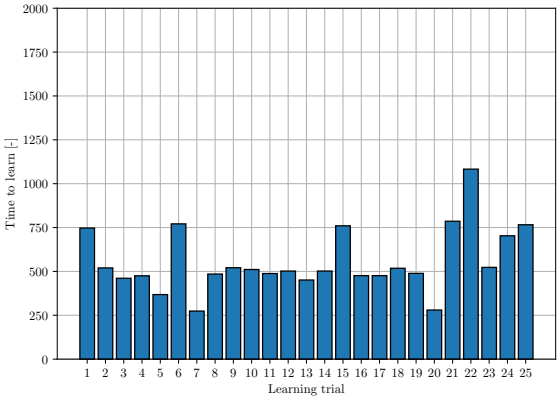
(d) Real eigenvalue components of closed-loop system

Figure 7.9: Example learning trial for the MSD-3 stage 3 (i.e., original) tracking task as part of the inter-task learning curriculum, with basic initial kernel derived from stage 2 (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

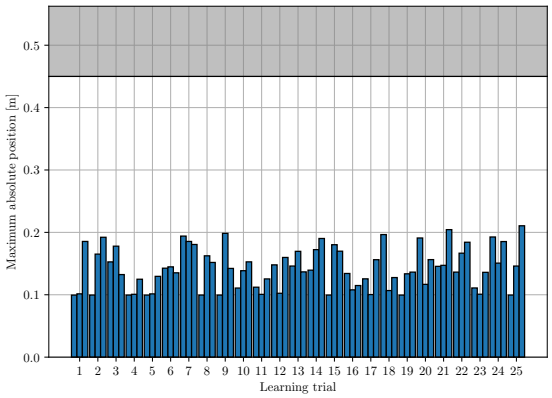
is reflected by the white regions in the figure. This basic strategy is sufficient for establishing a high-quality initial policy, as can be observed from Figure 7.7. Despite the fact that the MSD-2 system is inherently unstable, the closed-loop dynamics are stable even at the onset of learning. The tracking error for m_3 very quickly diminishes due to the fact that little learning is necessary to adjust the associated kernel elements to their optimal values. This is also reflected by Figure 7.6c, which shows that the majority of the kernel values for the known state-action space keeps the same order of magnitude. For regulation of m_2 however, more learning is required due to the bad initial estimate of the value function. Nevertheless, Figure 7.10d shows that all errors drop below the 5% RMSE threshold after about 10-15 seconds.

Subsequently, the same mapping strategy will be used to bias learning in the original MSD-3 tracking task. The relevant kernel transformations are presented in Figure 7.8, which shows a similar picture as for the previous transfer step. Figure 7.9 illustrates how learning progresses in this final stage. Very good control performance already at the onset of learning is again observed for the mapped elements, despite the unstable dynamics of the inner element. The initial policy is again of sufficient quality to stabilize the entire system, which implies that most time is spend in learning optimal regulation of the added mass. This is also reflected again by the converged kernel matrix shown in Figure 7.8c.

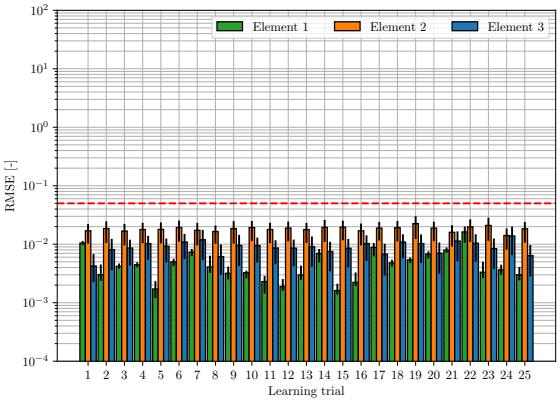
Figure 7.10 summarizes learning statistics over 25 learning trials using the basic inter-task curriculum strategy. The results in Figure 7.10b show that learning is consistently safe, as no crashes are recorded for any learning stage in any of the learning trials. This is in strong contrast with the flat learning case, which caused the system to crash almost consistently. On first sight however, little has been gained in terms of learning efficiency, as the cumulative total time to reach the 5 % RMSE limit in each stage is still in the range of 20-25 seconds. This is primarily due to the fact that no effort has been made to initialize the unknown parts of the



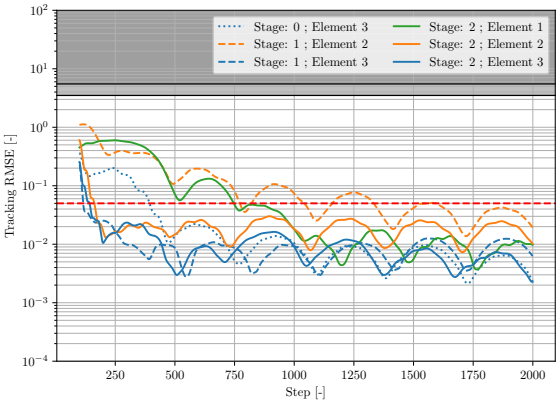
(a) Time-to-learn: learning stage 3



(b) Maximum position attained by element no. 3



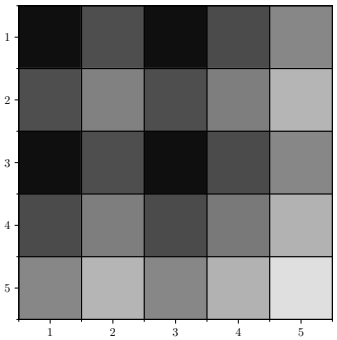
(c) RMSE at end of learning stage 3



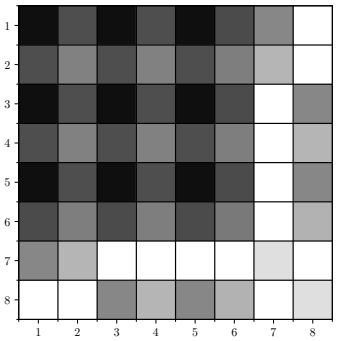
(d) RMSE evolution (example trial)

Figure 7.10: Learning performance statistics over 25 independent trial runs for the MSD-3 tracking task using inter-task curriculum learning with basic kernel mappings (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

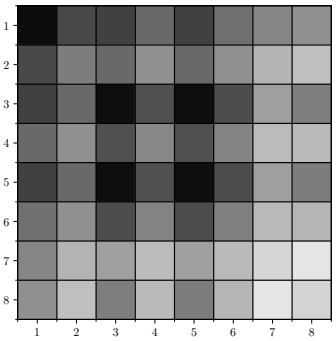
state-action space. Still, this time could be significantly reduced if the timing of transfer would be managed more strictly - that is, if transfer would take place before learning has converged in the intermediate stages.



(a) Source kernel H_{source}

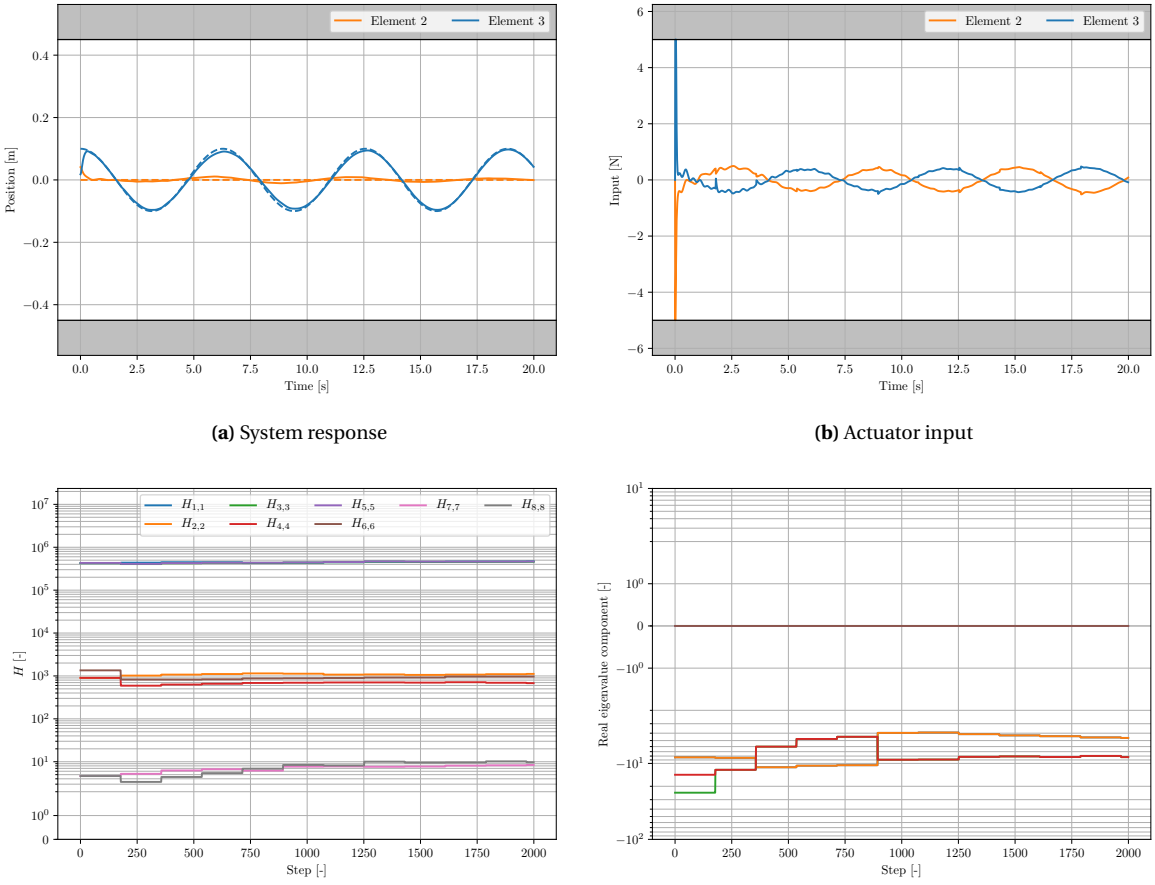


(b) Mapped kernel H_{target}



(c) Converged kernel H_{target}

Figure 7.11: Greedy kernel mapping for the MSD-3 stage 2 tracking task as part of the inter-task learning curriculum; grayscale depth is based on logarithmic scale, equivalent intensity levels implies same order of magnitude.



(a) System response

(b) Actuator input

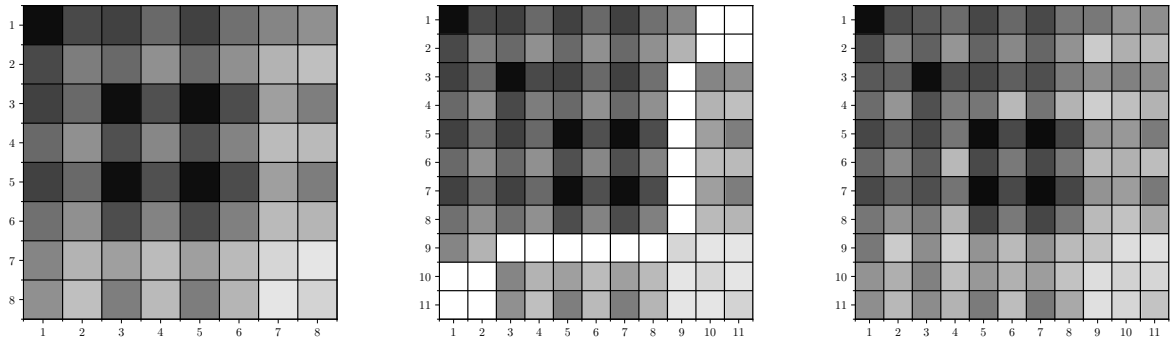
(c) Kernel evolution (diagonal only)

(d) Real eigenvalue components of closed-loop system

Figure 7.12: Example learning trial for the MSD-3 stage 2 tracking task as part of the inter-task learning curriculum, with greedy initial kernel derived from stage 1 (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

7.2.2. Greedy kernel mapping strategy

Based on these findings, it can now be investigated how the learning process would change if there is a form of nonzero kernel initialization for the unknown state-action space. One apparent approach would be to take

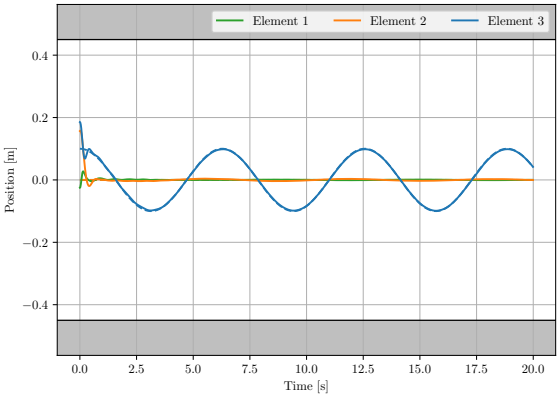


(a) Source kernel H_{source}

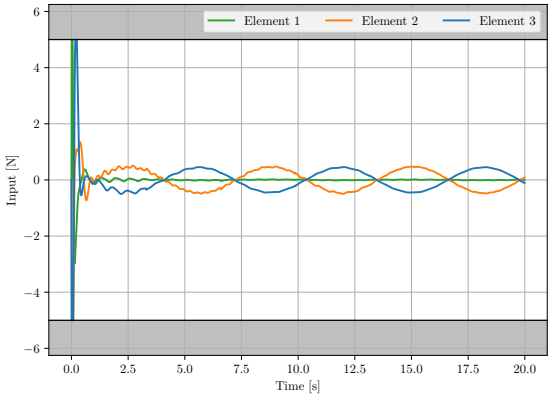
(b) Mapped kernel H_{target}

(c) Converged kernel H_{target}

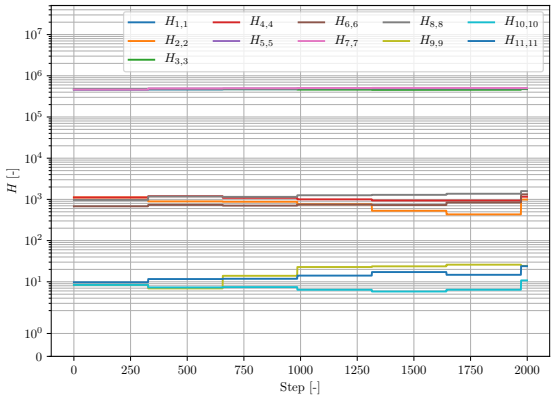
Figure 7.13: Greedy kernel mapping for the MSD-3 stage 3 tracking task as part of the inter-task learning curriculum; grayscale depth is based on logarithmic scale, equivalent levels of intensity implies same order of magnitude. Note that Figure (a) is equivalent to Figure 7.11c.



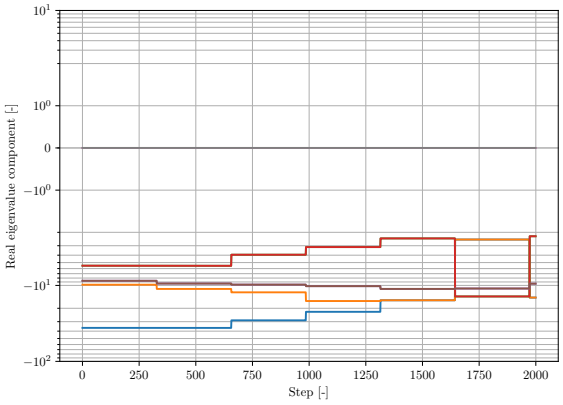
(a) System response



(b) Actuator input



(c) Kernel evolution (diagonal only)



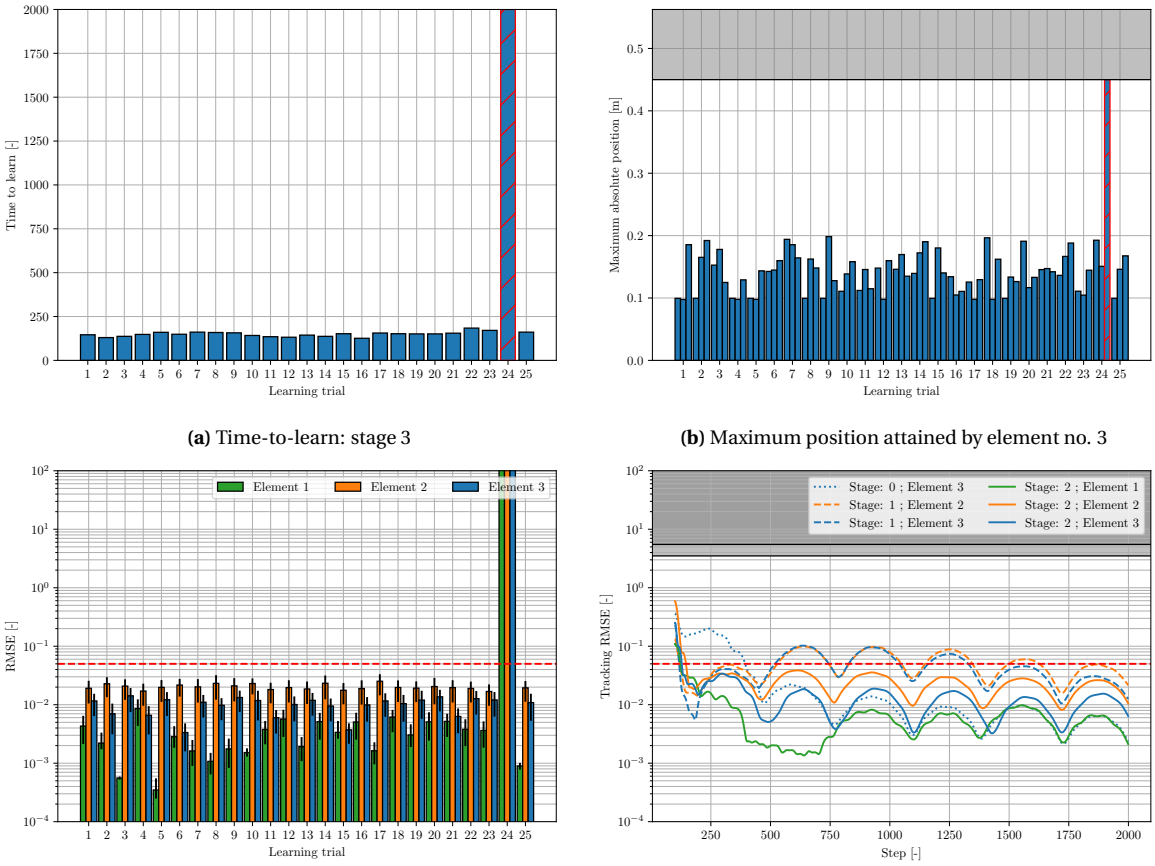
(d) Real eigenvalue components of closed-loop system

Figure 7.14: Example learning trial for the MSD-3 stage 3 tracking task as part of the inter-task learning curriculum, with greedy initial kernel derived from stage 2 (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

the kernel values corresponding to an element for which the dynamics are similar, and assign these to the new element. This will be referred to as the greedy strategy. The greedy mapping between stage 1 and 2 is visualized in Figure 7.11. Compared to the basic mapping strategy, the kernel values for m_2 are now initialized in the same way as m_3, including their couplings. Similarly, the 'local policy' terms between the added states and input have been duplicated from their stage 1 equivalents. This leads to a better initial local policy for the new element, which was still quite unsatisfactory in terms of performance with the basic mapping strategy. The resulting learning behavior is presented in Figure 7.12. Figures 7.13 and 7.14 show the same information for the final MSD-3 learning stage.

Compared to the basic mapping approach, these results illustrate that the learning process converges to the optimal control policy considerably faster under the greedy strategy. This is also reflected by Figure 7.15 where the statistics over 25 independent learning trials are summarized again. On average, the total time to learn is considerably shorter for the final stage, and the system remains stable throughout the curriculum. There is one notable exception however. In run 24, the first policy update in the final learning stage destabilizes the learning process and causes a system crash. Despite the fact that the initial mapped policy is stable in this case, the kernel values derived for the new state dimension appear to be invalid, which leads to an unsafe policy update. This shows that mapping greedily may be less reliable. Therefore, inappropriate initialization of the unknown state-action space may undo the stabilizing properties of curriculum learning.

The findings from this section show that an adequate inter-task learning curriculum is able to consistently stabilize the learning process. Generalizing knowledge to unknown parts of the state-action space generally has a positive impact on learning efficiency, but this comes at the expense of reliable stabilization properties. This result emphasizes again that curriculum learning alone cannot guarantee safety in general, and that



(a) Time-to-learn: stage 3

(b) Maximum position attained by element no. 3

(c) RMSE at end of learning stage 3

(d) RMSE evolution (example trial)

Figure 7.15: Learning performance statistics over 25 independent trial runs for the MSD-3 tracking task using inter-task curriculum learning with greedy kernel mappings (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

another safeguard mechanism is required to preserve ergodicity.

7.3. Intra-task Curriculum Learning

The inter-task curriculum strategy considered in the previous section demonstrated how a gradual buildup of learning complexity and transferring knowledge between subsequent stages can expedite learning in terms of safety and efficiency. However, since the system changes with every phase, it is not a very realistic example in the sense that in the physical world, the system cannot be taken apart. This section will therefore investigate intra-task staging, which does not allow the target system to change within the boundaries of every intermediate MDP. A visual overview of the intra-task learning curriculum is presented in Figure 7.16. Learning complexity can be managed in two ways in this curriculum. In the task domain, the possibilities are more limited compared to the inter-task example, as the dimensionality of the state space cannot be changed. However, the action space that is accessible to the agent can still be reduced in size, by putting a subset of the available actuators under the authority of an external supervisor. In the case study considered here, the actuators not available to the agent will be controlled by separate PD controllers. If the PD supervisors are able to regulate the non-controllable dynamics reasonably well, the closed-loop dynamics of the system essentially simulate the inter-task setting considered before. The other aspect of learning complexity then relates to the agent domain, where it can be chosen to either learn all dynamics from the start of learning, or observe only a given subset of the state-action space. However, this raises the question to what extent learning may be harmed by dynamics that are non-observable under the agent representation. Note that another discussion on mapping strategies will be avoided here.

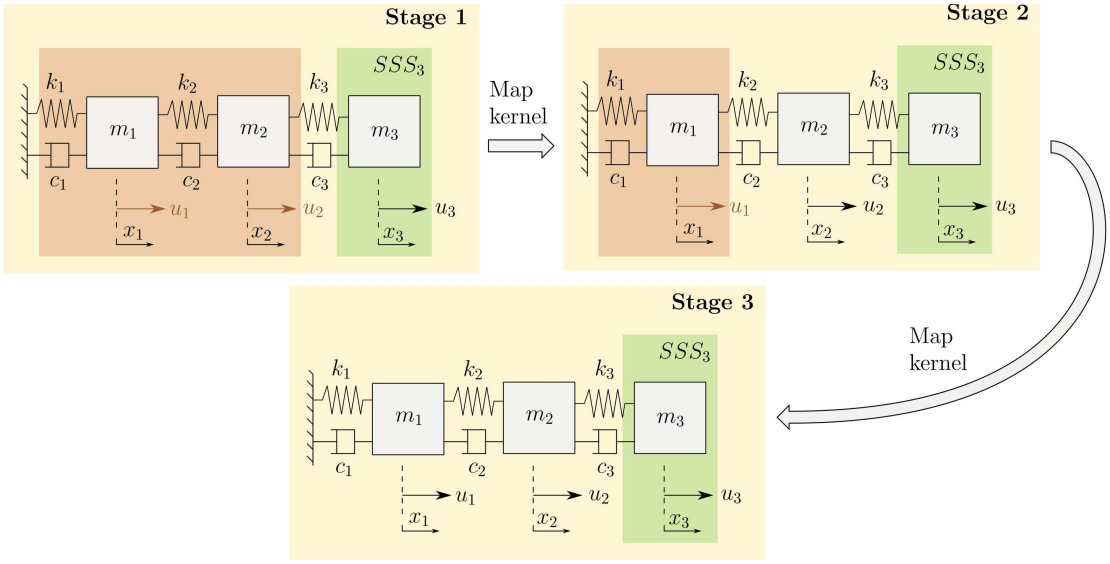


Figure 7.16: Intra-task curriculum setup for the cascaded mass-spring-damper (MSD-3) tracking experiment. Learning complexity can be adjusted in two ways: (1) by putting some actuators under authority of an external supervisor (indicated in brown); and (2) by limiting the internal agent representation to local dynamics only.

7.3.1. Global agent representation and effective assistance

First, the situation where the agent observes the full state space and the PD supervisors perform reasonably well in regulating the supervised nodes is considered. Learning performance during the first learning stage is presented in Figure 7.17. For this particular example, the proportional (P) and derivative (D) gains were set to 30 for node no. 1, and 60 for node no. 2, respectively. Figure 7.17b signifies the distinction between supervised and agent inputs by presenting the former as dash-dotted time traces. The results indicate that, although learning remains stable over the full length of the available learning time, the policy converges less quickly compared to stage 1 of the inter-task curriculum (Figure 7.5). The reason is that the batch size for the policy evaluation step is larger, since the full state space $\mathscr{X} \subset \mathbb{R}^6$ is observed. Another observation is that the kernel does not converge to the same values as seen before in Figures 7.9 and 7.14, which is a direct result of the fact that the agent learns to control the augmented system dynamics instead of the open-loop dynamics.

Since the policy still converges to the stage 1 optimal control, it is considered of sufficient quality to proceed with the next stage. Since the agent representation still changes across the curriculum (although less drastically than before), use shall be made again of kernel mappings. Figure 7.18 presents the mapping that is used to bias learning in stage 2. Since the dimensionality of every subsequent MDP only increases due to the added input channel, a particular simple mapping strategy can be adopted. Specifically, only one row and column need to be added, which implies that the majority of the kernel can remain intact. The principal mapping operation is then to directly clone the local policy terms for the new input channel from the known actuator, which is similar to the greedy inter-task mapping strategy presented before.

However, since the majority of the kernel remains unchanged, most state-related terms are slightly incorrect. The reason is that they refer to the augmented system dynamics from the previous stage, and not to the true system dynamics. Nevertheless, the learning process in stage 2 is stable and efficient, as can be observed from 7.19. The former cannot be said for the last learning stage, as shown in Figure 7.21. The kernel mapping for this last state, which is visualized in Figure 7.20, is again based on the closed-loop system dynamics from stage 2, in which the unstable stiffness of the inner-most node was actively compensated for by the PD supervisor. As a result, the initial policy is locally unstable, as is also reflected by the eigenvalue history shown in Figure 7.21d. However, it turns out that a single policy update is sufficient to correct for this. Finally, the kernel converges to the expected values at the end of learning.

Figure 7.22 again illustrates an overview of learning performance over 25 independent learning trials with random system initialization. In terms of safety, the same observation is made as for inter-task learning with greedy mapping: no crashes are observed for the majority of the trial runs, except for a single instance where the mapped kernel is insufficient to prevent the system from diverging. Learning efficiency is somewhat worse when considering the total training time over the curriculum, which can be largely attributed the larger

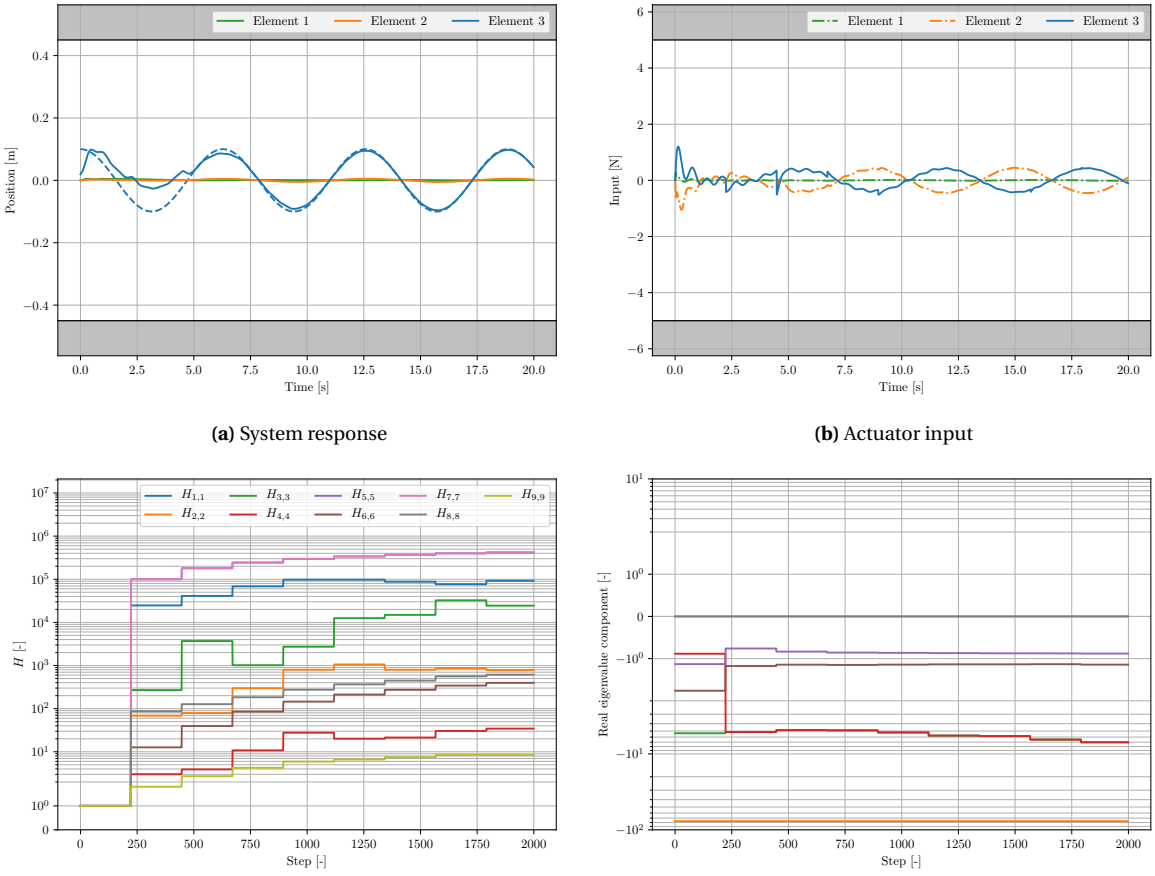
batch sizes in the first learning stages. Overall however, a significant improvement has been made again compared to learning from scratch.

7.3.2. Static agent representation and weak external assistance

The second case that will be considered in this discussion on intra-task learning curricula serves to investigate the effect of supervisor performance. The previous experiment assumed that the PD supervisors were tuned adequately to ensure strong regulation of the inner nodes. Here, a follow-up experiment will be performed where the supervisor performance is deliberately weakened. The proportional and derivative gains for element no. 1 and 2 will be considerably reduced by factors of 10 and 20, respectively. Consequently, the supervised nodes will suffer from larger deviations from their equilibrium points, which implies that uncontrollable dynamics are not as well hidden as before. However, the augmented system will still be stable. It is expected that weak PD regulation will have negative consequences for both learning safety as well as efficiency, since the control law will need to actively compensate for uncontrollable dynamics induced by the inner elements. This can also be considered a form of complex system dynamics.

The dynamic responses and corresponding inputs for each stage of this 'corrupted' version of the intra-task learning curriculum are shown in Figure 7.23. Surprisingly, learning is not very negatively affected in this example. Performance in the last learning stage has even improved, as the local instability of the initial policy is not visible here. A possible explanation is that the augmented dynamics in this case are closer to the system's natural dynamics, meaning that the mapped initial kernels are more relevant. When looking again at a larger distribution of learning runs however, the expected setbacks in terms of safety and learning efficiency are visible. The results over 25 independent learning trials have been summarized in 7.24.

However, these statistics may appear somewhat counterintuitive at first sight. All recorded system crashes occur in the last learning stage of the curriculum, despite the fact that the system is fully controllable in this



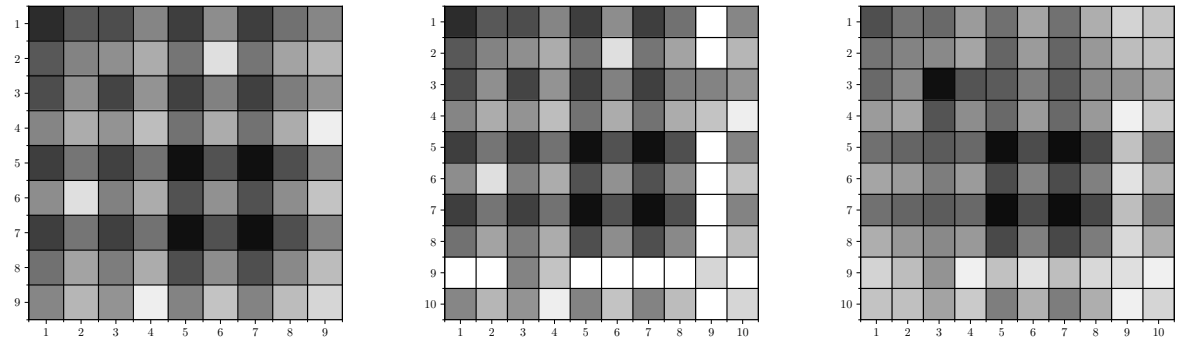
(a) System response

(b) Actuator input

(c) Kernel evolution (diagonal only)

(d) Real eigenvalue components for closed-loop system

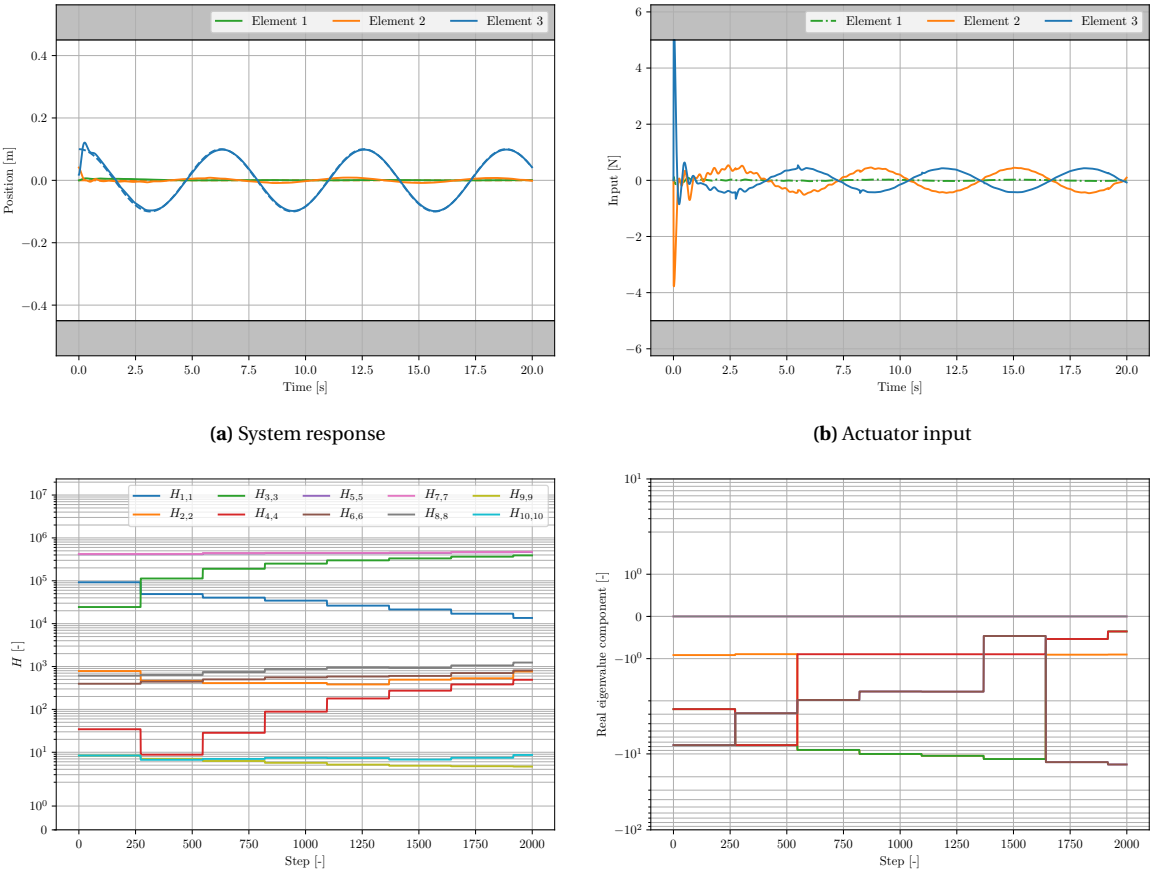
Figure 7.17: Example learning trial for the MSD-3 stage 1 tracking task as part of the intra-task learning curriculum with strongly regulating PD supervisors (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)



(a) Source kernel H_{source} **(b)** Mapped kernel H_{target} **(c)** Converged kernel H_{target}

Figure 7.18: Basic kernel mapping for the MSD-3 stage 2 tracking task as part of the intra-task learning curriculum with strongly regulating PD supervisors ; grayscale depth is based on logarithmic scale, equivalent intensity levels implies same order of magnitude.

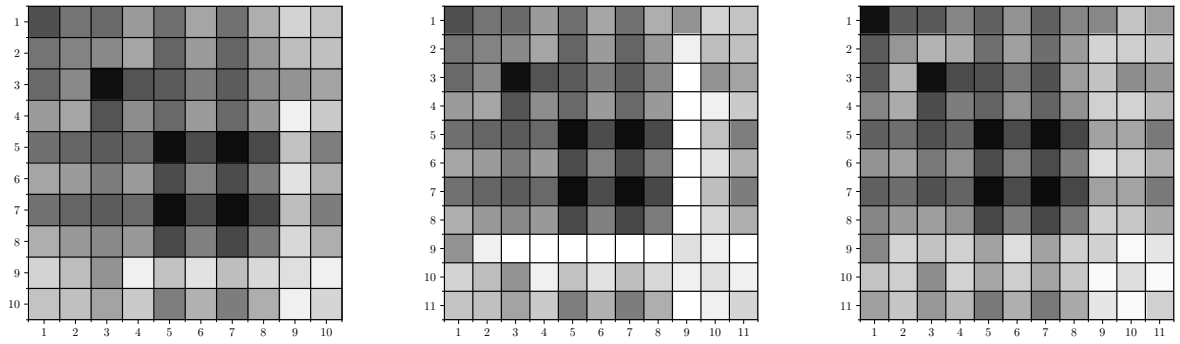
phase. It would be expected that learning is less reliable in the earlier phases, and that this would cause the system to crash prematurely. The opposite appears to be the case however, which is a direct effect of the kernel mapping strategy. Recall that for this intra-task curriculum, the kernel mapping is largely analogous to the greedy strategy presented in the previous Section. It appears that, since the agent has learned a con-



(a) System response **(b)** Actuator input

(c) Kernel evolution (diagonal only) **(d)** Real eigenvalue components of closed-loop system

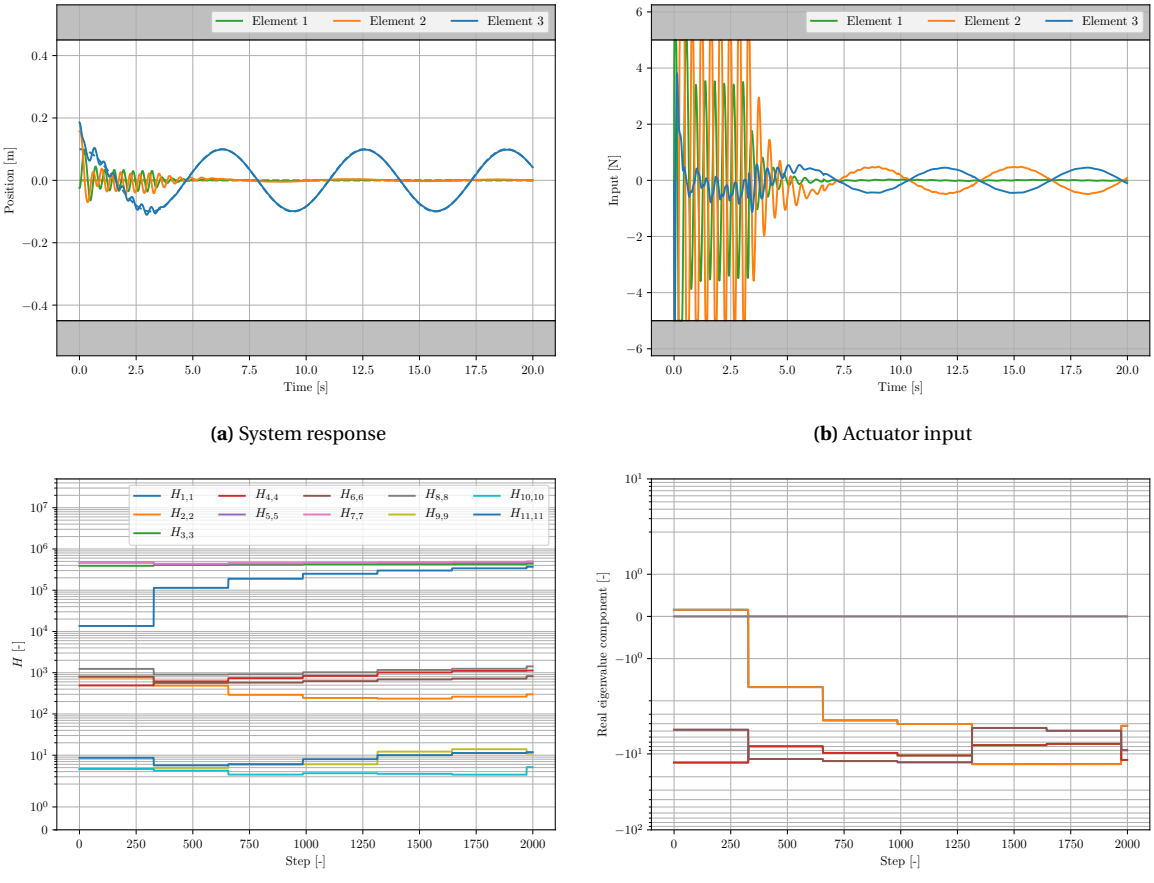
Figure 7.19: Example learning trial for the MSD-3 stage 2 tracking task as part of the intra-task learning curriculum with strongly regulating PD supervisors (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)



(a) Source kernel H_{source} **(b)** Mapped kernel H_{target} **(c)** Converged kernel H_{target}

Figure 7.20: Basic kernel mapping for the MSD-3 stage 3 tracking task as part of the intra-task learning curriculum with strongly regulating PD supervisors ; grayscale depth is based on logarithmic scale, equivalent intensity levels implies same order of magnitude. Note that Figure (a) is equivalent to Figure 7.18c.

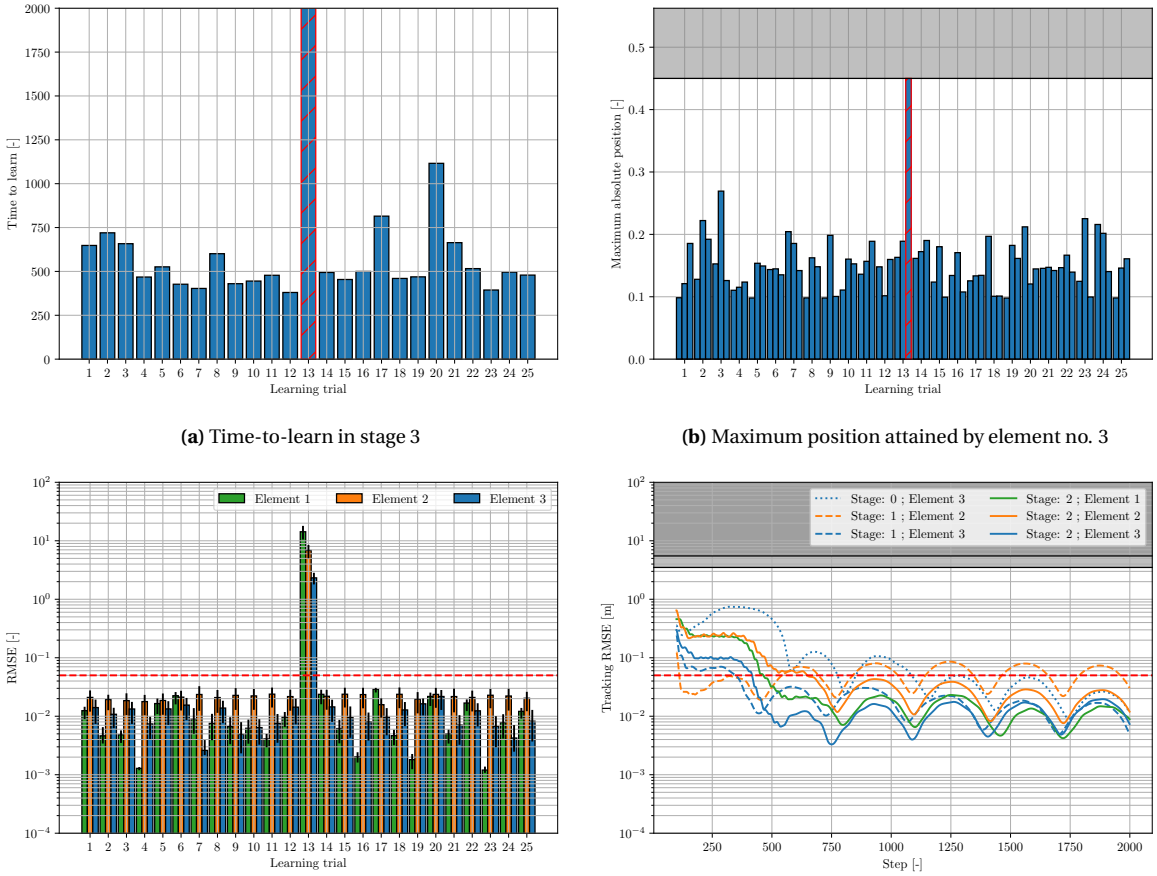
trol law that largely accommodates for the (uncontrollable) disturbing dynamics, the mapped policy is even more inappropriate than it maybe already is. This essentially amplifies the inherent unreliability of greedy mappings, which was the main conclusion from Subsection 7.2.2. Therefore, a weakly performing supervisor does have a detrimental effect, but its impact is strongly related to the mapping strategy.



(a) System response **(b)** Actuator input

(c) Kernel evolution (diagonal only) **(d)** Real eigenvalue components of closed-loop system

Figure 7.21: Example learning trial for the MSD-3 stage 3 tracking task as part of the intra-task learning curriculum with strongly regulating PD supervisors (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)



(a) Time-to-learn in stage 3

(b) Maximum position attained by element no. 3

(c) RMSE at end of learning stage 3

(d) RMSE evolution (example trial)

Figure 7.22: Learning performance statistics over 25 independent trial runs for the MSD-3 tracking task using intra-task curriculum learning with strongly regulating PD supervisors (*VI using Q-functions; batch LS policy evaluation; global agent representation; no safety mechanism; random init. with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

7.3.3. Flexible agent representation and effective assistance

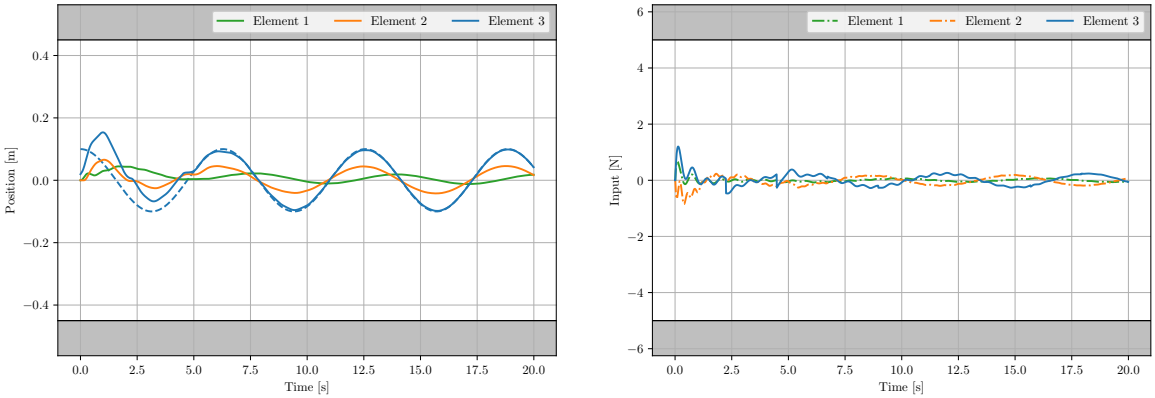
With this small investigation on the role of the task domain, the focus in our third experiment will be on the agent domain. In the previous case studies, the complete state space was observed from the very start of the learning curriculum. However, the taxonomy of curriculum learning says that learning complexity can be reduced by means of an appropriate agent representation, which can be of good value for intra-task curriculum learning for the MSD-3 tracking task. In this experiment, only those states that are associated with the elements controlled by the agent will be included in the intermediate kernel matrices. For the first stage, this means that x_3 and $\dot{x}_3$ are the only system states that are included in the agent's internal representation. Consequently, a major part of the system dynamics will be non-observable in the early stages of the learning curriculum. Note that no self-paced learning techniques will be considered here. Note that by structuring the intra-task learning curriculum in this way, the learning process becomes in fact quite similar to the inter-task curricula discussed in the previous Section.

Figure 7.25 shows learning during the first stage under this flexible agent representation, using the well-tuned PD supervisor architecture used before. Due to the good performance of the PD controllers, the non-observable dynamic effects are kept to a minimum here. Consequently, the agent is well capable of learning the optimal control policy, despite its limited ability to build a good internal model of the dynamics. Note that the learning evolution is remarkably similar to the inter-task setting with the same initial condition (see Figure 7.9). Compared to the previous inter-task experiments, learning time is considerably reduced due to the smaller batch sizes required for the policy evaluation step.

For the next learning stages, the basic mapping strategy introduced for inter-task learning will be used again. The resulting kernel mappings and learning profiles are visualized in Figures 7.26-7.29 for the remain-

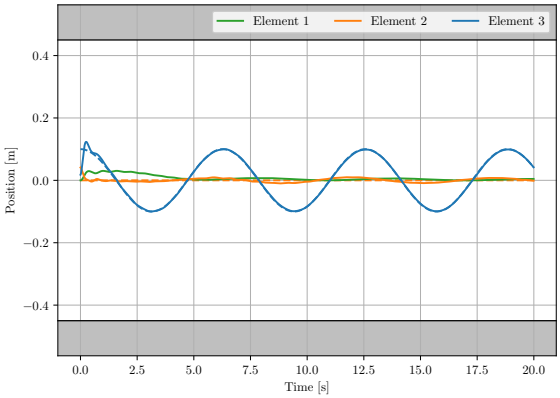
ing curriculum stages. The similarities with basic mapping inter-task learning are again evident, which is further emphasized by the equivalent kernel evolution patterns.

Figure 7.30 provides again a summary over 25 independent learning trials. Each curriculum results in crash-free learning, which is consistent with the findings that were made for inter-task curriculum learning with the same basic mapping strategy. Note that because of the mapping strategy, the sporadic crashing behavior that was seen in Subsection 7.3.1 is not observed here. One slightly negative difference however is the larger variability in the time-to-learn metric, which is likely to be a direct result of non-observable

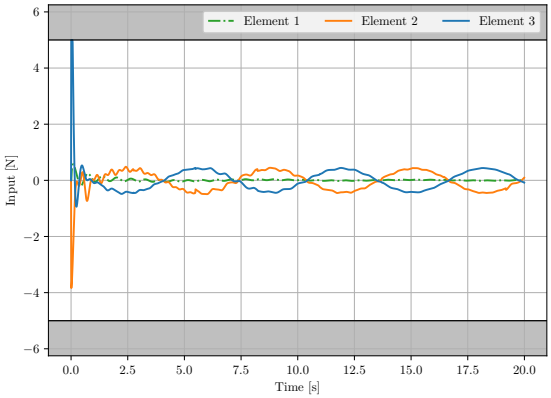


(a) System response during stage 1

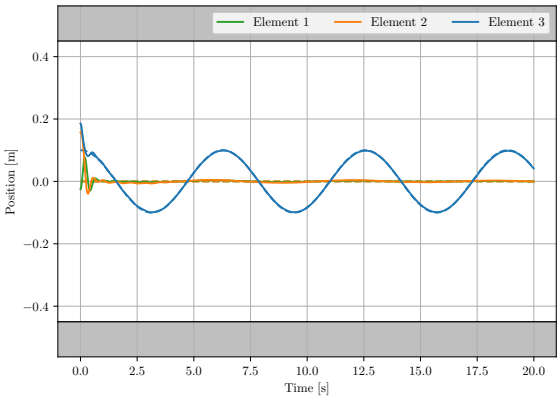
(b) Actuator inputs during stage 1



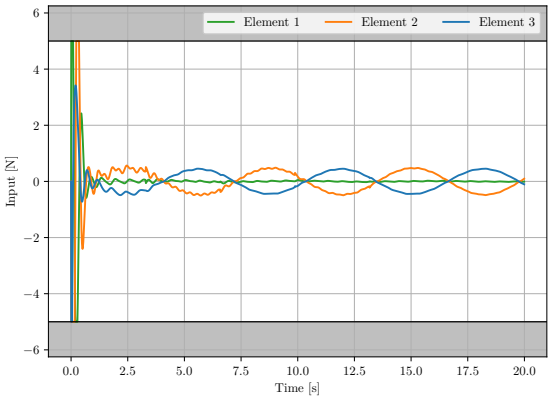
(c) System response during stage 2



(d) Actuator inputs during stage 2



(e) System response during stage 3



(f) Actuator inputs during stage 1

Figure 7.23: Example learning trial for the MSD-3 tracking task as part of the intra-task learning curriculum with weakly regulating PD supervisors (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

dynamics.

7.3.4. Flexible agent representation and weak assistance

Finally, the performance of the PD supervisors will again be considered in view of local agent representation. It is expected that bad regulation performance of the non-controlled and non-observable dynamics will pose a particularly large problem for learning in this setting. Therefore, it is interesting to investigate how the agent will behave during the learning curriculum when the same badly tuned PD settings considered before are used. The system response and actuator inputs for every stage of this 'corrupted' curriculum are shown in Figure 7.31. The first learning stage confirms our expectation, as the first policy update immediately destabilizes the system. Hence, this quantifies as unsafe learning behavior. Surprisingly however, a stabilizing control law is found in the next update - that is, even before the system crashes -, and the agent converges to an optimal tracking control law for the element no. 3. This is a remarkable result, as learning suffers from large observability and controllability issues, meaning that the task is essentially non-Markovian.

Subsequently, the learning curriculum proceeds with stage 2, for which the same basic kernel mapping strategy is used as before. Since learning cannot be harmed by a bad initialization of the added state-action dimensions with this strategy, the system remains stable during the entire process, and the agent converges again to the optimal tracking policy. Finally, the original MSD-3 learning task is considered again in stage 3. Note that the learning profile is again largely equivalent as the inter-task curriculum example from the previous section.

Finally, Figure 7.32 summarizes learning performance again over 25 independent learning trials. Similarly as for the case where there the agent was equipped with a global representation, an increased crashing

(a) Time-to-learn in stage 3

(b) Maximum position attained by element no. 3

(c) RMSE at end of learning stage 3

(d) RMSE evolution (example trial)

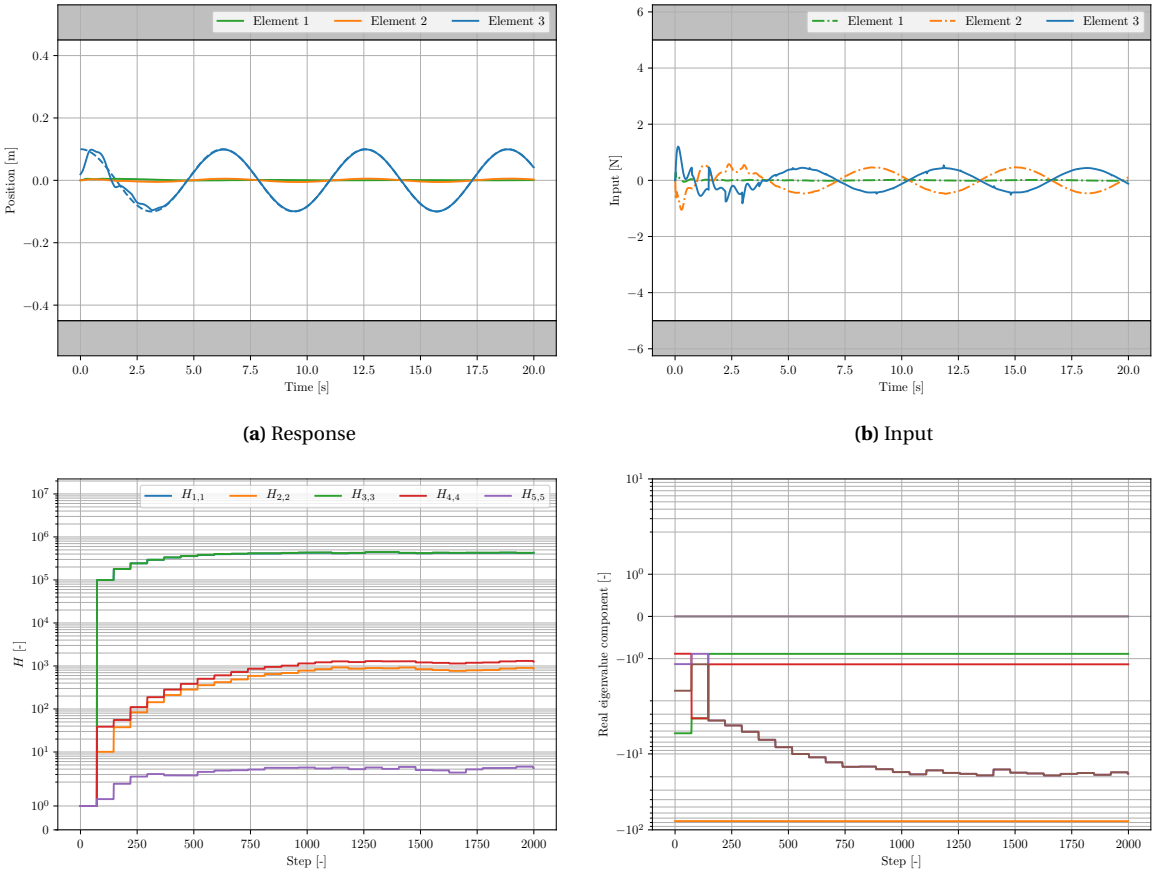
Figure 7.24: Learning performance statistics over 25 independent trial runs for the MSD-3 tracking task using intra-task curriculum learning with weakly regulating PD supervisors (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

frequency is observed as a consequence of the weak PD assistance. An important difference however is that the system crashes consistently in the first learning trial, whereas before this was the last learning trial. The expected setback in performance is now a direct result of poor observability and controllability properties at the onset of the learning curriculum, and not an indirect effect of an inadequate kernel mapping. Therefore, it can be concluded that limited agent representation and weak supervisor performance have a reinforcing negative effect on learning safety.

Overall, this section has shown that intra-task curriculum learning is an effective option to make online adaptive optimal control considerably safer and more efficient. It has been investigated how such a curriculum behaves under different task and agent domain characteristics. It appears that for this particular LQT problem, these aspects are quite forgiving, in the sense that the learning process will still be stable and converging towards the optimal control policy for a wide range of seemingly sub-optimal learning conditions.

7.4. Safe Learning

So far, it has been demonstrated that well-designed learning curricula can lead to safety through stability, which in the context of the MSD system implies that ergodicity is preserved implicitly. However, there have also been multiple instances where learning leads to unstable policy updates, which almost certainly causes the system to crash. As argued in Chapter 6, truly safe learning requires a safety mechanism that guarantees ergodicity at every step. For this reason, two examples of safe learning techniques from literature are investigated for the MSD-3 tracking task in this section. The first case study focuses on the SHERPA algorithm [65, 66, 70], which is based on a heuristic online backup identification approach and was discussed in Section 5.2.1. In theory, this algorithm will ensure that exploration can continue indefinitely if a backup control sequence can be found for every time step. As will be seen however, finding backups becomes a highly



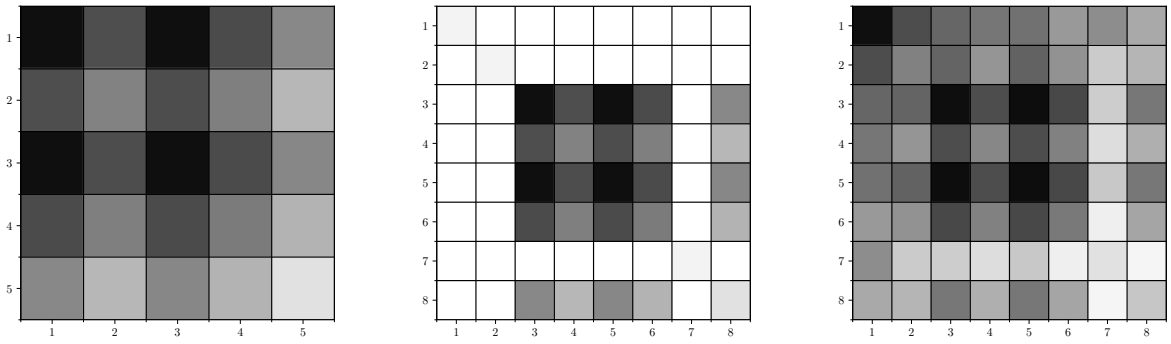
(a) Response

(b) Input

(c) Kernel evolution (diagonal only)

(d) Closed-loop real eigenvalue components

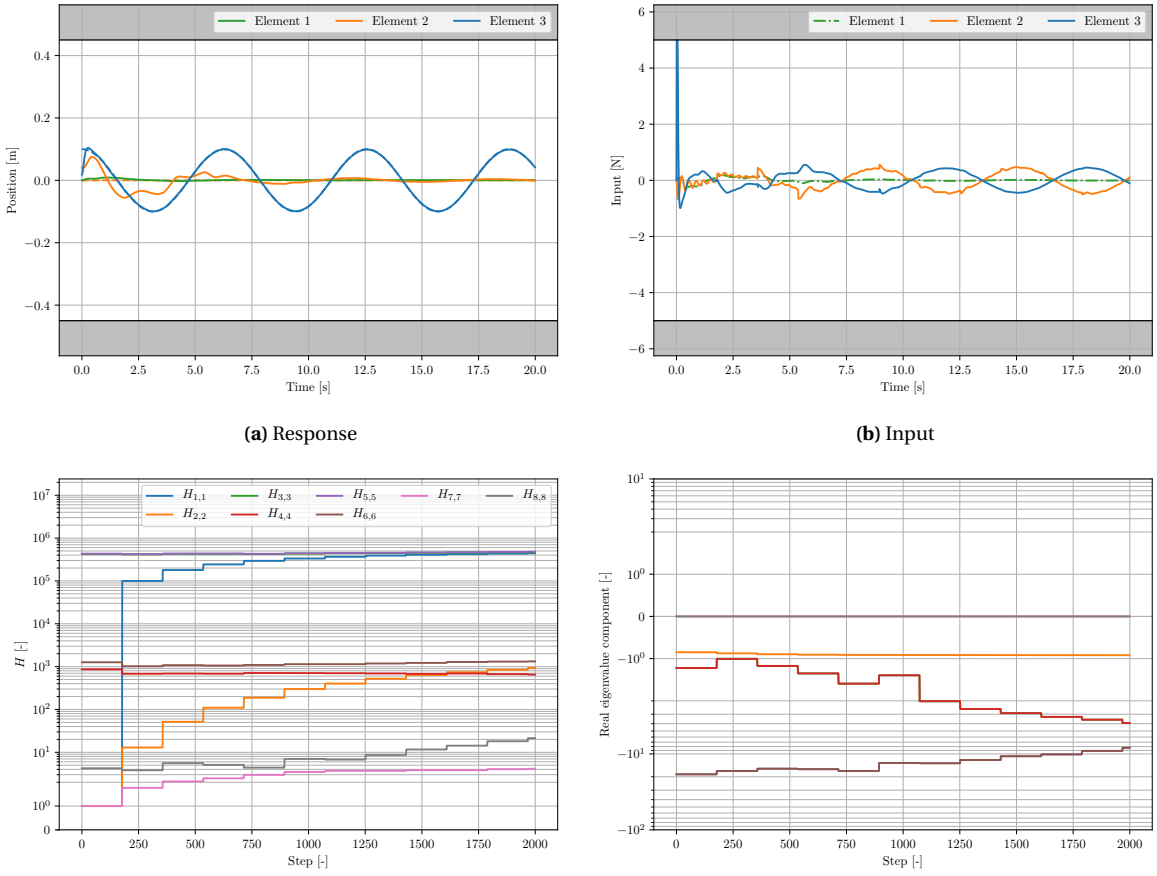
Figure 7.25: Example learning trial for the MSD-3 stage 1 tracking task as part of the intra-task learning curriculum with strongly regulating PD supervisors and local agent representation (*value iteration using Q-functions; batch LS policy evaluation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)



(a) Source kernel H_{source} **(b)** Mapped kernel H_{target} **(c)** Converged kernel H_{target}

Figure 7.26: Basic kernel mapping for the MSD-3 stage 2 tracking task as part of the intra-task learning curriculum with strongly regulating PD supervisors and local agent representation; grayscale depth is based on logarithmic scale, equivalent intensity levels implies same order of magnitude.

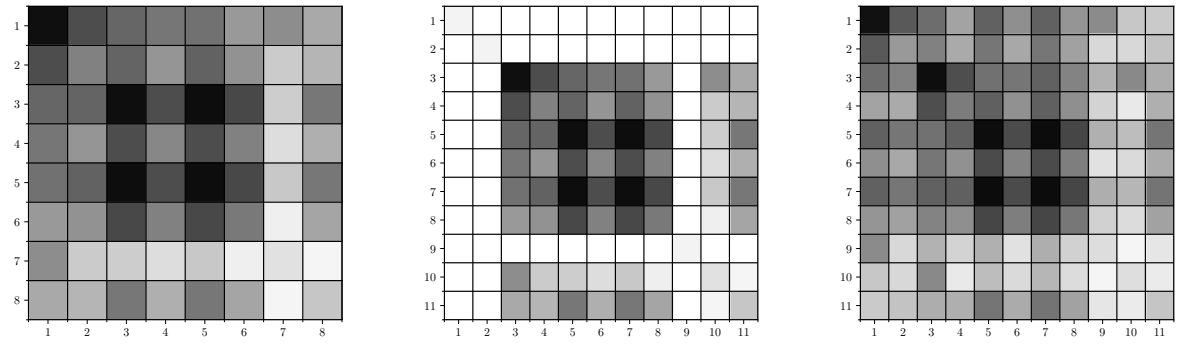
tedious process in situations where the state-action space is high-dimensional and the system dynamics include multiple levels of integration. The successor algorithm, optiSHERPA [65, 70], aims to mitigate these problems, and will therefore be studied in a second experiment. The key concepts behind optiSHERPA were discussed in Subsection 5.2.2. Note that optiSHERPA does not provide any theoretical guarantees in terms of



(a) Response **(b)** Input

(c) Kernel evolution (diagonal only) **(d)** Closed-loop real eigenvalue components

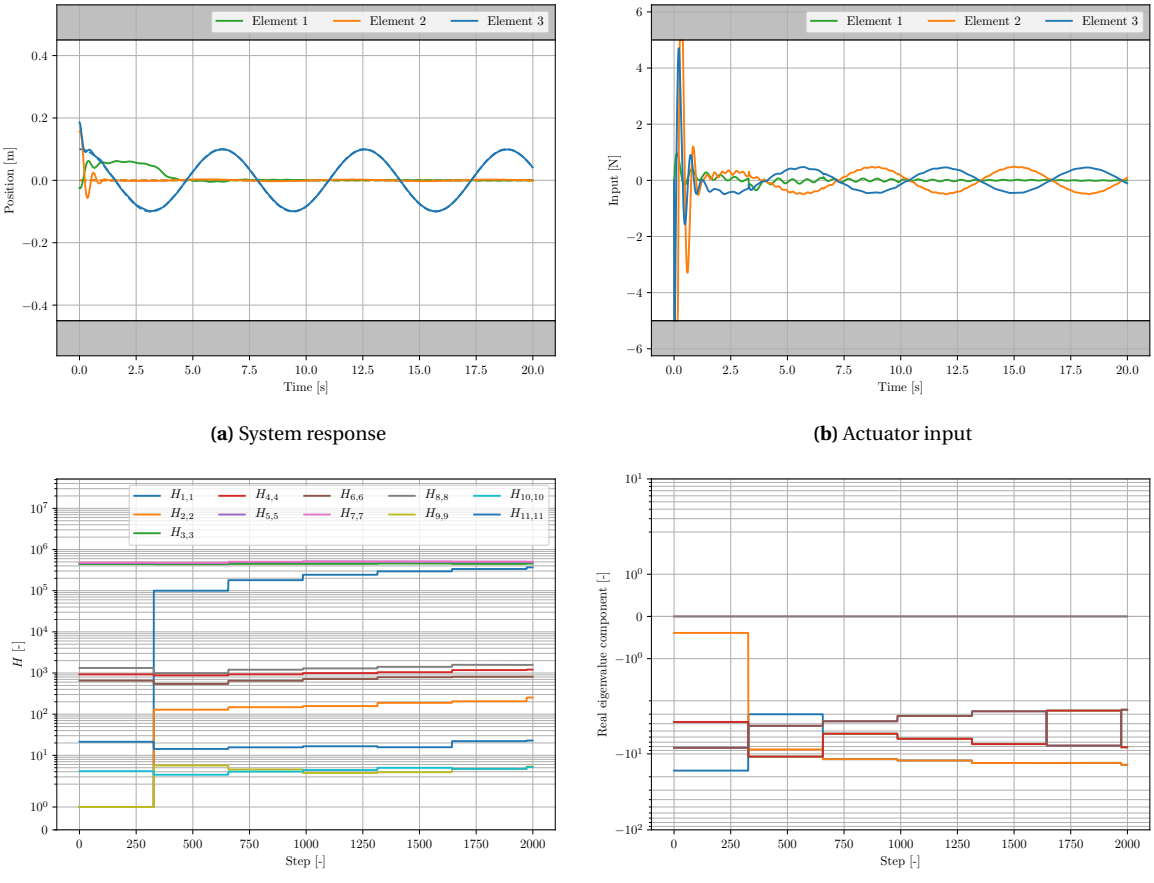
Figure 7.27: Example learning trial for the MSD-3 stage 2 tracking task as part of the intra-task learning curriculum with strongly regulating PD supervisors and local agent representation (*value iteration using Q-functions; batch LS policy evaluation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)



(a) Source kernel H_{source} **(b)** Mapped kernel H_{target} **(c)** Converged kernel H_{target}

Figure 7.28: Basic kernel mapping for the MSD-3 stage 3 tracking task as part of the intra-task learning curriculum with strongly regulating PD supervisors and local agent representation; grayscale depth is based on logarithmic scale, equivalent intensity levels implies same order of magnitude. Note that Figure (a) is equivalent to Figure 7.26c.

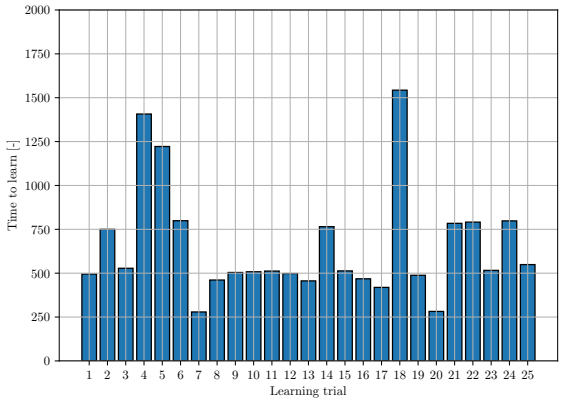
preserving ergodicity. However, it forms an interesting benchmark as a practical safe learning mechanism.



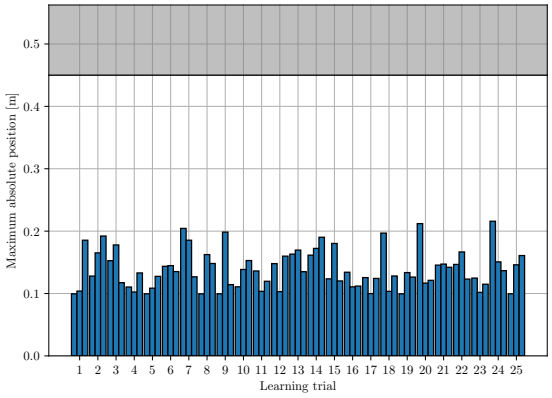
(a) System response **(b)** Actuator input

(c) Kernel evolution (diagonal only) **(d)** Real eigenvalue components of closed-loop system

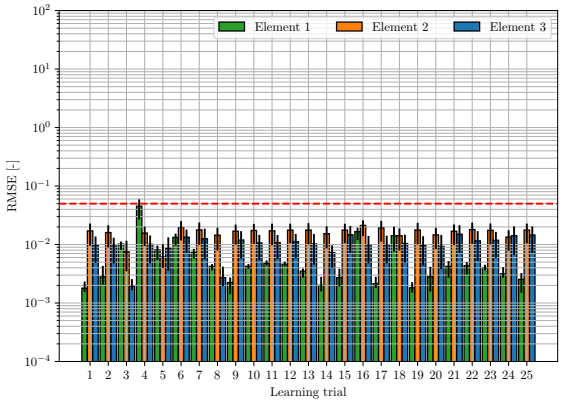
Figure 7.29: Example learning trial for the MSD-3 stage 3 tracking task as part of the intra-task learning curriculum with strongly regulating PD supervisors and local agent representation (*value iteration using Q-functions; batch LS policy evaluation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)



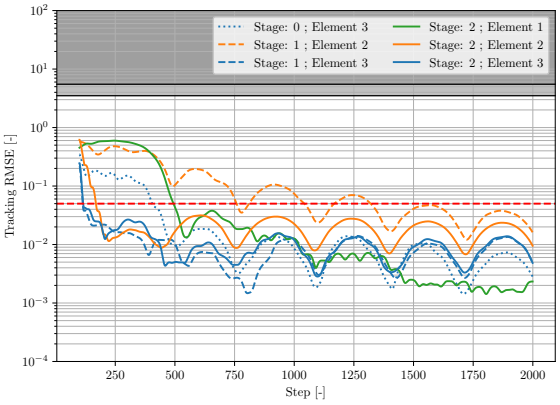
(a) Time-to-learn (target stage)



(b) Maximum position attained by element no. 3



(c) Performance at end of learning (stage 3)



(d) RMSE over 1s window (illustrative trial)

Figure 7.30: Learning performance statistics over 25 independent trial runs for the MSD-3 tracking task using intra-task curriculum learning with strongly regulating PD supervisors and local agent representation (*value iteration using Q-functions; batch LS policy evaluation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

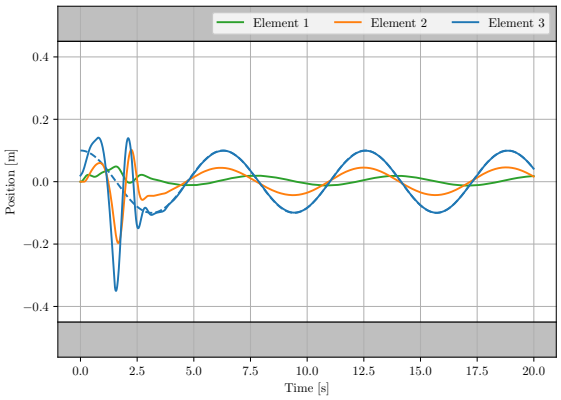
7.4.1. Heuristic Backup Identification using SHERPA

SHERPA uses an over-approximation of the true system dynamics in the form of a bounding model to ensure the existence of safe return trajectories. Although the dynamics of the MSD-3 system are fully known by virtue of Table 7.1, such a bounding model will also be generated for this experiment to mimic the common situation where the dynamics are uncertain. Since the dynamics are fully linear, the following model applies:

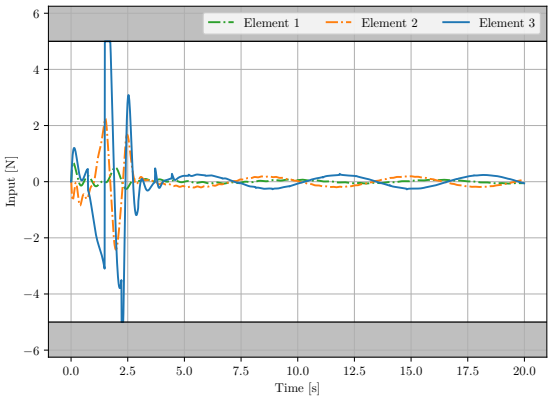
$$\boldsymbol{\Delta}(\boldsymbol{x}_t, \boldsymbol{u}_t) = [A]\boldsymbol{x}_t + [B]\boldsymbol{u}_t \tag{7.1}$$

where $[A]_{ij} = \left[(1-\kappa)A_{ij}, (1+\kappa)A_{ij}\right]$, $[B]_{ij} = \left[(1-\kappa)B_{ij}, (1+\kappa)B_{ij}\right]$, and κ is sampled from a uniform distribution $[0, \kappa_{max}]$. In this way, the true linear dynamics are always captured by a random bounding model. This model can be efficiently propagated forward in time by means of interval arithmetic.

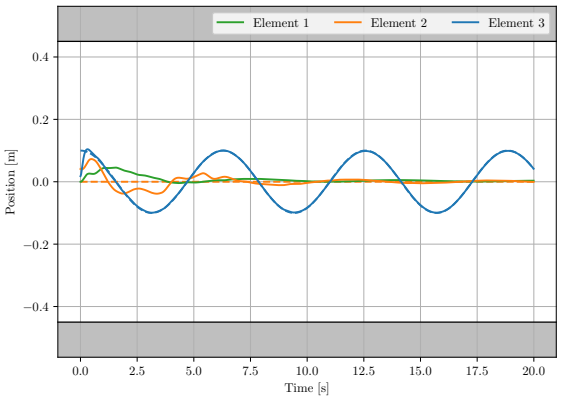
Next, a set of heuristics needs to be defined to enable practical operation of the algorithm. First, it is assumed that the outer element m_3 is equipped with distance sensors, which enables SHERPA to detect when the restricted fatal state space $RFSS = RSS \cap \{\mathscr{X} \setminus SSS_3\}$ is in reach. This sensing capability forms SHERPA's key feature for perceiving risks and estimating the true SSS online, which is assumed to be unknown a-priori. The distance sensor is assumed to have a range of 0.2 m. Second, adequate values for the backup closeness conditions need to be determined. In this experiment, only the reaching interval $[\boldsymbol{\epsilon}]$ will be of relevance, since the system only has a single equilibrium state that can be straightforwardly added to the set of already visited states. Selecting adequate values for $[\epsilon]_i = [-\epsilon_i^{max}, \epsilon_i^{max}]$ involves a trade-off between backup reach and backup quality: selecting the reaching interval too narrow results in very low chances that a backup can be found within a limited number of iterations, whereas selecting it too wide will result in backups that drive the



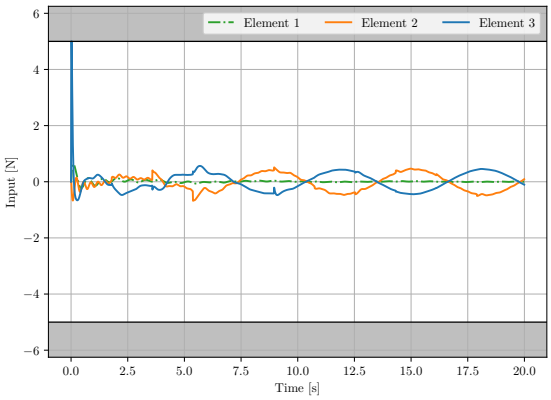
(a) System response during stage 1



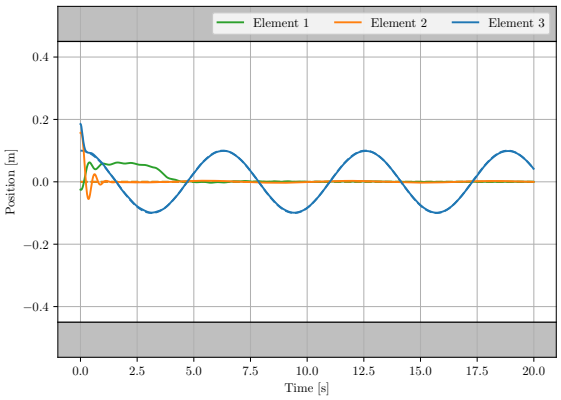
(b) Actuator inputs during stage 1



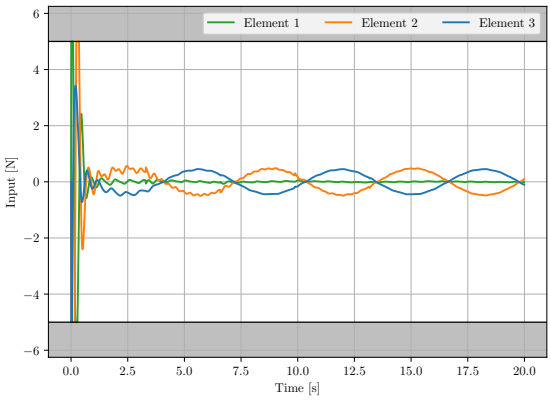
(c) System response during stage 2



(d) Actuator inputs during stage 2



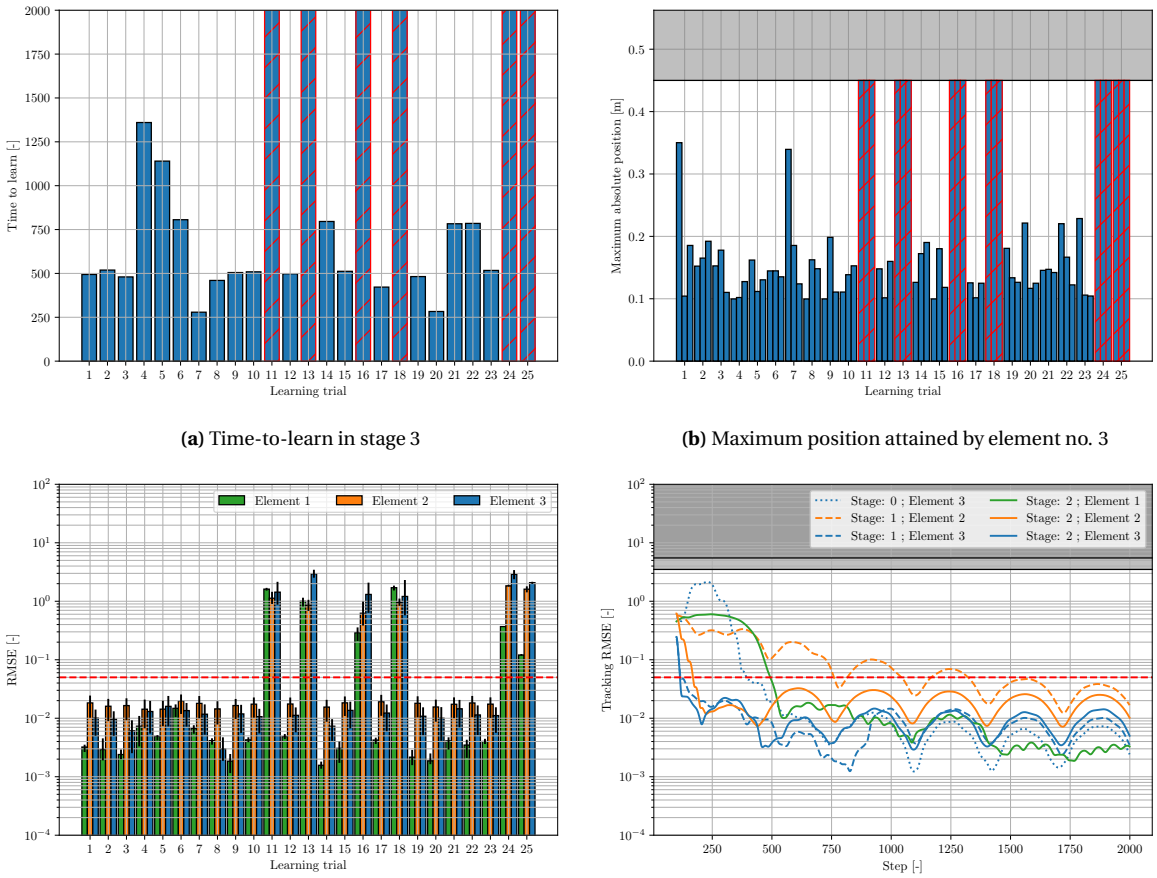
(e) System response during stage 3



(f) Actuator inputs during stage 3

Figure 7.31: Example learning trial for the MSD-3 tracking task as part of the intra-task learning curriculum with weakly regulating PD supervisors and local agent representation (*value iteration using Q-functions; batch LS policy evaluation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

system too far from already explored states. In this experiment, values of $\epsilon_{x_i}^{max} = 0.1$ m and $\epsilon_{\dot{x}_i}^{max} = 0.1$ m/s will be used. Third, decisions must be made on the maximum number of alternative inputs SHERPA may propose before the iteration loop terminates, as well as on the total amount of backup iterations available within every input iteration (recall SHERPA's algorithmic logic, visualized in Figure 5.2). Since backup calculations are essentially nothing more than a random search, a sufficient number of iterations must be available to ensure effective operation of the algorithm. On the other hand however, if a too large number of iterations is allowed



(a) Time-to-learn in stage 3

(b) Maximum position attained by element no. 3

(c) RMSE at end of learning stage 3

(d) RMSE evolution (example trial)

Figure 7.32: Learning performance statistics over 25 independent trial runs for the MSD-3 tracking task using intra-task curriculum learning with weakly regulating PD supervisors and local agent representation (*value iteration using Q-functions; batch LS policy evaluation; no safety mechanism; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

for, the computation complexity will become unacceptably high, especially for real-time[2] operations. Here, the total number of input and backup iterations have been limited to 10 and 30, respectively. These values still result in high computational complexity, but have only been selected for the sake of demonstration. In addition, the maximum length of the backup sequence will be limited to 60 time steps.

Finally, some additional settings will be introduced to increase SHERPA's effectiveness. First, instead of generating a random input for every time step, the backup sequence will consist of a set of input 'bands' that will keep the backup control constant over a multiple number of time steps. In this way, the backup will consist of a meaningful control sequence, rather than some discrete noise signal. In the current example, these bands will have a width of 20 time steps, resulting in a control backup sequence that is three blocks wide at most. Second, there is the dimensionality of the action space. It is expected that the search for backups is most tedious in those situations where the system state evolves towards the RFSS, which often requires a large control input to get away from. If the cardinality of the input space is large, however, each input has an extremely small probability of being selected. Therefore, only a discretized form of the total input space will be made available to SHERPA, which in this case is defined as $\hat{\mathcal{U}}_i = \{-5, -2.5, 0, 2.5, 5\}$ N. Moreover, the observed state will only be stored once every 10 time steps, to further expedite the backup search process.

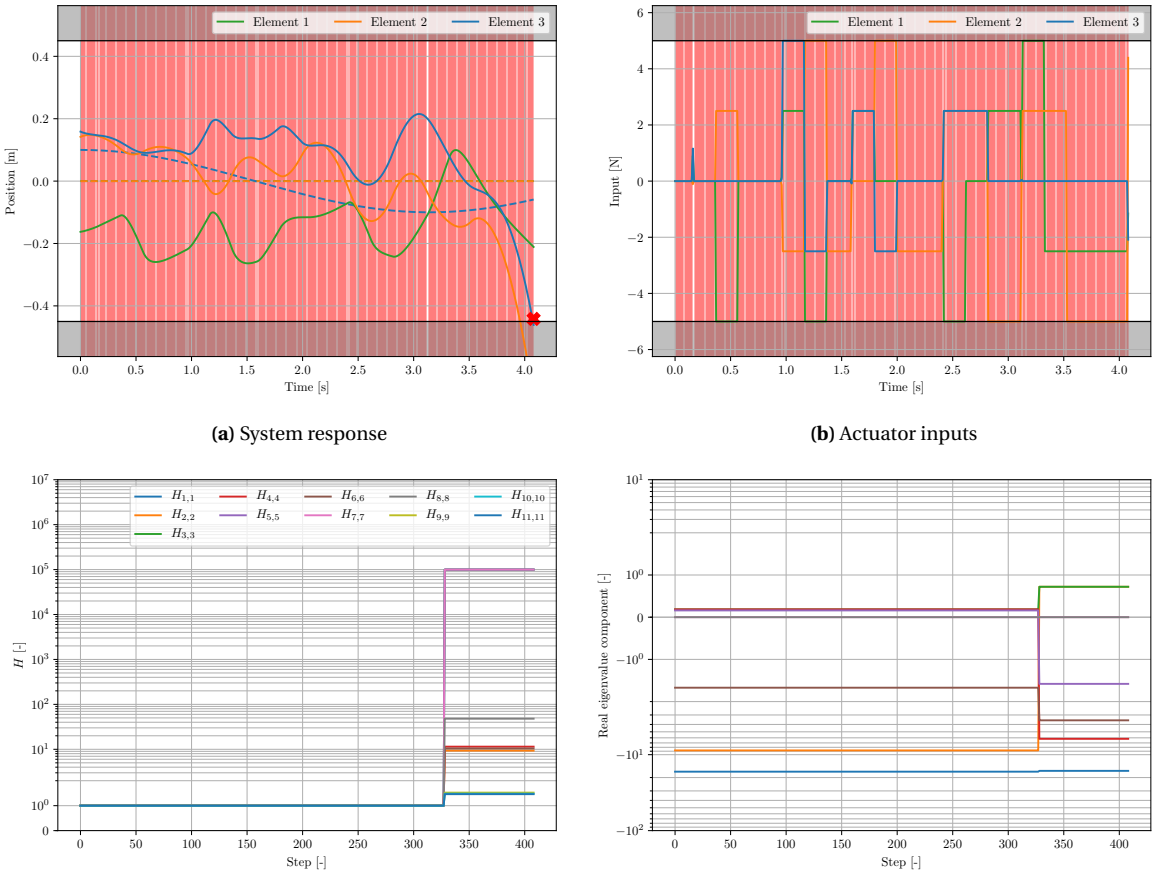
This is all that is required to add SHERPA to the original LQT Q-learning framework. The only modification to the existing learning architecture is an input validation step by SHERPA that ensures whether ergodicity will be preserved under the input proposed by the agent.

Figure 7.33 shows a typical learning trial for the MSD-3 system under supervision by SHERPA. For this par-

[2] In this experiment, computations will not be constrained to real-time.

ticular example, it was assumed that the bounding model is in fact a crisp model, meaning that $\epsilon_i^{max} = 0$ for all states. This provides for an upper benchmark in terms of performance, since the space of feasible backups is large when the uncertainty is small. However, in the case of the MSD-3 system, even with full prediction capabilities SHERPA is not able to always identify a feasible backup. Figure 7.33a reflects this in two ways. First, a red region indicates that SHERPA is performing a backup. Considering the time history shown here, it becomes clear that SHERPA is taking backups throughout the entire learning trial. However, these are not triggered by perceived risk, but simply by the fact that SHERPA depletes all its available iterations for finding a backup at a given state. If this situation occurs, the algorithm has no better option than taking a backup, which approximately restores the system to a state that has been visited before. Hence, in this example, the discovery of a backup is more exception than rule, despite the relatively large number of iterations available. A second example of failed backup computation is the last phase before the system crashes. Normally, once SHERPA initiates a backup, it already starts computing a new backup for the expected arrival interval. Given that a backup lasts 60 time steps in this case, this implies that the total number of backups that can be computed equals $60 \times 30 \times 10 = 18,000$. Often, the algorithm is able to find at least one backup within this window. For the last phase however, this is not the case, meaning that ergodicity is no longer preserved. Consequently, SHERPA has no option other than just taking some control input, which in this case implies that the last input is repeated. However, this brings the system inadvertently in an LTF state, and ultimately in the RFSS. Hence, unfortunately, the promising theoretical guarantees provided by SHERPA are difficult to retain in practice, even when there is no uncertainty in the bounding model.

It turns out that these observations are not unusual. In Figure 7.34, performance over 10 learning trials under supervision by SHERPA is summarized. Note that a smaller number of independent runs was performed



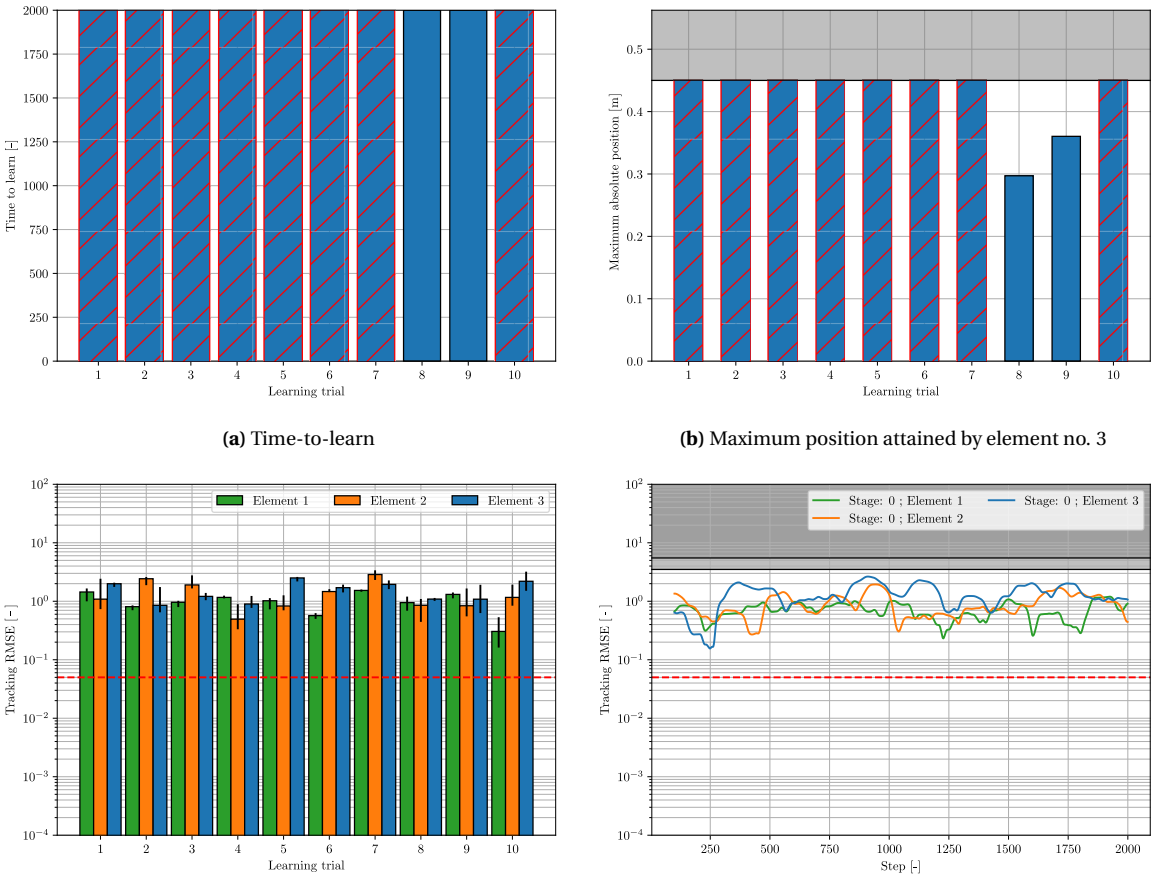
(a) System response

(b) Actuator inputs

(c) Kernel evolution (diagonal only)

(d) Real eigenvalue components of closed-loop system

Figure 7.33: Example learning trial for the MSD-3 tracking task using SHERPA as safety mechanism with 0% bounding model uncertainty ($\epsilon_i^{max} = 0$); red regions indicate that SHERPA is performing a control backup (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)



(a) Time-to-learn

(b) Maximum position attained by element no. 3

(c) RMSE at end of learning

(d) RMSE evolution (example trial)

Figure 7.34: Learning performance statistics over 10 independent trial runs for the MSD-3 tracking task using SHERPA as safety mechanism with 0% bounding model uncertainty ($\epsilon_i^{max} = 0$) (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

than before, due to the long running times involved with the simulations. Two conclusions can be drawn here. First, despite its theoretical guarantees, this implementation of SHERPA cannot provide for safety in the MSD-3 tracking task. It can be seen that the full learning time of 20 seconds is achieved in only 2 out of 10 trials, with varying number of maximum system deviations (see Figure 7.34b). Second, the fact that the algorithm is very often not able to identify adequate backups implies that the learning agent almost never has authority over the controls, whereas it has already learned the optimal control law. Ironically, although the agent is already able to preserve ergodicity by itself, SHERPA's attempts to verify this prevents the agent from taking over the controls and in fact often leads to unsafe behavior. The former is also reflected by Figure 7.34d, which shows the tracking RMSE for a rare successful learning trial.

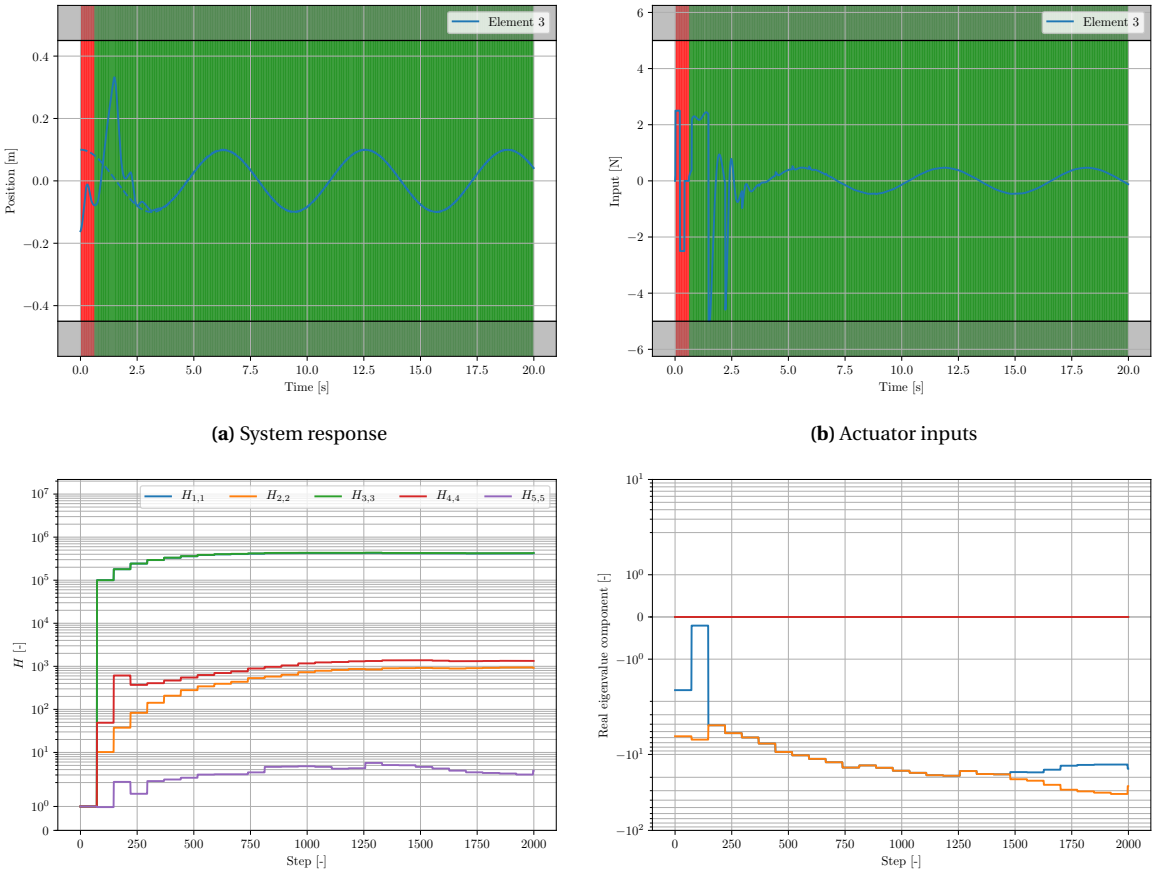
In order to further compare SHERPA's performance with what is reported in e.g. [70], the same experiment will be repeated for the MSD-1 system. Since this system has a smaller state-action space and is inherently stable, it is expected that both learning performance and safety will be considerably enhanced. It is again assumed that the bounding model is in fact crisp, for reason of comparison. Figure 7.35 illustrates a typical learning trial for this example. Because of the speedy random initialization, SHERPA immediately decides that a backup is necessary in the form of a doublet. However, once the backup has ended, control authority is returned to the learned agent, since ergodicity under its input has been verified. In fact, this continues for the remainder of the learning process, which is provably safe over its full length as indicated by the green area. At the same time, the agent is able to quickly converge to the optimal control law. Hence, this is in fact the first example of provably safe adaptive optimal control. Figure 7.36 gives a statistical overview of the effectiveness of SHERPA in the MSD-1 tracking task.

7.4.2. Ranking Control Options using optiSHERPA

The tedious backup search process for the MSD-3 system forms SHERPA's key limiting factor. This section therefore investigates an equivalent experiment with optiSHERPA, which takes a different approach to safety by evaluating a finite set of control input sequences and ranking these according to safety of the system. In this way, a less strict distinction can be made between acceptable and unacceptable behaviors. As explained in Subsection 5.2.2, optiSHERPA constraints the system evolution by means of a distance and an evasion metric. The former serves to keep the system close to already visited regions, and is repeated here:

$$d(\boldsymbol{x}_p,[\boldsymbol{x}_k]) = \left\| \left(\boldsymbol{x}_p - \frac{\sup([\boldsymbol{x}_k]) + \inf([\boldsymbol{x}_k])}{2} \right) \odot \boldsymbol{v}_r \right\| + \rho \cdot \left\| \left(\frac{\sup([\boldsymbol{x}_k]) - \inf([\boldsymbol{x}_k])}{2} \right) \odot \boldsymbol{v}_r \right\| \tag{5.9}$$

Essentially, this distance metric is applied in two ways. First, it serves to evaluate the future input sequence generated by the agent, which is now supplied with a predictive model that is sampled randomly from the bounding model. The reason that a model instance is being used is that the LQT Q-learning agent acts as a feedback controller, which implies that inputs can only be generated by crisp states. However, recall that our formulation of the LQT problem is based on the augmented system dynamics, which implies that the true future feedback control signal can only be retrieved if we also propagate the reference signal. This would make the problem non-causal, since normally the future reference cannot be predicted. There are two ways that one may account for this: one can either make some predictions based on past behavior, or one can assume that the reference signal stays constant over the prediction window. For this study, the latter option has been chosen. The second way the distance metric is implemented is to evaluate randomly generated future control signals, which are proposed by optiSHERPA as alternatives to the controls proposed



(a) System response

(b) Actuator inputs

(c) Kernel evolution (diagonal only)

(d) Real eigenvalue components of closed-loop system

Figure 7.35: Example learning trial for the MSD-1 tracking task using SHERPA as safety mechanism with 0% bounding model uncertainty ($\epsilon_i^{max} = 0$); red regions indicate that SHERPA is performing a control backup, whereas green regions imply that ergodicity can be preserved for the control input proposed by the agent (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

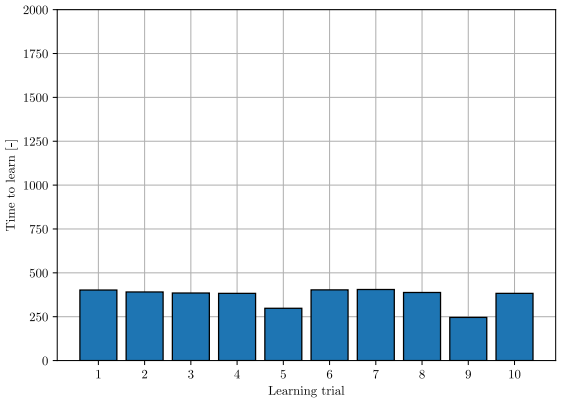
by the learning agent. These inputs are essentially open-loop, which implies that the bounding model will be used to propagate the future state.

The distance metric makes use of two parameters, being the weighting terms $\boldsymbol{v}_r \in \mathbb{R}^{n+}$ and $\rho < 1$. The former needs to reflect the level of risk associated with each state $\boldsymbol{x} \in \mathscr{X}$, which for this case implies that the highest weight should be assigned to x_3. However, since the distance metric essentially forms an alternative to SHERPA's closeness condition, it can be argued that $\dot{x}_3$ should be assigned similar values to ensure bounded excitation of the system. In this implementation, it was chosen to assign each position and velocity state a weight of 20 m and 20 m/s, respectively. By assigning the same values to all states, rather than higher values for element 3 only, one embeds some form of a-priori information about the fact that all elements are coupled. For ρ, a value of 0.5 was selected in analogy to [70].

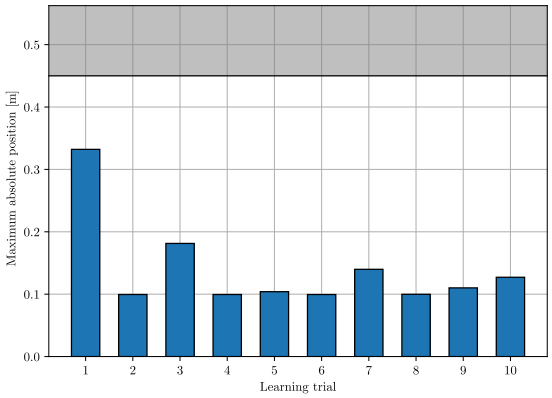
Subsequently, the evasion metric needs to be implemented in case the distance sensor detects a distance between the outer element and the RFSS smaller than 0.2 m. The result produced by this second metric is nothing more than the sum of a range of intersections with a collection of hierarchically ordered regions R_i. The evasive input sequence for which this sum is the smallest is than considered the optimal control, as given by Equation 5.10 and repeated here:

$$\tau^* = \arg\min_{\tau_j \in \boldsymbol{\tau}} \sum_{R_j} \rho_{ij} w_i \tag{5.10}$$

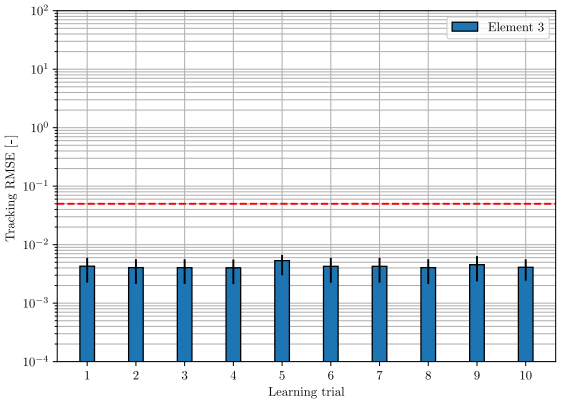
In this example, three regions R_{safe}, $R_{unknown}$, and R_{fatal} will be considered, ordered in increasing level of risk. The former is equal to the internal SSS, whereas the latter corresponds to the true FSS. The remaining region $R_{unknown}$ then forms the complement, i.e. $\mathscr{X} \setminus \{R_{safe} \cup R_{fatal}\}$. The weights selected for each of



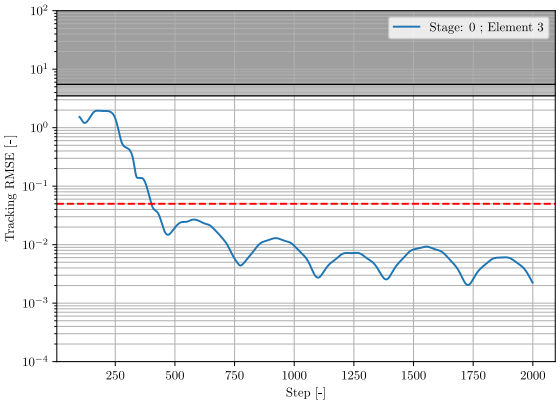
(a) Time-to-learn



(b) Maximum position attained by element no. 3



(c) RMSE at end of learning



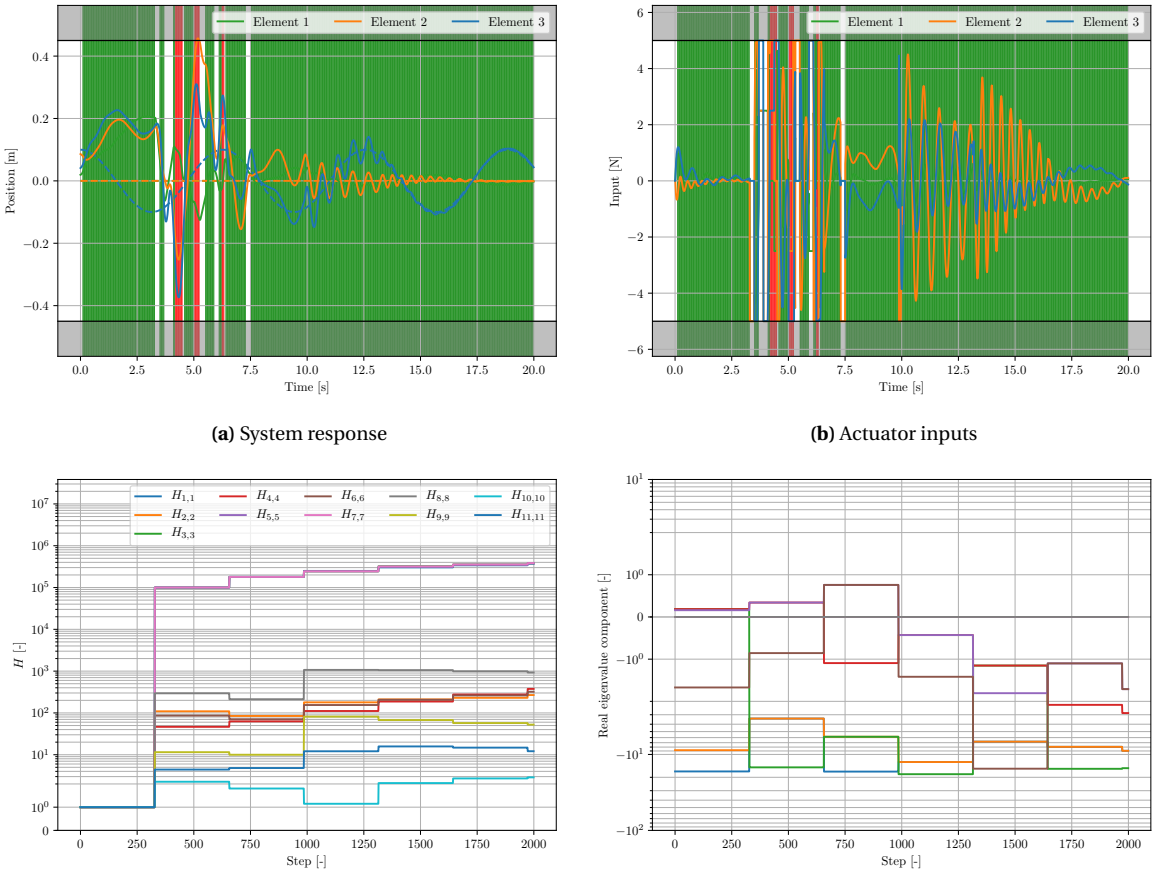
(d) RMSE evolution (example trial)

Figure 7.36: Learning performance statistics over 10 independent trial runs for the MSD-1 tracking task using SHERPA as safety mechanism with 0% bounding model uncertainty ($\epsilon_i^{max} = 0$) (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

these regions are taken as 10, 10^1 and 10^3, respectively, which well reflects the inherent risk levels. Before Equation 5.10 can be applied however, it must be verified for each trajectory τ_j that $\tau_j \cap \{R_{safe} \cup R_{unknown}\} \neq \varnothing$, and that $[\boldsymbol{x}_N] \subset R_{safe}$ with N the length of τ_j, as described in [70]. Assuming that τ_j passes this test, one can proceed with calculating its intersection ρ_{ij} with each region R_i. For general regions with complex geometries, this is not necessarily very straightforward. However, the heuristic nature of the computations allows for a significant margin of error, at least when ρ_{ij} is over-approximated. Therefore, each intersection will simply be the sum of the one-dimensional intersections for each dimension $i = 1, 2, \ldots, n$ in $\mathscr{X}^n$.

Finally, some settings are required for practical operation of optiSHERPA, similarly to SHERPA. The sensing capability, the input sequence structure and length, and the discretized action space are all taken directly from the SHERPA implementation, as described in the previous section. The total number of uniquely generated input sequences that are evaluated under the distance and evasion metrics is set to 20. Similarly, optiSHERPA is only invoked every 10 time steps to save on computational complexity, except when risk is detected. Additionally, trajectory evaluation under either the distance or evasion metric is discretized in steps of 20. For the distance metric, another setting is introduced in the form of an *agent priority level*, which assigns a minimum level of control authority to the learning agent. In this case, an initial value of 25% is selected, which indicates that optiSHERPA has to accept the proposed agent input sequence if it is among 25% of the best performing candidates. Moreover, to enable gradual transfer from exploration to exploitation of the control policy, the agent priority level is increased by 0.05% with every time step.

The first case study will again assume that optiSHERPA is equipped with a crisp model of the system dynamics, in order to have an upper benchmark in terms of performance. Figure 7.37 shows a successful learning trial for the MSD-3 task under supervision of optiSHERPA. For the majority of the time, the control



(a) System response

(b) Actuator inputs

(c) Kernel evolution (diagonal only)

(d) Real eigenvalue components closed-loop system

Figure 7.37: Example learning trial for the MSD-3 tracking task using optiSHERPA as safety mechanism with 1% bounding model uncertainty ($\epsilon_i^{max} = 0.01$); red regions indicate that optiSHERPA performs an evasion, whereas green regions imply that the agent input has been considered safe (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

inputs are quite messy, which is mostly due to the random nature of the output returned by optiSHERPA if it considers the agent input inadequate. In the figure, green regions indicate that the optiSHERPA has accepted the agent input, whereas red regions correspond to evasive maneuvers. Uncolored regions imply that a random control sequence has been favored over the agent input under the distance metric. It is interesting to see that during the first half of the learning trial, optiSHERPA takes over for the majority of the time, which is a direct result of the fact that the closed-loop dynamics under the intermediate control law are unstable. In the second half however, where the policy has stabilized, optiSHERPA starts to accept the agent input more and more, which is also the result of the increased agent priority level. This is very neat example of safe reinforcement learning[3].

In Figure 7.38, performance over 25 independent learning trials under supervision of optiSHERPA with $\epsilon_i^{max} = 0.01$ has been summarized. Although safety has been improved compared to learning from scratch, system crashes are still observed in 60% of the runs. Intuitively, the statistics become even worse as model uncertainty is increased. Figure 7.39 shows the same information when $\epsilon_i^{max} = 0.25$, i.e. when the bounding model uncertainty equals ±25%. In this case, the percentage of trials for which learning is safe drops even below 10%. There are two main factors that contribute to these low numbers. First, there is again the dimensionality of the state-action space that also played a detrimental role for learning under supervision by SHERPA. A second aspect is that the evasive maneuvers enforced by optiSHERPA are not necessarily safe in itself. The reason is that the optimal evasive input sequence $\boldsymbol{U}_e^*$ that maximizes Equation 5.10 is only guaranteed to minimize the intersection with R_{fatal}, but this intersection will very often not be empty. By adopting the first input of every $\boldsymbol{U}_e^*$ successively, the system may inadvertently brought into a lead-to-fatal

[3] However, it must be kept in mind here that optiSHERPA does not provide strong theoretical guarantees.

(a) Time-to-learn

(b) Maximum position attained by element no. 3

(c) RMSE at end of learning

(d) RMSE evolution (example trial)

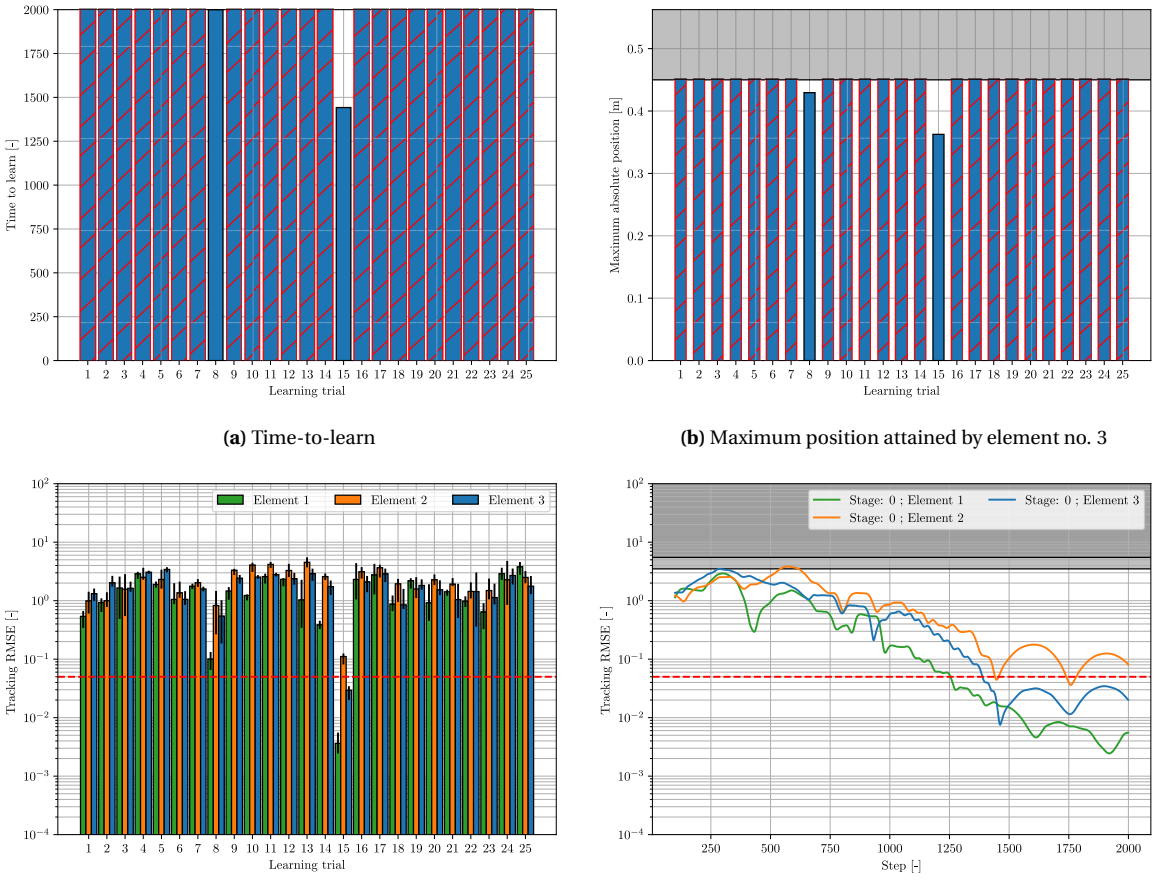
Figure 7.38: Learning performance statistics over 25 independent trial runs for the MSD-3 tracking task using optiSHERPA as safety mechanism with 1% bounding model uncertainty ($\epsilon_i^{max} = 0.01$) (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

state. Therefore, these results underscore that optiSHERPA often cannot preserve ergodicity.

7.5. Safe Curriculum Learning

Thus far, it has been established that both inter-task and intra-task curriculum learning can lead to safety through stability for a wide range of configurations. The ability to stabilize the dynamics of the MSD-3 system implies that the condition of ergodicity is often satisfied implicitly, although this cannot be guaranteed. The latter can be achieved by ergodicity-preserving mechanisms such as SHERPA, but this algorithm often fails to identify control backups in case of high-dimensional systems with considerable uncertainties. This can lead to the highly undesirable situation where an attempt to explicitly verify the safety of the system can lead to unsafe behavior, whereas the agent policy has already matured sufficiently to prevent unsafe transitions by itself. The approach used by optiSHERPA works better in practice, which trades the strong theoretical guarantees by SHERPA for enhanced computational complexity properties. However, even when optiSHERPA is equipped with a crisp model of the dynamics, the MSD-3 system is often driven into a crash state when learning from scratch.

In this section, the integrated safe curriculum learning approach is investigated for its ability to ensure safe and efficient online learning of the optimal control policy for the MSD-3 system. Specifically, it is investigated whether the existence of intermediate stabilizing policies can enhance SHERPA's ability to identify control backups for a given state $\boldsymbol{x} \in \{\mathscr{X} \setminus \mathscr{X}_{safe}\} \cap \mathscr{X}^{\pi}_{stable}$. Note that for the MSD-3 system, it is that $\mathscr{X}^{\pi}_{stable} = \mathbb{R}^n$ for the complete space of stabilizing policies; therefore, it is assumed that the effects of stabilizing behavior can be observed through the fact that the stabilizing policy will keep the system state sufficiently far away from the FSS. The existence of this region will be denoted as $\overline{\mathscr{X}}^{\pi} \subset \mathscr{X}_{safe}$. This is analogous to the



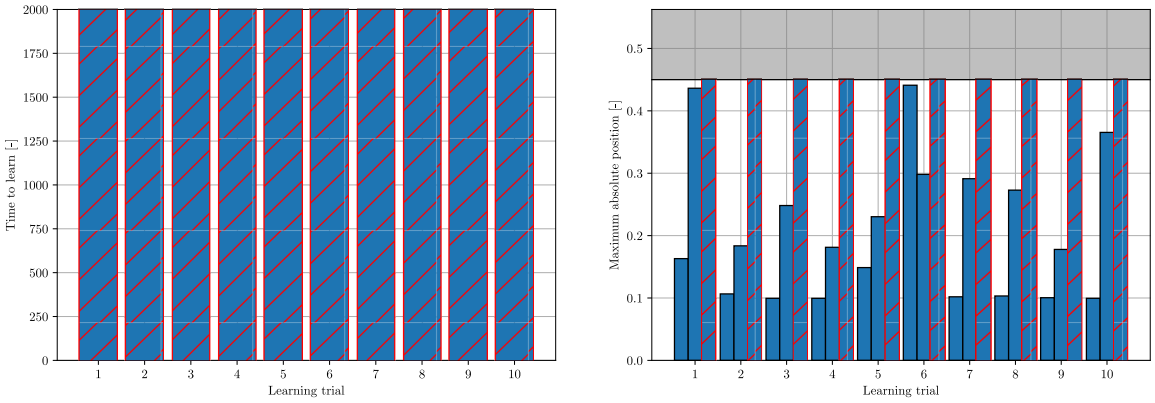
(a) Time-to-learn

(b) Maximum position attained by element no. 3

(c) RMSE at end of learning

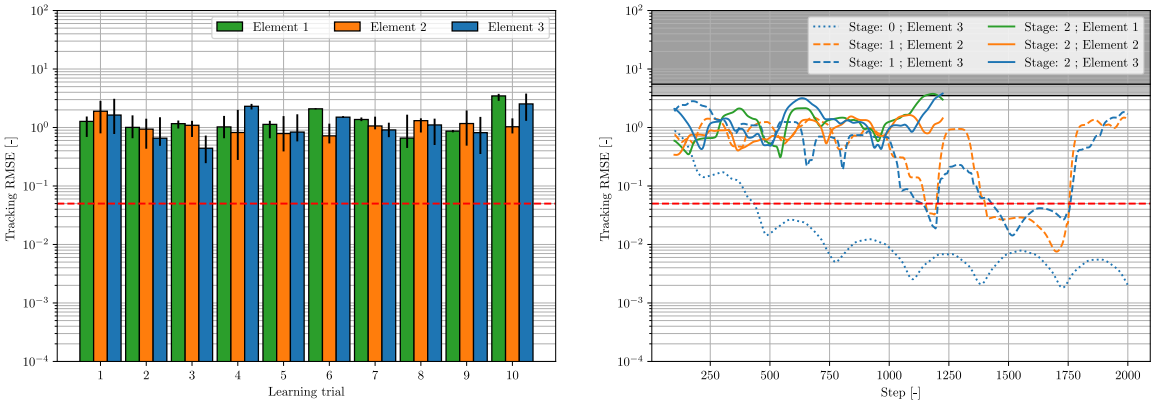
(d) RMSE evolution (example trial)

Figure 7.39: Learning performance statistics over 25 independent trial runs for the MSD-3 tracking task using optiSHERPA as safety mechanism with 25% bounding model uncertainty ($\epsilon_i^{max} = 0.25$) (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)



(a) Time-to-learn (target stage)

(b) Maximum RMSE during learning



(c) Performance at end of learning (stage 3)

(d) RMSE over 1s window (illustrative trial)

Figure 7.40: Learning performance statistics over 25 independent trial runs for the MSD-3 tracking task using SHERPA as safety mechanism with 0% bounding model uncertainty ($\epsilon_i^{max} = 0$) as part of the inter-task learning curriculum from Figure 7.4 with basic kernel mappings (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

situation where a locally stabilizing policy maintains the system state close to a given operating point (e.g., the origin). Secondly, a learning curriculum assisted by optiSHERPA will be investigated. This approach results in a weaker form of safe curriculum learning, since ergodicity during learning is not fully guaranteed. This implies that the framework from Figure 6.1 does not fully apply. However, this is still a very important practical approach to safe curriculum learning. For benchmarking purposes, the simple inter-task learning curriculum from Figure 7.4 will be used, in combination with the basic kernel mapping strategy.

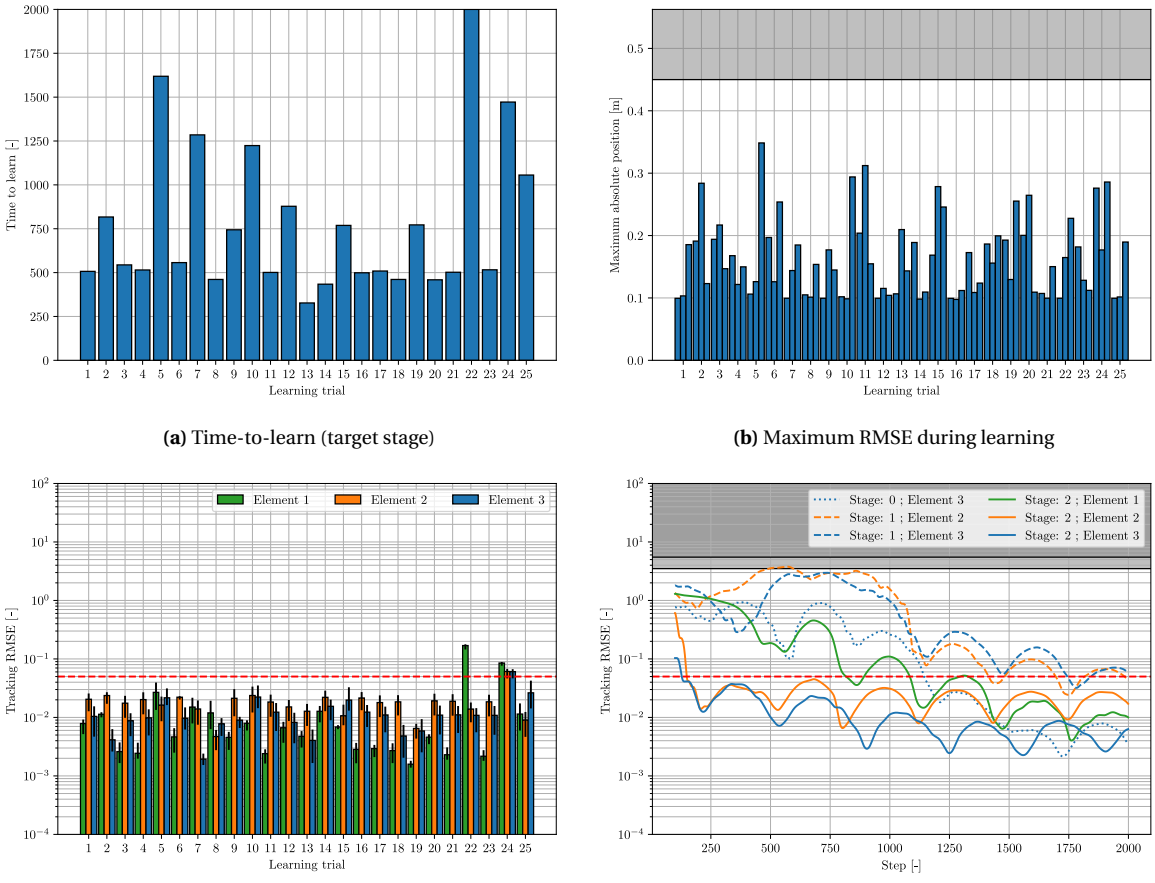
7.5.1. Inter-task staging assisted by SHERPA

Figure 7.40 summarizes recorded performance over 10 independent learning trials for the inter-task learning curriculum with SHERPA as safety mechanism. Unfortunately, the expected benefits of curriculum learning on SHERPA's backup identification ability are not visible here, as the system is seen to crash consistently over the trial distribution. Even worse, the small rate of success seen earlier in Subsection 7.4.1 has completely vanished. However, due to the relatively small distribution size, it cannot be concluded that this performance deterioration can be attributed to the influence of the learning curriculum. Still, despite the intuition that the space of available backup sequences is larger when the system state remains bounded to the region $\overline{\mathscr{X}}^{\pi}$, the current setting does not reflect this hypothesis. It is believed that the reason for this is the availability of too little data to consistently meet the closeness condition (Equation 5.5) for the control backup. Whereas it may be that $\boldsymbol{x} \in \overline{\mathscr{X}}^{\pi}$ $\forall t \in [0, \infty)$ by virtue of the initially stabilizing policy, this only means that the backup trajectory will fall within the SSS with high probability (Equations 5.6 and 5.7). The fact that a too limited number of states is sampled in memory means that the space of backup policies is still heavily constrained. Therefore, although the hypothesis that stability can add to ensuring ergodicity still stands, this experiment

shows that this property can be heavily undermined if the closeness condition cannot be met.

An appealing strategy to resolve the issues with insufficient experience in a new learning stage is to perform instance transfer of states observed in a previous stage, and use these transformed states to seed SHERPA's memory in the new stage. In this way, the space of backup sequences that satisfy the closeness condition can be considerably enlarged, although such a strategy is in strong tension with the theoretical guarantees on ergodicity. The reason is that infinite exploration can only be guaranteed if it is known that a safe return strategy exists to an already visited state. A mapped observation from a different, but related MDP will theoretically speaking not satisfy this condition. However, the fact that SHERPA already adopts finite reaching intervals $[\boldsymbol{\epsilon}]$ and $[\boldsymbol{\delta}]$ by itself implies that the quality of the backup will be limited by definition. Therefore, in case a good mapping between the successive stages can be found, a similar backup quality could hypothetically be achieved. Moreover, it could be opted to let SHERPA query transferred observations only if has an insufficient number of real observations at its disposal. This is comparable to the strategy adopted by the TIMBREL algorithm, as reported in [103]. In this way, curriculum learning could be tied more closely to the practical operation of safety-preserving algorithms such as SHERPA. Still, even if there would be a dense seeding of SHERPA's memory, the fact that backup identification is performed based on random search can still result in issues with iteration depletion. This is also visible in Figure 7.40d, where the RMSE changes drastically even upon convergence of the agent policy.

Another approach to further integrate curriculum learning and safe learning is to alter the definition of the bounding model such that the beneficial effect of dynamic stability under the agent policy can be used to expedite the search for ergodicity-preserving backup sequences. This strategy is adopted in the safe curriculum learning framework described in Part I.



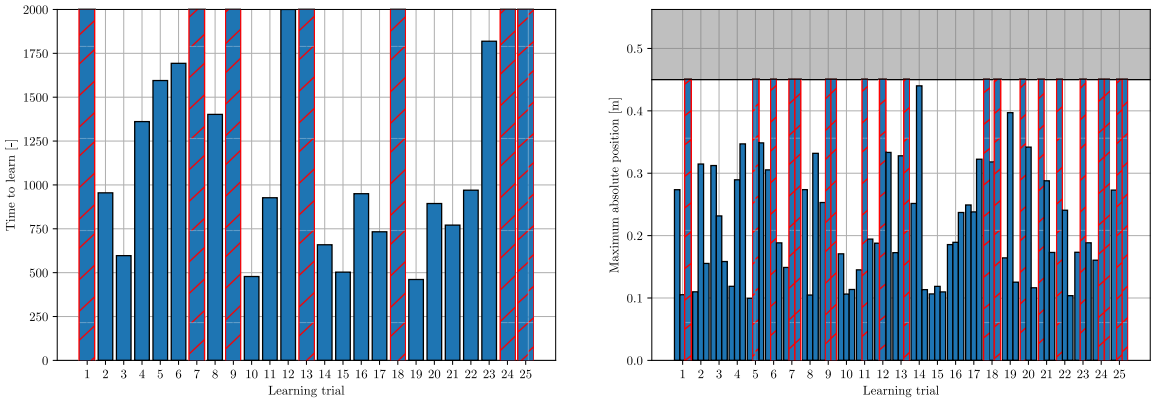
(a) Time-to-learn (target stage)

(b) Maximum RMSE during learning

(c) Performance at end of learning (stage 3)

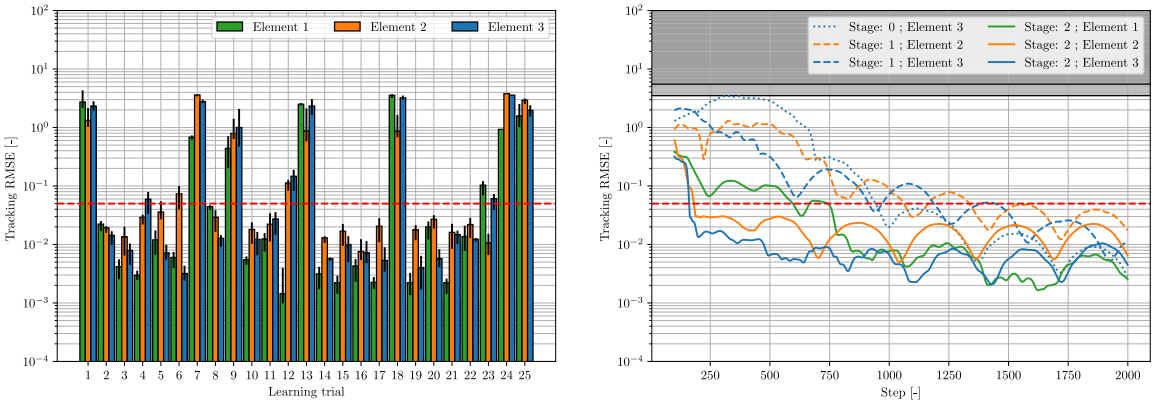
(d) RMSE over 1s window (illustrative trial)

Figure 7.41: Learning performance statistics over 25 independent trial runs for the MSD-3 tracking task using optiSHERPA as safety mechanism with 1% bounding model uncertainty ($\epsilon_i^{max} = 0.01$) as part of the inter-task learning curriculum from Figure 7.4 with basic kernel mappings (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)



(a) Time-to-learn (target stage)

(b) Maximum RMSE during learning



(c) Performance at end of learning (stage 3)

(d) RMSE over 1s window (illustrative trial)

Figure 7.42: Learning performance statistics over 25 independent trial runs for the MSD-3 tracking task using optiSHERPA as safety mechanism with 25% bounding model uncertainty ($\epsilon_i^{max} = 0.25$) as part of the inter-task learning curriculum from Figure 7.4 with basic kernel mappings (*value iteration using Q-functions; batch LS policy evaluation; global agent representation; random initialization with* $x_i \in [-0.2, 0.2]$, $\dot{x}_i \in [-0.25, 0.25]$)

7.5.2. Inter-task staging assisted by optiSHERPA

The fact that optiSHERPA uses a different strategy for enforcing safety that is not dependent on strict closeness conditions implies that the advantages of curriculum learning may become more evident. The way that the distance metric has been defined suggests that optiSHERPA may be more compatible as a safety algorithm (that is, without further adaption), as the stabilizing properties of curriculum learning already keep the system close to a known region. Figure 7.41 visualizes observed performance over 25 independent learning trials for the inter-task learning curriculum with optiSHERPA as safety mechanism. In this experiment, use was made of a close-to-crisp prediction model ($\epsilon_i^{max} = 0.01$). Evidently, safety has considerably increased compared to optiSHERPA-assisted flat learning. Whereas for the latter the algorithm could only prevent the system from crashing in 40% of the trial runs, here the system remains consistently safe. This demonstrates the powerful capabilities of the safe curriculum learning paradigm. Compared to standard curriculum learning however, learning efficiency is somewhat reduced, which is a direct result of the at times inefficient application of alternative control sequences.

Finally, the same experiment has been performed again with an uncertain bounding model, with $\epsilon_i^{max} = 0.25$. The results are presented in Figure 7.42. Whereas previously optiSHERPA was only able to ensure safety in less than 10% of the learning trials for this case, thanks to the curriculum learning approach this number has risen to over 70%. The ability of mapped intermediate policies to keep the system state bounded to a region $\overline{\mathscr{X}}^{\pi}$ $\forall t \in [0, \infty)$ is largely reflected by optiSHERPA's increased approval rate, even in face of large prediction uncertainties. Still, the latter is responsible for the fact that about 30% of the trials does not end successfully, which shows that there is considerable room for improvement. In this view, online adaption of the bounding model would be worth further investigation, as argued in Section 6.2.

7.6. Summary

With the empirical results obtained with the unstable MSD system, practical insight has been obtained into how the safe curriculum learning paradigm may expedite online learning of adaptive optimal control laws in terms of safety and efficiency. Although the system dynamics, the reinforcement learning approach, and the curriculum setups considered are all particularly simple, tangible conclusions can be drawn on curriculum effectiveness, the influence of the nature of staging and agent flexibility, and how a learning curriculum can enable provably safe learning through ergodicity even for high-dimensional systems. The main outcomes are briefly recited here.

It was found that an a-priori designed inter-task curriculum greatly increases safety through stability during the learning process, and that efficiency can be enhanced by adopting greedy mapping strategies. However, mapping greedily may reduce the reliability of intermediate kernel updates, leading to closed-loop instability during the early phases of learning. This also illustrates that provably safe learning can only be achieved by means of a separate safeguard mechanism. However, for real-life applications where a reinforcement learning agent needs to interact with a physical process or system, inter-task curriculum learning is often not a feasible option. Therefore, several experiments where performed with intra-task curriculum learning, which represents the other extreme of the nature of staging in the curriculum learning taxonomy. The variables that were of primary interest in these experiments where 1) the internal representation of the agent and 2) the effectiveness of the external supervisor, which was implemented as a pair of suboptimal PD controllers. It appears that intra-task curriculum learning is a viable option, even when the agent only has a limited (local) representation of the system dynamics, and the assistance provided by the supervisor is weak. This shows that the learning framework considered here is highly robust to situations where observability and controllability are very limited. This is a remarkable result, especially since the optimal control problem becomes non-Markovian with the former. Still, the safety-enhancing properties of curriculum learning are considerably deteriorated if the learning curriculum is further corrupted.

To make learning provably safe, the practical capabilities of two ergodicity-preserving safe learning mechanisms were investigated for learning from scratch and in a curriculum learning setting. The results obtained with the SHERPA algorithm for flat learning revealed that online backup identification suffers heavily from the curse of dimensionality, even in case the algorithm is equipped with a crisp model of the system dynamics. Here, the undesirable situation arises where the inability of the safe learning mechanism to verify whether the input proposed by the agent is ergodicity-preserving leads to unsafe behavior in itself. An equivalent experiment with the optiSHERPA algorithm showed less issues in this regard, as a result of its different strategy with respect to safety. Although the system was seen to crash less often compared to completely flat learning, the results obtained with optiSHERPA were not as desirable as what can be achieved with a learning curriculum.

Finally, the results obtained with the integrated safe curriculum learning approach provide important insight and further suggestions into how curriculum learning can be made provably safe. The inter-task curriculum setting with SHERPA as safety mechanism revealed little improvements compared to learning from scratch or SHERPA-assisted flat learning, despite the expected enhanced backup identification capabilities. It is hypothesized here that the backup policy space is still too severely constrained by a lack of stored samples, which means that the closeness condition is rarely met. This implies that additional measures are required to improve this aspect. By contrast, with optiSHERPA acting as the external safeguard, the stabilizing properties of curriculum learning were shown to be beneficial, even for large uncertainties in the predictive bounding model. This is a key result, as it demonstrates that safe learning of optimal adaptive control laws can be achieved for uncertain, unstable and high-dimensional systems with continuous state and input spaces in a fully online manner. However, due to the fact that optiSHERPA does not provide any theoretical guarantees on the preservation of ergodicity, it may be worthwhile to further explore possibilities that enable the integration of the curriculum learning framework with the SHERPA approach.

IV

Wrap-up

8

Conclusions and future research

In this final part, the thesis is concluded by drawing answers to the research questions and discuss recommendations for future work. However, a brief synopsis of the contributions and findings is provided first.

8.1. Synopsis

The main contribution of this thesis is an integrated framework known as safe curriculum learning to simultaneously improve the sample efficiency and safety of reinforcement learning techniques in the context of adaptive optimal control for online safety-critical applications. To this end, existing views on curriculum learning, transfer learning, and safe learning have been combined into a coherent paradigm that can be used for optimal control applications. This framework exploits the concept of transferring safe policies across successive tasks in a learning sequence, resulting in curriculum strategies that can be used to enhance safety in a similar way as traditional curriculum learning aims to improve learning efficiency. Implementation of the learning scheme has been demonstrated in the context of optimal control of a generic cascaded unstable mass-spring damper system, as well as optimal attitude control of a nonlinear quadrotor UAV. The latter serves primarily as a proof-of-concept that shows how safe curriculum learning can be adopted in primary flight control applications.

A broad taxonomy has been established to further substantiate the curriculum learning paradigm. The key motivation for curriculum learning is that the ability of an RL algorithm to converge towards the optimal control policy is inherently related to the complexity of the task at hand. This complexity covers multiple aspects, such as the dimensionality of the state and input spaces, the nature of the dynamics, and the richness of the reward signal. By reducing the learning complexity in a sequence of auxiliary source tasks, a learning agent may be better able to converge towards a desired control strategy. The proposed taxonomy suggests that there are three main areas that define a generic reinforcement learning curriculum: the shaping dynamics, the nature of staging, and the flexibility of the agent representation. These dimensions describe the extent to which the training distribution is organized by an external domain agent or autonomously by the agent itself, how the sequence of tasks is related to each other, and whether the agent representation (i.e., policy, value function, etc.) will undergo any transformations during the curriculum. This taxonomy shows that a learning curriculum can take many forms and that a wide range of techniques may be involved, such as those related to self-paced learning and transfer learning.

Similarly, an extensive review of the field of safe reinforcement learning has been conducted to identify the key concepts behind safe learning. Whereas the problem of safety can appear in both the exploitation and exploration phases of learning and can be highly problem-dependent, online interaction with safety-critical systems generally requires the system state to remain bounded to a safe subset of the complete state space. This condition can only be guaranteed if some form of a-priori knowledge is available. In systems theory, a central topic that captures this objective is known as Lyapunov stability. Although the existence of a (control) Lyapunov function can be used to guarantee boundedness of the system state in a wide range of applications, including those involving state- and input-constrained systems with bounded uncertainties, finding this function can be challenging for general nonlinear systems. Therefore, using other forms of a-priori knowledge, such as a-priori global reference information or prediction models, may result in a more practical approach to safe learning. Here, a key topic reported in literature is known as ergodicity, which

ensures that there exists a feasible control sequence between any arbitrary pair of states in the safe subspace. Ensuring ergodicity requires a belief or (uncertain) model of the system dynamics, which makes the process inherently model-based. Ergodicity-preserving policies can be obtained by adopting a global approach, but can also be identified online on a local level. In this thesis, a local approach has been adopted in the form of an external safety filter that overrides unsafe agent inputs by alternative control sequences that maintain ergodicity.

Despite its appealing properties, it has been demonstrated that online verification of ergodicity during learning can be difficult to achieve for uncertain, unstable, high-dimensional systems with continuous state and input spaces. A central aspect of the proposed safe curriculum learning paradigm is that an appropriately designed learning curriculum can increase the likelihood of ergodicity preservation in tasks of higher complexity. This is a direct benefit of the scaling of control policies obtained in one or more low-complexity source tasks towards succeeding tasks of higher complexity. For safety-critical optimal control tasks of relatively low complexity, such as those involving open-loop unstable systems with low-dimensional state and input spaces, online identification of ergodicity-preserving policies has a relatively high rate of success, which implies that control inputs generated by the learning agent or those related to persistent system excitation (exploration) can be effectively contained by the ergodicity-preserving mechanism. As a result, the agent can learn safely in the low-complexity source task. As the agent gets closer towards the optimal control policy in the source task, this information is used to initialize learning in a succeeding task of higher complexity, such that the more complex system is closed-loop stable as well. Under the safe curriculum learning framework, this dynamic stability property forms one of the key aspects that is used to expedite the search for ergodicity-preserving control sequences to ensure that no safety constraints are violated when learning in the more complex target task. Another aspect is formed by reduced uncertainty propagation in the associated internal model predictions.

Depending on the design of the learning curriculum, which follows from the curriculum learning taxonomy, various approaches to information scaling can be conceived. In the work presented in this thesis, the scope has been limited to a-priori designed learning curricula with flexible agent representations. For this particular class of learning curricula, the use of knowledge mappings is required to transfer information across successive representations of the value function and/or policy. The exact form depends primarily on the type of function approximation used for the agent representation. In the Linear Quadratic Tracking (LQT) experiment, the linear matrix representation of the state-action value function warrants the use of kernel mappings to transfer Q-values across successive learning stages. For the nonlinear neural-network based actor-critic learning scheme, network mapping strategies have been proposed that map neural network weights and biases between successive representations of the actor and the critic. Because of their central role in the fixed type of learning curricula considered in this thesis, the qualtiy of these knowledge mappings fully determines the safety and efficiency properties of the framework. In terms of safety, a primary objective for these mappings is that the dynamic system is stable under the autonomy of the mapped policy. Regarding learning efficiency, a primary objective would be to have the mapped representation approximate the optimal representation as much as possible. However, this generally requires an external form of domain knowledge. Moreover, knowledge mappings can be quite task-specific.

8.2. Answers to the research questions

The research sub-questions are considered first, by addressing them on an individual basis. This yields the necessary information to formulate an answer to the main research question. The answers to these sub-questions are formulated below.

Research sub-question 1

1. *How can safe learning be supported by reducing the complexity of a given task?*
   (a) *What defines task complexity?* In the task domain, learning complexity arises in three main areas: (1) the dimensionality of the state space and/or action space; (2) the nature of the dynamics; and (3) the richness of the reward signal. Regarding the nature of the dynamics, complexity arises in terms of e.g. strongly coupled dynamics, nonlinear dependencies, discontinuities, instabilities, badly or non-controllable dynamics, and strong stochastic effects. A key insight is that the task domain is not solely responsible for the problem of learning

complexity, but that there is a strong connection with the abilities of the agent itself.

(b) *What are the key concepts behind safe learning?* The system's risk profile, which must be fully known a-priori, is the driving factor in the need for safety measures in the learning process. This risk profile culminates in a necessity to keep the system state bounded to a safe subspace of the complete state space. This can be ensured if ergodicity is preserved throughout the learning process, which implies that at any moment in time a sequence of control inputs is known to exist for which it is guaranteed that the system state can be driven back to any other arbitrary state in the safe subspace. In this view, model-based search techniques form a practical approach to safe learning.

(c) *Which relationships can be identified across these concepts?* The relative subspace of ergodicity-preserving policies forms the primary factor that relates task complexity to learning safety and vice versa. In case this subspace is relatively small, which is often case for high-complexity tasks that involve e.g. high-dimensional state and input spaces, verification of ergodicity can be difficult to achieve. The opposite is also true.

Research sub-question 2

2. *What are the requirements for a sequence of intermediate flight control learning stages with respect to the task domain and the agent domain, such that fast learning can be achieved in a safe way?*

   (a) *What is needed in terms of task complexity, ordering, and timing?* An adequate learning curriculum implies that the complexity and diversity of learning samples should be monotonically increasing with every learning stage. This should be reflected in the complexity of each intermediate task itself, together with the ordering of the sequence. Learning efficiency can be optimized through a strict curriculum pace, which implies that the timing of complexification and diversification should be such that the agent has gained sufficient proficiency from a preceding stage. However, safety enforces a minimum constraint on timing.

   (b) *What are the capabilities that the agent should possess at each stage of the learning process?* A minimum requirement for intermediate agent abilities is that the policy should at least lead to closed-loop stability for semantically similar operating conditions in the early phases of learning in a new stage. This expedites the verification of ergodicity by an external safety mechanism.

   (c) *How does the objective function need to be set up across the learning curriculum?* From a safety perspective, any objective function that results in stable optimal policies suffices. In the context of low-level optimal control, quadratic utility functions prove to be an effective solution in this regard.

   (d) *How to represent safety constraints?* Safety constraints must be conform the system's risk profile, and must be directly perceivable by the actor. A sensor-based approach is the most evident solution here, which enables the actor to determine whether fatal parts of the state space are in close reach. Hence, the designer must explicitly prescribe the boundaries of the various risk modes.

Research sub-question 3

3. *How can knowledge gained in a previous learning stage (the source) be adopted to bias learning in a later stage (the target) in a safe way?*

   (a) *What knowledge needs to be extracted from the source?* A wide range of different types of knowledge can be considered, including direct experience in the form of observed transition samples, low-level knowledge such as value functions or policies, or high-level information such as learning rules or options. For low-level optimal control tasks, value functions and/or

policies form the primary candidates in this regard, and can be used either in tabular or parametrized form.

(b) *How can this knowledge be represented?* The most appropriate way of representing value functions and/or policies depends on the nature of the optimal control task. The only requirement from the perspective of knowledge transfer is that existing knowledge must retain its semantic significance if it is to be re-used in successive learning tasks.

(c) *What needs to be known about the target?* To achieve positive transfer, it must be known how the target is related semantically to the source task under consideration. In the case of knowledge mappings, this implies that it must be known which regions of the state-input space correspond best to their source equivalents.

(d) *How to handle those parts of the target state and input space for which no knowledge is available?* Different strategies can be adopted here, depending on the nature of the learning task. From the perspective of safety, a primary requirement is that safe control policies can be safely re-used in successive learning tasks. For example, an agent policy that stabilizes the system in the source task can be safely re-used if the successor system is also Lyapunov stable under the autonomy of the transformed policy. In this view, it may be sufficient to leave stable unknown regions unbiased and stabilize the dynamics by virtue of the mapped local policy for the known areas. However, learning can be further expedited by duplicating knowledge from semantically similar regions.

(e) *What technique can be used to facilitate safe and effective transfer of knowledge across these stages?* For fixed learning curricula with flexible agent representations, knowledge mappings are required to transfer information across successive learning stages. Depending on the type of function approximation selected, simple strategies that involves duplication of function parameters can be adopted. In case of linear matrix representations, it is sufficient to simply duplicate (map) kernel values from the source kernel to the target kernel. For neural-network based actor-critic implementations, a strategy that involves the mapping of network weights and biases can be adopted.

Research sub-question 4

4. *What reinforcement learning architecture must be adopted to learn increasingly advanced flight control capabilities in a safe manner?*

(a) *What elements does the learning architecture need to consist of?* In general, for low-level, continuous, intelligent control laws, the learning architecture must consist of at least a value function, a(n) (uncertain) predictive model, and an external ergodicity-preserving entity. The predictive model takes two separate roles at the same time: it serves as the primary tool for verifying ergodicity, while it is also used to learn the optimal control law in case a state value function is used. For nonlinear system implementations, the policy can be represented using a separate actor network to reduce the level of knowledge required in the policy update step about the internal state transition dynamics of the system.

(b) *What learning technique needs to be used for solving the optimal control problem?* This is dependent on the learning problem at hand. In case of (approximately) linear dynamics, it is generally advantageous to use a quadratic representation of the value function and use least-squares techniques to solve the HJB equation. For highly nonlinear dynamics, however, stochastic gradient descent (SGD) techniques must be adopted.

(c) *What function approximation technique is suitable in terms of approximation power, sample efficiency, and transfer flexibility?* For parametric function approximators, it can be said that functionals that are linear-in-the-parameters are most sample-efficient since use can

be made of least-squares techniques as opposed to SGD. However, for general nonlinear applications, nonlinear function approximators such as multi-layer artificial neural networks (ANNs) need to be adopted. The latter class of functions typically enjoy considerably higher approximation power, which is especially useful in case of unknown and possibly complex function images such as those typically associated with value functions. It has been demonstrated that both function classes are suitable for knowledge transfer purposes.

(d) *What exploration strategy needs to be adopted?* For learning adaptive optimal control, persistent excitation of the system is of vital importance to identification of the system dynamics and convergence to the (global) optimal control policy. It has been demonstrated that using random exogenous exploration inputs is generally sufficient.

Research sub-question 5

5. *What metrics are required to quantify learning efficiency and safety?* It appears that the time-to-learn, performance at the end of learning, and the number of constraint violations provide good indications of learning efficiency and safety during a particular learning trial.

Based on these accounts, the answer to the main research question is formulated blow.

Main research question

How can a learning curriculum be implemented for a reinforcement learning agent to autonomously learn optimal primary flight control laws for an Unmanned Aerial Vehicle (UAV) with uncertain dynamics, such that learning efficiency is enhanced and the risk of violating safety constraints is reduced?

A learning curriculum can enhance safety and efficiency during learning if the principles of the safe curriculum learning paradigm are adhered to in its design. With this framework, safe policies are exploited across increasingly complex successor tasks in a learning sequence to bias the behavior of an intelligent learning agent more closely to the optimal control strategy, and expedite the search for ergodicity-preserving safe policies by an external safety filter.

8.3. Recommendations

The work presented in this thesis can be considered as an initial contribution to the safe curriculum learning framework for optimal control. As such, there are many challenges and directions for improvement that need to be addressed in future research. A primary challenge is the further reduction of the level of a-priori knowledge required about the system dynamics to have a true advantage in terms of model-dependency over the traditional flight control design cycle. In the first place, this relates to the ergodicity-preserving safety mechanism. Although it has been demonstrated that re-using safe agent policies can expedite the search for ergodicity-preserving control sequences, the level of uncertainty embedded in the internal bounding model quickly limits the performance of the safety algorithm. Therefore, model-learning techniques can be considered to reduce these uncertainties based on online data gathered from the system, resulting in the concept of contracting bounding models. As a result, it is hypothesized that the level of uncertainty can be relatively high at the start of learning, whereas verification of ergodicity may be more accurate over time as a better model of the system dynamics becomes available. In the second place, the level of knowledge required for task complexity management, i.e. curriculum design, could be further reduced as well by adopting self-paced learning techniques. This also the primary dimension of the curriculum learning taxonomy that has not been explored in this thesis.

Other directions for future research relate more to the practical operation of the safe curriculum optimal control framework. In particular, other approaches to verification of ergodicity can be considered. In this work, the selected mechanism is based on a random search of local ergodicity-preserving control sequences,

for which it has been demonstrated that performance can be limited in high-complexity learning tasks. Although it has been shown that its performance can be improved by re-using safe policies in a curriculum approach, there may be other ways to further improve safety during learning. For example, model-predictive control methods could be considered that solve the problem of local ergodicity by adopting an optimization-based approach based on the internal bounding model. A learning curriculum could still be beneficial in this situation, as a result of its property to constrain uncertainty propagation to reduced subsets of the state space. Moreover, re-use of safe agent policies could reduce the time required for optimization.

Another aspect that relates to the practical operation of the framework is the role of function approximation and the impact of hyperparameters. In this thesis, linear matrix and neural-network based agent representations have been used in the optimal control learning scheme. For highly nonlinear systems for which generalization may be problematic, it may be beneficial to use other types of function approximation that update the policy and value function only locally. Regarding the impact of hyperparameters, it has been experienced that these can be very limiting in the design, performance, and analysis of both the learning and safety schemes. Considerable effort must be spent to obtain adequate settings for neural network training and the safety filter. This is seen as a potential weakness of reinforcement learning based strategies compared to the more traditional approaches to flight control design, and therefore requires further investigation.

Finally, the safe curriculum optimal control framework can be further explored by implementing it on more demanding control tasks. In this thesis, most of the design and analysis has been based on simple cascaded mass-spring-damper systems. Although a proof-of-concept has been demonstrated in the context of optimal attitude control of a quadrotor UAV, more elaborate analyses need to be performed to further demonstrate the framework on this platform. Moreover, as a result of the selected safe state space, the dynamics of the quadrotor remained very close to linear. Further research can focus on increased nonlinearities of both the internal dynamics and control effectiveness. Another aspect worthwhile investigating is the effect of rotor saturation. Finally, the safe curriculum paradigm can be explored in more high-level tasks such as navigation. It is expected that its properties can be beneficial for this class of applications as well.

Bibliography

[1] Pieter Abbeel and Andrew Y. Ng. Exploration and Apprenticeship Learning in Reinforcement Learning. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 1–8, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5. doi: 10.1145/1102351.1102352.

[2] Pieter Abbeel, Morgan Quigley, and Andrew Y. Ng. Using Inaccurate Models in Reinforcement Learning. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 1–8, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2. doi: 10.1145/1143844.1143845.

[3] Paul Acquatella, E van Kampen, and Qi Ping Chu. Incremental backstepping for robust nonlinear flight control. *Proceedings of the EuroGNC 2013*, 2013.

[4] 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*, pages 1424–1431, Dec 2014. doi: 10.1109/CDC.2014.7039601.

[5] A. Al-Tamimi, F. L. Lewis, and M. Abu-Khalaf. Discrete-Time Nonlinear HJB Solution Using Approximate Dynamic Programming: Convergence Proof. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(4):943–949, Aug 2008. doi: 10.1109/TSMCB.2008.926614.

[6] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul F. Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *ArXiv*, abs/1606.06565, 2016.

[7] Anil Aswani, Humberto Gonzalez, S. Shankar Sastry, and Claire Tomlin. Provably Safe and Robust Learning-based Model Predictive Control. *Automatica*, 49(5):1216–1226, May 2013. doi: 10.1016/j.automatica.2013.02.003.

[8] Richard E. Bellman. *Dynamic Programming*. Princeton University press, 1957.

[9] Hamid Benbrahim and Judy A. Franklin. Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems*, 22(3):283 – 302, 1997. doi: https://doi.org/10.1016/S0921-8890(97)00043-2. Robot Learning: The New Wave.

[10] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, 2009. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553380.

[11] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause. Safe model-based reinforcement learning with stability guarantees. In *Proceedings of Neural Information Processing Systems*, pages 908–918, 2017.

[12] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2nd edition, 2000. ISBN 1886529094.

[13] Dimitri P. Bertsekas. Dynamic Programming and Suboptimal Control: A Survey from ADP to MPC*. *European Journal of Control*, 11(4):310 – 334, 2005. doi: https://doi.org/10.3166/ejc.11.310-334.

[14] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1st edition, 1996. ISBN 1886529108.

[15] H. W. Boschloo, T. M. Lam, M. Mulder, and M. M. van Paassen. Collision avoidance for a remotely-operated helicopter using haptic feedback. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, volume 1, pages 229–235 vol.1, Oct 2004. doi: 10.1109/ICSMC.2004.1398302.

[16] Steven J. Bradtke and Andrew G. Barto. Linear Least-Squares algorithms for temporal difference learning. *Machine Learning*, 22(1):33–57, Mar 1996. doi: 10.1007/BF00114723.

[17] S. Braun, D. Neil, and S. Liu. A curriculum learning method for improved noise robustness in automatic speech recognition. In *2017 25th European Signal Processing Conference (EUSIPCO)*, pages 548–552, Aug 2017. doi: 10.23919/EUSIPCO.2017.8081267.

[18] L. Busoniu, R. Babuska, D. Schutter, and D. Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. Automation and Control Engineering. CRC Press, Inc., 2010. ISBN 9781351833820.

[19] G.C.H.E. de Croon, M. Perçin, B.D.W. Remes, R. Ruijsink, and C. De Wagter. *The DelFly: Design, Aerodynamics, and Artificial Intelligence of a Flapping Wing Robot*. Springer Netherlands, 2015. ISBN 9789401792080.

[20] Kenji Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, January 2000. doi: 10.1162/089976600300015961.

[21] Randy A. Freeman and Petar V. Kototovic. *Robust Nonlinear Control Design: State-space and Lyapunov Techniques*. Birkhauser Boston Inc., Cambridge, MA, USA, 1996. ISBN 0-8176-3930-6.

[22] Javier García and Fernando Fernández. Safe Exploration of State and Action Spaces in Reinforcement Learning. *Journal of Artificial Intelligence Research*, 45(1):515–564, September 2012.

[23] Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(42):1437–1480, 2015.

[24] Clement Gehring and Doina Precup. Smart Exploration in Reinforcement Learning Using Absolute Temporal Difference Errors. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 1037–1044, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-1993-5.

[25] Peter Geibel. *Risk-Sensitive Approaches for Reinforcement Learning*. Shaker-Verlag, 2006.

[26] J. H. Gillula and C. J. Tomlin. Guaranteed Safe Online Learning via Reachability: tracking a ground target using a quadrotor. In *2012 IEEE International Conference on Robotics and Automation*, pages 2723–2730, May 2012. doi: 10.1109/ICRA.2012.6225136.

[27] 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*, pages 2979–2984, Sept 2011. doi: 10.1109/IROS.2011.6095101.

[28] Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1311–1320, 2017.

[29] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska. A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, Nov 2012. doi: 10.1109/TSMCC.2012.2218595.

[30] I. Grondman, M. Vaandrager, L. Busoniu, R. Babuska, and E. Schuitema. Efficient Model Learning Methods for Actor–Critic Control. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(3):591–602, June 2012. doi: 10.1109/TSMCB.2011.2170565.

[31] Guang-Hong Yang, J. Lam, and Jianliang Wang. Reliable H_∞ control for affine nonlinear systems. *IEEE Transactions on Automatic Control*, 43(8):1112–1117, Aug 1998. ISSN 0018-9286. doi: 10.1109/9.704984.

[32] Alexander Hans, Daniel Schneegaß, Anton Maximilian Schäfer, and Steffen Udluft. Safe Exploration for Reinforcement Learning. *Proceedings of the 16th European Symposium on Artificial Neural Networks*, pages 143–148, 2008.

[33] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin A. Riedmiller, and David Silver. Emergence of Locomotion Behaviours in Rich Environments. *CoRR*, abs/1707.02286, 2017. URL `http://arxiv.org/abs/1707.02286`.

[34] David Held, Zoe McCarthy, Michael Zhang, Fred Shentu, and Pieter Abbeel. Probabilistically Safe Policy Transfer. *CoRR*, abs/1705.05394, 2017. URL `http://arxiv.org/abs/1705.05394`.

[35] Alexander Helmer, Coen C. de Visser, and E van Kampen. Flexible heuristic dynamic programming for reinforcement learning in quad-rotors. In *2018 AIAA Information Systems-AIAA Infotech @ Aerospace*, 2018. doi: 10.2514/6.2018-2134.

[36] Lu Jiang, Deyu Meng, Shoou-I Yu, Zhenzhong Lan, Shiguang Shan, and Alexander Hauptmann. Self-paced learning with diversity. In *Advances in Neural Information Processing Systems 27*, pages 2078–2086. 2014.

[37] Lu Jiang, Deyu Meng, Qian Zhao, Shiguang Shan, and Alexander G. Hauptmann. Self-paced curriculum learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 2694–2700. AAAI Press, 2015.

[38] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4(1):237–285, May 1996.

[39] P. P. Khargonekar, I. R. Petersen, and K. Zhou. Robust stabilization of uncertain linear systems: quadratic stabilizability and h/sup infinity / control theory. *IEEE Transactions on Automatic Control*, 35(3):356–361, March 1990. doi: 10.1109/9.50357.

[40] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 500–505, March 1985. doi: 10.1109/ROBOT.1985.1087247.

[41] B. Kiumarsi and F. L. Lewis. Actor–Critic-Based Optimal Tracking for Partially Unknown Nonlinear Discrete-Time Systems. *IEEE Transactions on Neural Networks and Learning Systems*, 26(1):140–151, Jan 2015. doi: 10.1109/TNNLS.2014.2358227.

[42] B. Kiumarsi, F. L. Lewis, M. Naghibi-Sistani, and A. Karimpour. Optimal Tracking Control of Unknown Discrete-Time Linear Systems Using Input-Output Measured Data. *IEEE Transactions on Cybernetics*, 45(12):2770–2779, Dec 2015. doi: 10.1109/TCYB.2014.2384016.

[43] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis. Optimal and Autonomous Control Using Reinforcement Learning: A Survey. *IEEE Transactions on Neural Networks and Learning Systems*, 29(6): 2042–2062, June 2018. doi: 10.1109/TNNLS.2017.2773458.

[44] Bahare Kiumarsi, Frank L. Lewis, Hamidreza Modares, Ali Karimpour, and Mohammad-Bagher Naghibi-Sistani. Reinforcement Q-learning for optimal tracking control of linear discrete-time systems with unknown dynamics. *Automatica*, 50(4):1167 – 1175, 2014. doi: https://doi.org/10.1016/j.automatica.2014.02.015.

[45] Bahare Kiumarsi, Hamidreza Modares, and Frank L Lewis. Optimal tracking control of uncertain systems: On-policy and off-policy reinforcement learning approaches. In *Control of Complex Systems*, pages 165–186. Elsevier, 2016.

[46] B. Kiumarsi-Khomartash, F. L. Lewis, M. Naghibi-Sistani, and A. Karimpour. Optimal tracking control for linear discrete-time systems using reinforcement learning. In *52nd IEEE Conference on Decision and Control*, pages 3845–3850, Dec 2013. doi: 10.1109/CDC.2013.6760476.

[47] Vladislav Klein and Eugene A Morelli. *Aircraft system identification: theory and practice*. American Institute of Aeronautics and Astronautics Reston, VA, 2006.

[48] Torsten Koller, Felix Berkenkamp, Matteo Turchetta, and Andreas Krause. Learning-based Model Predictive Control for Safe Exploration and Reinforcement Learning. *CoRR*, abs/1803.08287, 2018. URL `http://arxiv.org/abs/1803.08287`.

[49] George Konidaris and Andrew Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 489–496. ACM, 2006.

[50] B.H. Krogh. *A Generalized Potential Field Approach to Obstacle Avoidance Control*. Creative manufacturing engineering program. RI/SME, 1984.

[51] M. P. Kumar, Benjamin Packer, and Daphne Koller. Self-Paced Learning for Latent Variable Models. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1189–1197. Curran Associates, Inc., 2010.

[52] Michail G. Lagoudakis and Michael L. Littman. Algorithm Selection Using Reinforcement Learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 511–518, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-707-2.

[53] Michail G. Lagoudakis, Ronald Parr, and Michael L. Littman. Least-Squares Methods in Reinforcement Learning for Control. In Ioannis P. Vlahavas and Constantine D. Spyropoulos, editors, *Methods and Applications of Artificial Intelligence*, pages 249–260, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-46014-5.

[54] T. M. Lam, H. W. Boschloo, M. Mulder, and M. M. van Paassen. Artificial Force Field for Haptic Feedback in UAV Teleoperation. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 39(6):1316–1330, Nov 2009. doi: 10.1109/TSMCA.2009.2028239.

[55] Alessandro Lazaric. *Transfer in Reinforcement Learning: A Framework and a Survey*, pages 143–173. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-27645-3. doi: 10.1007/978-3-642-27645-3_5.

[56] J. Y. Lee, J. B. Park, and Y. H. Choi. Integral Reinforcement Learning for Continuous-Time Input-Affine Nonlinear Systems With Simultaneous Invariant Explorations. *IEEE Transactions on Neural Networks and Learning Systems*, 26(5):916–932, May 2015. doi: 10.1109/TNNLS.2014.2328590.

[57] Sergey Levine and Pieter Abbeel. Learning dynamic manipulation skills under unknown dynamics with guided policy search. *Advances in Neural Information Processing Systems*, 27:1, 2014.

[58] F. L. Lewis and D. Vrabie. Reinforcement learning and adaptive dynamic programming for feedback control. *IEEE Circuits and Systems Magazine*, 9(3):32–50, Third 2009. doi: 10.1109/MCAS.2009.933854.

[59] F.L. Lewis, D. Vrabie, and V.L. Syrmos. *Optimal Control*. EngineeringPro collection. Wiley, 2012. ISBN 9781118122723.

[60] D. Liu, Q. Wei, D. Wang, X. Yang, and H. Li. *Adaptive Dynamic Programming with Applications in Optimal Control*. Advances in Industrial Control. Springer International Publishing, 2017. ISBN 9783319508153.

[61] Derong Liu, Qinglai Wei, Ding Wang, Xiong Yang, and Hongliang Li. *Adaptive dynamic programming with applications in optimal control*. Springer, 2017.

[62] Jie Lu, Vahid Behbood, Peng Hao, Hua Zuo, Shan Xue, and Guangquan Zhang. Transfer learning using computational intelligence: A survey. *Knowledge-Based Systems*, 80:14 – 23, 2015. doi: https://doi.org/10.1016/j.knosys.2015.01.010. 25th anniversary of Knowledge-Based Systems.

[63] B. Luo, D. Liu, T. Huang, and D. Wang. Model-Free Optimal Tracking Control via Critic-Only Q-Learning. *IEEE Transactions on Neural Networks and Learning Systems*, 27(10):2134–2144, Oct 2016. doi: 10.1109/TNNLS.2016.2585520.

[64] B. Luo, D. Liu, and H. Wu. Adaptive Constrained Optimal Control Design for Data-Based Nonlinear Discrete-Time Systems With Critic-Only Structure. *IEEE Transactions on Neural Networks and Learning Systems*, 29(6):2099–2111, June 2018. doi: 10.1109/TNNLS.2017.2751018.

[65] T Mannucci. *Safe Online Robust Exploration for Reinforcement Learning Control*. PhD thesis, Delft University of Technology, 2017.

[66] T. Mannucci, E. Van Kampen, C.C. de Visser, and Q.P. Chu. SHERPA: a safe exploration algorithm for Reinforcement Learning controllers. *AIAA Guidance, Navigation, and Control Conference*, (February), 2015. doi: 10.2514/6.2015-1757.

[67] T. Mannucci, E. Van Kampen, C.C. de Visser, and Q.P. Chu. Graph based dynamic policy for UAV navigation. In *AIAA Guidance, Navigation, and Control Conference*, AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, jan 2016. doi: doi:10.2514/6.2016-1144.

[68] T. Mannucci, E. Van Kampen, C.C. de Visser, and Q.P. Chu. Hierarchically Structured Controllers for Safe UAV Reinforcement Learning Applications. In *AIAA Information Systems-AIAA Infotech @ Aerospace*, AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, jan 2017. doi: doi:10.2514/6.2017-0791.

[69] T. Mannucci, E. Van Kampen, C.C. de Visser, and Q.P. Chu. Safe and Autonomous UAV Navigation using Graph Policies. In *AIAA Information Systems-AIAA Infotech @ Aerospace*, AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, jan 2017. doi: doi:10.2514/6.2017-1750.

[70] T. Mannucci, E. van Kampen, C. de Visser, and Q. Chu. Safe Exploration Algorithms for Reinforcement Learning Controllers. *IEEE Transactions on Neural Networks and Learning Systems*, 29(4):1069–1081, April 2018. doi: 10.1109/TNNLS.2017.2654539.

[71] I. M. Mitchell, A. M. Bayen, and C. J. Tomlin. A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on Automatic Control*, 50(7):947–957, July 2005. doi: 10.1109/TAC.2005.851439.

[72] Teodor Mihai Moldovan and Pieter Abbeel. Safe Exploration in Markov Decision Processes. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pages 1451–1458, USA, 2012. Omnipress. ISBN 978-1-4503-1285-1.

[73] Sanmit Narvekar, Jivko Sinapov, Matteo Leonetti, and Peter Stone. Source Task Creation for Curriculum Learning. In *Proceedings of the 2016 International Conference on Autonomous Agents 38; Multiagent Systems*, AAMAS '16, pages 566–574, Richland, SC, 2016. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-4239-1.

[74] Sanmit Narvekar, Jivko Sinapov, and Peter Stone. Autonomous Task Sequencing for Customized Curriculum Design in Reinforcement Learning. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 2536–2542, 2017. doi: 10.24963/ijcai.2017/353.

[75] A. Nedić and D. P. Bertsekas. Least Squares Policy Evaluation Algorithms with Linear Function Approximation. *Discrete Event Dynamic Systems*, 13(1):79–110, Jan 2003. doi: 10.1023/A:1022192903948.

[76] Andrew Y Ng. *Shaping and policy search in reinforcement learning*. PhD thesis, University of California, Berkeley, 2003.

[77] Michael A Niestroy, Kenneth M Dorsett, and Katherine Markstein. A Tailless Fighter Aircraft Model for Control-Related Research and Development. In *AIAA Modeling and Simulation Technologies Conference*, AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, jan 2017. doi: doi:10.2514/6.2017-1757.

[78] Martin Pecka and Tomas Svoboda. Safe Exploration Techniques for Reinforcement Learning – An Overview. *Lecture Notes in Computer Science*, pages 1–19, 2014. doi: 10.1007/978-3-319-13823-7.

[79] Theodore J. Perkins and Andrew G. Barto. Lyapunov Design for Safe Reinforcement Learning. *Journal of Machine Learning Research*, 3:803–832, March 2003. doi: 10.1162/jmlr.2003.3.4-5.803.

[80] Warren B. Powell. What you should know about approximate dynamic programming. *Naval Research Logistics (NRL)*, 56(3):239–249, 2009. doi: 10.1002/nav.20347.

[81] D. V. Prokhorov and D. C. Wunsch. Adaptive critic designs. *IEEE Transactions on Neural Networks*, 8(5): 997–1007, Sept 1997. doi: 10.1109/72.623201.

[82] Z. Ren, D. Dong, H. Li, and C. Chen. Self-Paced Prioritized Curriculum Learning With Coverage Penalty in Deep Reinforcement Learning. *IEEE Transactions on Neural Networks and Learning Systems*, 29(6): 2216–2226, June 2018. doi: 10.1109/TNNLS.2018.2790981.

[83] Michael T Rosenstein and Andrew G Barto. Supervised learning combined with an actor-critic architecture. *Department of Computer Science, University of Massachusetts, Tech. Rep*, pages 02–41, 2002.

[84] Michael T Rosenstein and Andrew G Barto. Reinforcement learning with supervision by a stable controller. In *Proceedings of the American Control Conference*, volume 5, pages 4517–4522, 2004. doi: 10.1109/ACC.2004.182663.

[85] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85 – 117, 2015. doi: https://doi.org/10.1016/j.neunet.2014.09.003.

[86] Harm Seijen and Rich Sutton. True online TD (lambda). In *International Conference on Machine Learning*, pages 692–700, 2014.

[87] Oliver G Selfridge, Richard S Sutton, and Andrew G Barto. Training and tracking in robotics. In *IJCAI*, pages 670–672, 1985.

[88] J. Si and Yu-Tsung Wang. Online learning control by association and reinforcement. *IEEE Transactions on Neural Networks*, 12(2):264–276, March 2001. doi: 10.1109/72.914523.

[89] J. Si, A.G. Barto, W.B. Powell, and D. Wunsch. *Handbook of Learning and Approximate Dynamic Programming*. Wiley, 2004.

[90] S. Sieberling, Q. P. Chu, and J. A. Mulder. Robust flight control using incremental nonlinear dynamic inversion and angular acceleration prediction. *Journal of Guidance, Control, and Dynamics*, 33(6): 1732–1742, 2010. doi: 10.2514/1.49978.

[91] B.F. Skinner. Reinforcement today. *American Psychologist*, 13(3):94–99, 1958. doi: 10.1037/h0049039.

[92] J. Slotine and W. Li. *Applied nonlinear control*. Prentice Hall, Englewood Cliffs, NJ, 1991.

[93] L. Sonneveldt, Q.P. Chu, and J.A. Mulder. Nonlinear Flight Control Design Using Constrained Adaptive Backstepping. *Journal of Guidance, Control, and Dynamics*, 30(2):322–336, mar 2007. doi: 10.2514/1.25834.

[94] B.L. Stevens, F.L. Lewis, and E.N. Johnson. *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*. Wiley, 2015. ISBN 9781118870976.

[95] R. S. Sutton, A. G. Barto, and R. J. Williams. Reinforcement learning is direct adaptive optimal control. *IEEE Control Systems Magazine*, 12(2):19–22, April 1992. doi: 10.1109/37.126844.

[96] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

[97] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

[98] R.S. Sutton, A.G. Barto, and F. Bach. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, 2018. ISBN 9780262039246.

[99] Csaba Szepesvári. Algorithms for Reinforcement Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 4(1):1–103, 2010.

[100] Matthew E. Taylor. Assisting transfer-enabled machine learning algorithms: Leveraging human knowledge for curriculum design. In *Agents that Learn from Human Teachers, Papers from the 2009 AAAI Spring Symposium, Technical Report SS-09-01, Stanford, California, USA, March 23-25, 2009*, pages 141–143, 2009.

[101] Matthew E. Taylor and Peter Stone. Representation Transfer for Reinforcement Learning. In *AAAI 2007 Fall Symposium on Computational Approaches to Representation Change during Learning and Development*, pages 78–85, November 2007.

[102] Matthew E. Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1):2125–2167, 2007.

[103] Matthew E Taylor, Nicholas K Jong, and Peter Stone. Transferring instances for model-based reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 488–505. Springer, 2008.

[104] Matthew E Taylor, Gregory Kuhlmann, and Peter Stone. Autonomous transfer for reinforcement learning. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 283–290. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[105] M.E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009.

[106] Sebastian B. Thrun and Knut Möller. Active exploration in dynamic environments. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 531–538. Morgan-Kaufmann, 1992.

[107] K. Vamvoudakis and S. Jagannathan. *Control of Complex Systems: Theory and Applications*. Elsevier Science, 2016. ISBN 9780128054376.

[108] E. van Kampen, Q.P. Chu, and J.A. Mulder. Continuous Adaptive Critic Flight Control Aided with Approximated Plant Dynamics. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*. 2006. doi: 10.2514/6.2006-6429.

[109] R.J. Veillette, J.B. Medanic, and W.R. Perkins. Design of reliable control systems. *IEEE transactions on automatic control*, 37(3):290–304, 1992.

[110] F. Wang, H. Zhang, and D. Liu. Adaptive dynamic programming: An introduction. *IEEE Computational Intelligence Magazine*, 4(2):39–47, May 2009. doi: 10.1109/MCI.2009.932261.

[111] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. doi: 10.1007/BF00992698.

[112] Paul Werbos. Approximate dynamic programming for realtime control and neural modelling. *Handbook of intelligent control: neural, fuzzy and adaptive approaches*, pages 493–525, 1992.

[113] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[114] Li-Juan Xie, Guang-rong Xie, Huan-wen Chen, and Xiao-Li Li. Solution to reinforcement learning problems with artificial potential field. *Journal of Central South University of Technology*, 15(4):552–557, 2008. doi: 10.1007/s11771-008-0104-x.

[115] H. Zhang, D. Liu, Y. Luo, and D. Wang. *Adaptive Dynamic Programming for Control: Algorithms and Stability*. Communications and Control Engineering. Springer London, 2013. ISBN 9781447147572.

[116] Y. Zhou. *Online reinforcement learning control for aerospace systems*. PhD thesis, Delft University of Technology, 2018.

[117] Y. Zhou, E. van Kampen, and Q.P. Chu. Incremental model based heuristic dynamic programming for nonlinear adaptive flight control. In *Proceedings of the International Micro Air Vehicles Conference and Competition 2016, Beijing, China*, 2016.

[118] Y. Zhou, E. van Kampen, and Q.P. Chu. Incremental Approximate Dynamic Programming for Nonlinear Adaptive Tracking Control with Partial Observability. *Journal of Guidance, Control, and Dynamics*, pages 1–14, oct 2018. doi: 10.2514/1.G003472.

[119] Y. Zhou, E. van Kampen, and Q.P. Chu. Incremental model based online dual heuristic programming for nonlinear adaptive control. *Control Engineering Practice*, 73:13–25, 2018.