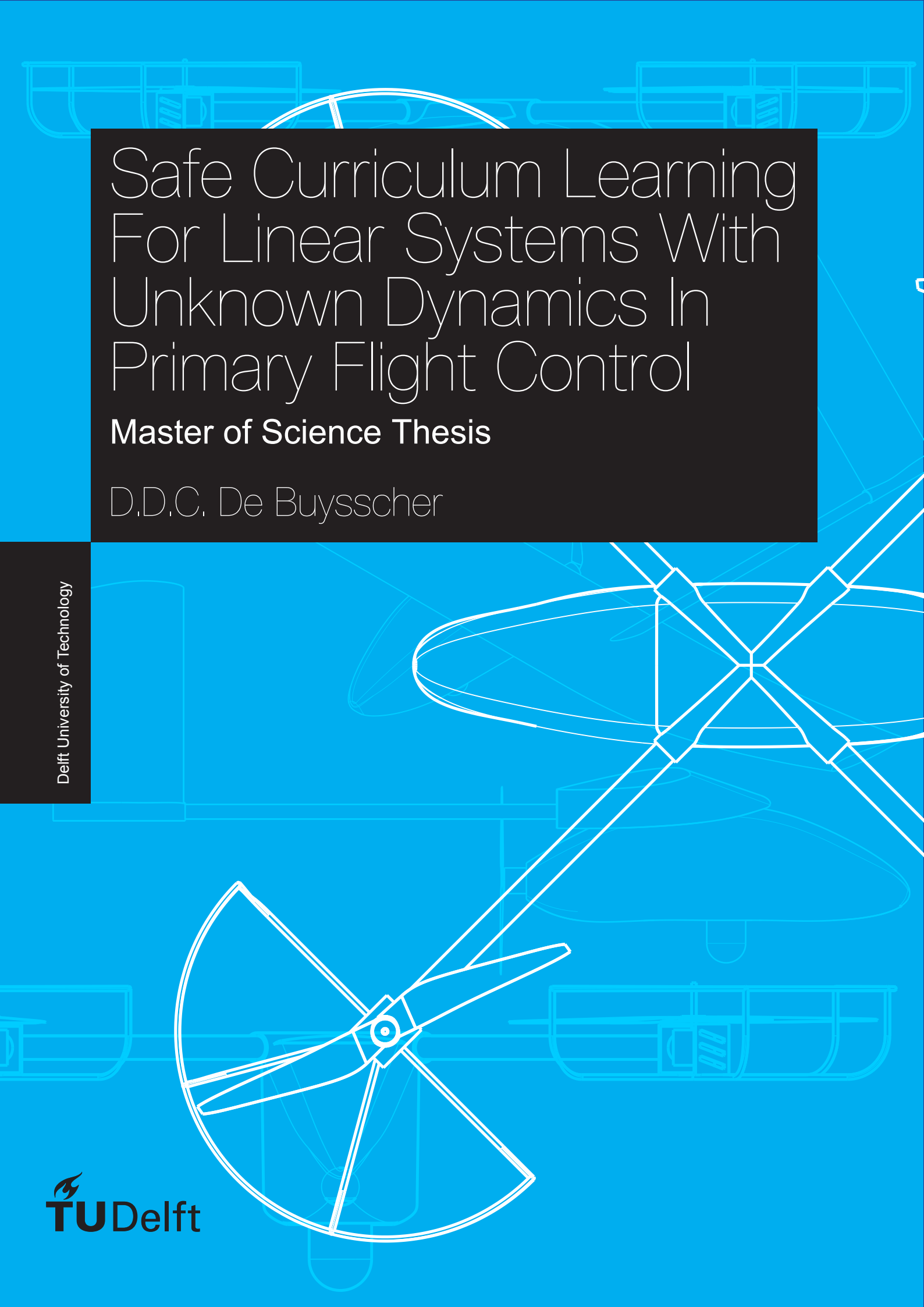
Safe Curriculum Learning For Linear Systems With Unknown Dynamics In Primary Flight Control

Master of Science Thesis

D.D.C. De Buysscher

Delft University of Technology



TU Delft

Safe Curriculum Learning For Linear Systems With Unknown Dynamics In Primary Flight Control

Master of Science Thesis

by

D.D.C. De Buysscher

To obtain degree of Master of Science
at the TU Delft University of Technology

to be presented publicly on Friday February 12, 2021 at 1:00 PM.

Student number:	4394143	
Thesis committee:	Prof. dr. G.C.H.E. de Croon	TU Delft, Chairman
	Dr. ir. E. van Kampen	TU Delft, Supervisor
	Dr. ir. E. Mooij	TU Delft, External Examiner
	ir. T.S.C. Pollack	TU Delft, Supervisor

TU Delft Delft University of Technology

Preface

The thesis project presented in this report concludes my studies at the TU Delft University of Technology. The past five and a half years have been an interesting mix of academic discoveries and memorable times with friends and Forze dream team colleagues. I would like to express my gratitude towards my daily supervisors, Dr. ir. E. Van Kampen and ir. T. Pollack, without whom this thesis would not have been possible. Their guidance and constructive feedback allowed me to progress through a the field of machine learning applied on control, if which I was not very knowledgeable prior to the start of my thesis. Furthermore, I extend my gratitude to my good friends, Anthony, Nicolas, Gervase, Charles, Sébastien and Philippe for their continuous and tremendous support along the way in my studies and this project specifically. Finally, not least important, I would like to thank my family for their unconditional support throughout my studies, in both the good and bad moments. They are the drive behind my successful completion of these tedious studies and my numerous projects along the way.

D.D.C. De Buysscher
January, 2021

Contents

List of Figures

List of Tables

List of Symbols

Abbreviations

ANN	Artificial Neural Network
ARE	Algebraic Ricatti Equation
CurL	Curriculum Learning
DARE	Discrete-time Algebraic Ricatti Equation
DDPG	Deep Deterministic Policy Gradient
DP	Dynamic Programming
FSS	Fatal State Space
HJB	Hamilton-Jacobi-Bellman
LFSS	Lead-to-Fatal State Space
LQ	Linear Quadratic
LQR	Linear Quadratic Regulation
LQT	Linear Quadratic Tracking
MC	Monte Carlo
MDP	Markov Decision Process
ML	Machine Learning
MSD	Mass-Spring-Damper
OLS	Ordinary Least Squares
PE	Persistent Excitation
PI	Policy Iteration
PPO	Proximal Policy Optimisation
PR-SRL	Policy Reuse Safe Reinforcement Learning
PR	Policy Reuse
RBF	Radial Basis Function
ReLU	Rectifier Linear Unit
RL	Reinforcement Learning
RSS	Restricted State Space
SGD	Stochastic Gradient Descent
SHERPA	Safety Handling Exploration with Risk Perception Algorithm
SRL	Safe Reinforcement Learning
SS	State Space

SSS	Safe State Space
TD	Temporal Difference
TL	Transfer Learning
TRPO	Trust Region Policy Optimisation
UAV	Unmanned Aerial Vehicle
VI	Value Iteration
VTOL	Vertical Take-Off and Landing

Greek symbols

Γ	Set of all non-error states	–
γ	Discout factor	–
Ω	Set of all known states	–
Φ	Set of all error states	–
π	Agent policy	–
τ	Transition between two states	–
Υ	Set of all unknown states	–

Latin symbols

$\mathcal{A}$	Set of actions available to the agent	–
a_t	Action from $\mathcal{A}$ taken by the agent at time step t	–
$\mathbb{E}_\pi$	The expected value of a random variable, conditioned to π	–
$\mathcal{R}$	Reward function	–
$\mathcal{S}$	Set of all states	–
s_t	State from $\mathcal{S}$ at time step t	–
$\mathcal{T}$	Transition function, generates transitions τ_t	–
t	Time	[s]
v_π	Value function conditioned to policy π	–

1 Introduction

The trend towards unmanned systems has increased drastically over the past years. The aerospace sector desires increasingly complex systems to become autonomous. The technology of automated unmanned systems progressively introduces itself in numerous fields, ranging from agricultural applications to lawnmowers, vacuum cleaner, and drone ships. Thus, becoming an essential part of practical applications around the globe as well as representing a profitable economic asset. The rapid spread of such technology, and the variability of its applications and requirements, demands a controller that can support such reach.

Classical control methods strongly rely on model-based methods (such as non-linear dynamic inversion controller [29]), which have a relatively high drawback of being hard to verify and validate for complex systems. Such controllers tend to take a relatively large amount of time to develop, usually months to decades, depending on the system complexity. Consequently, a problem that is becoming more and more prominent is the rising variety and complexity of applications. Indeed, designing controllers would require the construction of an adequate model, which has to be validated. Machine Learning (ML) would solve this issue by proposing a model-free solution, which generalises the control laws of the topical application. As the name suggests, the controller is learned by an agent instead of being derived by an engineer. In the past decade, a variety of Vertical Take-Off and Landing (VTOL) vehicles is developed by companies including Airbus and Uber, to transport passengers from point A to point B. Such trajectories are based on a flight plan approved by the local air traffic controller, which the vehicle would have to track. For these transportation applications, a controller optimised for tracking a reference is required. Besides, VTOL vehicles are accompanied by highly complex system dynamics. In this case, ML auto-pilots would provide a solution where system dynamics are not to be modelled. Furthermore, as it has been seen by the likes of Tesla, auto-pilots based on ML are proving their worth in driving vehicles in real life. On complex applications, however, the reinforcement learning approaches in their simplest have a more difficult task of learning the control policy. In cases where the task is seemingly too complex, curriculum learning is used to provide a more gradual learning framework for the agent. Additionally, the curriculum learning framework improves the learning efficiency and performance.

However, ML methods have been found to exhibit erratic learning behaviour in the initial stages of the learning process. Such behaviour can lead the system into inherently dangerous states, such as crashes for example. When applications mentioned previously have as goal to transport humans, lives are at stake and safety is paramount. For this reason, safety has to be considered with diligence when developing controllers that are applied to such systems. In learning, when faced with unseen cases, the controller has to act safely. To tackle this significant drawback of ML safety mechanisms are to be scrutinised to improve learning safety and operational safety.

The aim of this thesis is to set the foundation safe reinforcement learning process for controllers used on systems with unknown dynamics. Use is made of a curriculum through which the learner is paced for enhancing learning performance and efficiency.

1.1. Motivation

In this work, the analysis presented in [38], will be further elaborated to accommodate learning on systems for which no inherent system dynamics are known. Currently, safe curriculum learning has been established as a viable method to tackle primary flight control. However, the work presented in [38] requires a crude, inaccurate model of the system dynamics. Consequently, the initial advantage of model-free Reinforcement Learning (RL) is not used to its full potential. Indeed, a model, albeit in the form of a bounding model, is required for the algorithm's safety aspect provided in [38]. In the thesis for which this work serves as a foundation, the aim is to eliminate dependency on such knowledge by introducing system identification capabilities to the algorithm. Allowing for a broader range of potential applications, and most importantly, retaining the applicability on complex systems.

With safe curriculum learning for systems with unknown dynamics, the applications are about infinite. By virtue of its independence of any system dynamics, the method can be applied to any system, as long as a decent curriculum can be conceived.

1.2. Research Logistics

Previously, the utility of the research presented in this report has been outlined, with examples of potential applications. However, in order for the research to bring conclusive results, a structure is required to set the focus on the correct subjects during the literature study and to consider the correct test setup for analysis. This section will elaborate on the logistics required to build the mentioned research structure. First, a set of definitions are given to avoid ambiguities during the reading of this technical report. After which, both the research objective and the research question are presented in the research scope. These form the foundation on which the research structure will be constructed.

1.2.1. Definitions

Before the research scope can be presented, a few terms need to be defined in order to straighten the record on their assumed meaning in this report. The aim is to align these definitions with what is found within literature and to inform the reader on the interpretation of the jargon used later in this preliminary thesis.

First two types of Reinforcement learning techniques that are used throughout this literature study are defined in order to fix the scope of this study as well as the consequent thesis.

Definition 1.1: Safe Learning

Safe Learning is a learning process in which the agent does not encounter a fatal terminal state.

Definition 1.2: Curriculum Learning

Curriculum Learning is a learning process in which the agent is forced through a set of curricular steps with increasing entropy, in order to simplify the learning process.

Second, a set of definitions is given relating to the broader field of RL, settling the ambiguities that might rise from divergent literature.

Definition 1.3: Agent

The agent or learner is considered to be that what makes decisions within an environment to reach a predefined goal. The agent's scope is comprised of the system and input dynamics.

Definition 1.4: Environment

The environment contains all the dynamics that are not comprised within the agent.

Definition 1.5: State

A state is a vector that contains the set of variables which combine all necessary knowledge to represent the condition of an agent within an environment unambiguously.

Definition 1.6: Exploration

Exploration is defined as the agent taking non-optimal actions at a given state in the hope to find a more optimal policy in the long run.

Definition 1.7: Exploitation

Exploitation is the act of using solely the current understanding of the system in order to make an optimal decision by the agent.

In definitions 1.6 and 1.7 both *optimal* and *non-optimal* actions refer to actions that are optimal according to the agent's policy, throughout the learning process. As such, the optimality of the actions is not a reference to the true optimal policy. Indeed, as stated in definition 1.6, when non-optimal action is taken, that action would not have been generated by the agent's current policy. When an optimal action is taken, the action is found to be optimal according to the agent's current understanding of the environment.

Finally, two definitions (definitions 1.8 and 1.9) are used to consolidate the understanding of their respective concepts. The concept of unknown dynamics laid out in definition 1.8 is derived from literature [22, 23, 27].

Definition 1.8: Unknown Dynamics

A system with unknown dynamics is a system for which the input and output representations (in the form of dimensions) are known, but knowledge of inherent system dynamics is nonexistent.

Definition 1.9: Model-Free

Model-free methods inherently do not use any form of model representation (true or approximate).

1.2.2. Research Scope

Once the meaning behind the cornerstone principles of this research has been established in the previous subsection by defining them, the research scope is set in this subsection. The scope is laid out by means of a research objective and a research question. The latter is then further divided in sub-questions to facilitate the focus of the literature study and experiments to be performed during the analysis.

First the objective of the research is defined. Based on the arguments given in the introduction of this chapter and the section containing the motivation, the objective is focused on complementing a safe curriculum learning method with system identification. More specifically to provide the safety aspect of the learning process with independence from knowledge of system dynamics. The research objective is stated definitively in the box below.

Research Objective

Adapt Safe Curriculum learning to be applicable on systems with unknown dynamics by complementing it with a system identification method.

Furthermore, no research is complete without a research question to provide direction in finding literature on the subjects required for this work. The main research question is given in the box below.

Research Question

How can a curriculum be implemented such that it incorporates efficient learning of the control policy and the system dynamics in a safe manner?

Answering this question would require a broad variety of literature. Hence, the focus of the research has to be limited in order to fit within the given time frame of 9 months. In this chapter, multiple concepts have appeared consecutively, which will form the pillars of the research. There are five pillars to this research: reinforcement learning, curriculum learning, safe learning, system identification and system control. These pillars form the basis of the division in sub-questions, which reflect the scope that is targeted for each pillar. The first pillar is RL, which is to be investigated for its ability to control systems, complex or not.

Second, there is the controlling aspect of the research. Indeed, the ultimate goal is to have a controller for a system. The scope for control has been set to determine a way to incorporate a tracking task in the RL algorithm. As has been mentioned previously in this chapter, tracking tasks can be correlated to following a flight path in air spaces, or even following set trajectories in obstacle avoidance applications.

Next, the curricular aspect has to be examined. Combining RL and curricula is called curriculum learning. Therefore, the purpose of the third research sub-question and its respective sub-sub-questions is to evaluate the defining characteristics of curriculum learning. That includes the underlying theory of what a curriculum constitutes, as well as curricular strategies and the way the curriculum is related to RL.

Regarding the safety pillar, the main research question does not specify the definition of safety, nor the way it can be tackled in harmony with RL. A series of sub-sub-questions do aim at investigating these subjects more thoroughly. They do so by finding the current trends in the field and search for the methods depending on system identification. The objective is to find out which methods would benefit from a model provided by a system identification capability. Which leads to the last pillar, founded in the field of system identification. It is of the highest interest of this research to have highly effective methods for producing models with the highest possible accuracy. This pillar is founded in two principal objectives, namely finding state-of-the-art model constructs and discovering the methods used to tune these constructs.

Research Sub-Questions

1. How is Reinforcement Learning advantageous for complex systems?
2. How can a tracking task be implemented with reinforcement learning?
3. How is an effective curriculum for a reinforcement learning agent designed?
   - 3.1 What are the key aspects defining a curriculum?
   - 3.2 What are the current strategies for RL curricula?
   - 3.3 How are curricula imposed on RL agents?
4. How does safe learning impact the efficiency of curriculum learning?
   - 4.1 What does safe learning entail?
   - 4.2 What are the current trends in safe learning?
   - 4.3 Which, if any, safe learning methods rely (partially) on knowledge of system dynamics?
5. How are system dynamics modeled when they are unknown?
   - 5.1 What are the state-of-the-art model constructs?
   - 5.2 What system identification methods can be combined with Reinforcement learning?

The division of the main research question presented in the box above, provides a more gradual incentive to collect answers to the broader research scope. It links the subjects required for the research at lower levels before combining them in an ultimate answer. The questions presented in this box are answered individually in the conclusion chapter of this report.

1.3. Report Structure

On the highest level, the preliminary thesis is structured in two parts. The first part outlines a literature study performed on the five pillars founding this research. With aim to capture literature to answer the aforementioned sub-questions. The second part contains a preliminary analysis. In this analysis the methods found in each pillar are combined to form the algorithm proposed by this work.

The schematic presented in Figure 1.1 outlines the structure of the algorithm. It can be seen that the curriculum contains multiple curricular steps which are in essence instances of an RL agent-environment interaction with a safety mechanism attached to it. The latter contains a system identification block to assist with the reliance on system dynamics. The schematic has helped formulate the structure of this report.

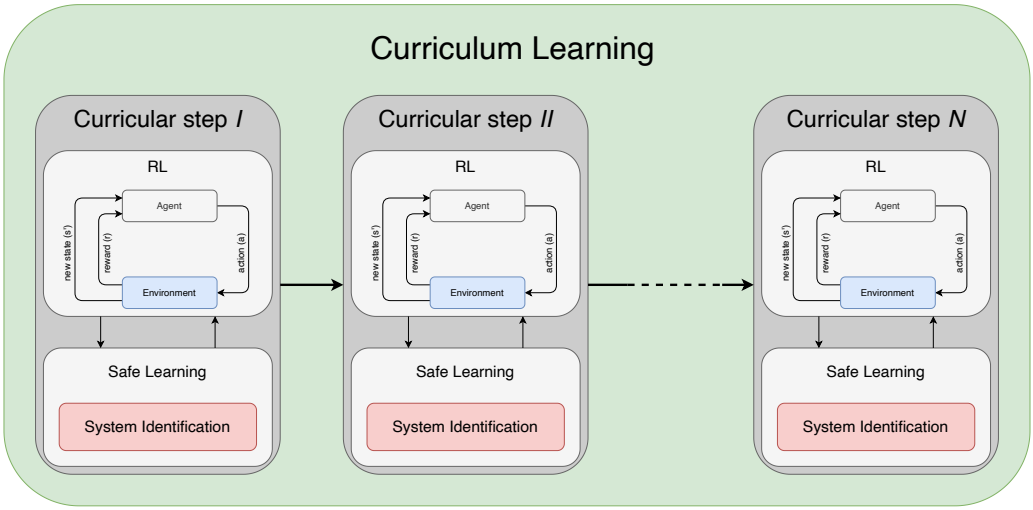


Figure 1.1: A schematic overview of the thesis workflow

Part I[1] of this report contains the article containing the final analysis in which the proposed architecture is tested on quadrotor flight dynamics. This is followed by the literature study in Part II, which is structured according to the chapters' dependency on knowledge gathered in other chapters. The basis of this research lies in the understanding of RL and its underlying principles. The basic algorithms and concepts of RL are presented in Chapter 2. In a second instance, the fundamental principles behind the mix of RL and a tracking task are investigated. Chapter 3 places tracking tasks in combination with RL on the more extensive map that is the field of control. Detailing how an RL agent can be stimulated to learn a policy based on the Linear Quadratic (LQ) cost function. The combination of Chapters 2 and 3 creates the necessary understanding to construct an RL interaction onto which safety algorithms can be added. However, it is essential to evaluate if the interaction found thus far can be integrated within a curriculum. For this, the inner workings of curriculum learning have to be examined. Described in Chapter 4, are the mathematical conditions defining curricula. Furthermore, this chapter contains an overview of the methods used to transfer knowledge between curricular steps.

To complete the safe curriculum learning algorithm, the safety aspect must be added to the RL agent-environment interaction. In Chapter 5, the concept of safe learning is examined alongside current trends and methods. A set of methods within these trends is then identified as being dependent on external knowledge, such as (uncertain) system dynamics. Finally, system identification is investigated in Chapter 6. With as objective to complement the safety mechanisms identified in Chapter 5 with an uncertain model of system dynamics. Additionally, model constructs used for this purpose are being presented in Chapter 6.

A preliminary analysis is performed on a mass-spring-damper system in Part III, where the safe curriculum learning paradigm proposed in this thesis is tested for its feasibility on stable and unstable systems. Furthermore, a better understanding of the interaction between the RL agent, the safety filter and the system identification module is achieved through a sensitivity analysis of the paradigm's hyper parameters.

Finally, Part IV contains additional results discovered during the final analysis on the quadrotor, that were not important enough to be included in the article.

[1] https://github.com/DBdiego/SafeCurriculumLearning.git

I

Article

Safe Curriculum Learning for Linear Systems With Unknown Dynamics in Primary Flight Control

D.D.C De Buysscher*

Delft University of Technology, Kluyverweg 1 2629 HS Delft, the Netherlands

Safe Curriculum Learning constitutes a collection of methods that aim at enabling Reinforcement Learning (RL) algorithms on complex systems and tasks whilst considering the safety and efficiency aspect of the learning process. On the one hand, curricular reinforcement learning approaches divide the task into more gradual complexity stages to promote learning efficiency. On the other, safe learning provides a framework to consider a system's safety during the learning process. The latter's contribution is significant on safety-critical systems, such as transport vehicles where stringent (safety) requirements apply. This paper proposes a black box safe curriculum learning architecture applicable to systems with unknown dynamics. It only requires knowledge of the state and action spaces' orders for a given task and system. By adding system identification capabilities to existing safe curriculum learning paradigms, the proposed architecture successfully ensures safe learning proceedings of tracking tasks without requiring initial knowledge of internal system dynamics. More specifically, a model estimate is generated online to complement safety filters that rely on uncertain models for their safety guarantees. This research explicitly targets linearised systems with decoupled dynamics in the experiments provided in this article as proof of concept. The paradigm is initially verified on a mass-spring-damper system. After that, the architecture is applied to a quadrotor where it is able to successfully track the system's four degrees of freedom independently, namely attitude angles and altitude. The RL agent is able to safely learn an optimal policy that can track an independent reference on each degree of freedom.

Nomenclature

$\mathcal{A}$	=	Set of all possible actions	u_t	=	Action vector at time t
c	=	Damper constant $[N/ms^{-1}]$	W()	=	Probabilistic weight function X_k
f_t	=	Thrust force $[N]$	X_k	=	Augmented state at time step k
H()	=	Statistical entropy function	x_t^r	=	Reference state vector at time t
H	=	Kernel matrix	x_t	=	State vector at time t
k	=	Spring constant$[N/m]$	γ	=	Discount factor
K	=	Gain	$\hat{\theta}$	=	Model parameter estimates
m	=	Mass $[kg]$	λ	=	Curricular step index
P	=	Regression matrix	μ	=	Distribution mean
q()	=	q value function	ν_π	=	Value function conditioned on π
Q	=	State weight matrix	ξ	=	Policy mapping matrix
R	=	Input weight matrix	π	=	Agent policy
$\mathcal{R}$	=	Reward function	σ	=	Distribution standard deviation
r or c	=	Reward or cost	$\tau_\bullet$	=	Torque around $\bullet$ axis $[Nm]$
$\mathcal{S}$	=	Set of all possible states	Ω_i	=	Rotor RPM of i^{th} rotor$[RPM]$
$\mathcal{T}$	=	Transition function			

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

I. Introduction

The field of control has become increasingly important in numerous domains around the world ranging from transportation to appliances or even the agricultural sector. A common desire in these industries is to have a vehicle that is capable of tracking a pre-defined trajectory. For example, in the agri-sector, tractors follow an optimised trajectory on a field and in the transport industry, personal UAVs follow a defined flight path. The complexity of these systems along with their ever increasing variety, demands for controllers that are more flexible in their design process. Additionally, complex systems are often difficult to model and validate. Consequently, controllers that can be derived using data-driven methods are preferred in such cases. Advanced sensor-based controllers exist, such as non-linear dynamic inversion controllers [1], to alleviate the burden of modelling complex systems. However, these methods suffer from state reconstruction dependencies and synchronisation issues with the sensor data. Another approach to data driven controller design resides in machine learning. More specifically, Reinforcement Learning (RL). The fundamental principle used in RL is the representation of the world as an agent (the system) being confronted with a choice of action. The agent learns a control policy by gaining experience between the dynamics of its environment and itself over time. Its basic principle is simple yet effective. However, RL agents have a tendency to struggle learning a policy reliably when applied on complex systems [2]. Curriculum Learning (CurL) provides a structured approach to allow RL on more complex applications by dividing the initial task into sub-tasks [3, 4]. This facilitates the agent's learning process and increases the likelihood of successfully finding a control policy [5]. Given the examples cited previously, certainly in transport applications where stringent (safety) requirements apply, the safety aspect of the learning process and the correct operation of the controller is of crucial importance. Unlike RL methods which in their simplest forms generally lack consideration of the safety aspect [6], Safe Learning (SL) does provide a framework to this end.

The research outlined in this paper proposes a safe curriculum learning architecture that builds on the research by Pollack and Van Kampen [7]. Here, the dependency on knowledge about an uncertain model for the safety algorithm is removed by complementing the paradigm in [7] with a system identification capability.

First a brief introduction to the fields of RL, Curriculum Learning, Safe Learning, and system identification is provided in sections II.A, II.B, II.C and II.D, respectively. This is followed by a detailed presentation of the approach chosen in this research outlined in Section III. Finally, the proposed paradigm is tested through two experiments. Initially, a Mass-Spring-Damper (MSD) system is used to verify the architecture for which the results are presented in Section IV.A. In Section IV.B, the results of a the safe curriculum architecture applied on a quadrotor are outlined. The paper is concluded with a discussion of the results of the experiments, as well as a conclusion and recommendations for further research for which this paper may serve as a foundation.

II. Safe Curriculum Learning Framework

The core principles in safe curriculum learning are derived from three research fields: reinforcement learning, curriculum learning and safe learning. Inherently, the fundamentals originate from the more general machine learning field of study that is reinforcement learning. This learning process is then altered in curriculum and safe learning approaches.

II.A. Reinforcement Learning

Fundamentally, the concept of reinforcement learning aims at translating decision-making tasks into a logical space that is more accessible to current computational implementations. It does so by using the high-level and more general formulation contained in a Markov Decision Process (MDP). Based on Markov chains [8], the perspective used in a MDP, is that at each point in time, one is faced with a decision to act in a certain way based on the situation at hand. In terms of a MDP, the traveler is called an agent and the two possible ways to go are called actions. In this example the action space, $\mathcal{A}$, contains two actions: go left or go right. The way an agent decides whether to go left or right is dependent on the current situation or state, x, in which the agent finds itself. The set of all possible states is defined as $\mathcal{S}$. Mathematically, the decision is characterised by the policy function, $\pi(x_t) = u_t$ [6]. It should be noted that all MDP's rest on the fundamental Markov property, stating informally that the future is independent of the past given the present [9]. In essence, this property describes that all information from the past is contained in the

present observation, which is used by the policy to define the action to be taken by the agent. Once the agent has made a decision on the action to take, the action is executed and the agent is then transitioned to a new state. All the transitions between states dependent on the chosen action are defined in the transition function $\mathcal{T} : \mathcal{A} \times \mathcal{S} \to \mathcal{S}$. Finally, each transition is rated by a reward function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$, assessing the optimality of the transition and as a consequence the state-action pair, (x, u). A MDP is uniquely defined by a tuple, $\langle \mathcal{A}, \mathcal{S}, \mathcal{T}, \mathcal{R} \rangle$ containing the aforementioned spaces and functions.

Canonical RL problems aim at optimising the path between an initial state, x_{init} to the goal state, x_{goal}, by finding the optimal policy for the MDP. With the optimal policy, π^*, a series of actions which are optimal under the reward function $\mathcal{R}$, are taken by the agent to reach the goal state. To reach this optimal policy, RL performs a non-linear optimisation on a MDP. As is the case for optimisation processes, a reward function is required to guide the process towards the optimum. In this case, the reward function of the MDP is used.

In accordance with [6], RL embodies two main optimisation strategies for MDP solvers, namely, Value Iteration (VI) and Policy Iteration (PI). Both make use of the General Policy Iteration (GPI) algorithm [6]. Computational complexity is reduced when using VI compared to PI, since the policy evaluation step does not require the value function to be the true one. The true value function is reached only in the limit. The policy evaluation step is outlined in Eq. (1), where $p(x', r|x, u)$ defines the probability of a subsequent state, x', occurring with the respective reward, r. Furthermore, $\pi(u|x)$ denotes the probability of an action to be taken at a given state, x. Finally, a discount factor, γ, is used to reduce the importance of the previous value function. The other iterative step in GPI, is policy improvement (Eq. (2)), which based on the current estimate of the value function will adapt the policy.

$$\nu_{\pi}^{k+1}(x) = \sum_{u} \pi(u|x) \sum_{x'} \sum_{r} p(x', r|x, u) \left[r + \gamma \nu_{\pi}^{k}(x') \right] \tag{1}$$

$$\pi^{k+1} = \begin{cases} \pi' & if \quad \nu_{\pi'}(x) \geq \nu_{\pi^k}(x) \\ \pi^k & if \quad \nu_{\pi'}(x) < \nu_{\pi^k}(x) \end{cases} \tag{2}$$

Solving the MDP can be done using multiple methods, where the method of choice is dependent on the knowledge of the MDP available to the operator. In [6], three methods are discussed, namely Dynamic Programming, Temporal Difference and Monte Carlo. Dynamic Programming requires full knowledge of the MDP, full state space, action space, transition function and reward function. The remaining two methods are used when a perfect model of the MDP is not available. Temporal difference and Monte Carlo focus on experience driven optimisation, through rigorous sampling of visited state and actions as well as the transitions that occurred with their respective rewards. The Monte Carlo approach is derived from the statistical theorem of large numbers where the more a state is visited the better the MDP modelling of that state is. All make use of the value and policy iteration principles provided earlier in this section.

Applicability of these solvers is reflected in two common implementation of RL, namely SARSA and Q-learning [6]. These take the concept of a value function further. Instead of assessing the value of states, both implementations make use of a q-function, $q(x, u)$, which considers the value of a state-action pair. Detailed algorithmic formulations are provided in [6].

Besides the VI and PI strategies, other approaches do exist, such as Deep Deterministic Policy Gradient (DDPG) [10], Proximal Policy Optimisation (PPO) [11] and Trust Region Policy Optimisation (TRPO) [12] to name a few. The core principle behind each of these, is the use of the policy gradient to find the optimal policy of the topical MDP. Due to their more complex implementation requirements, they are not considered in this research.

II.B. Curriculum Learning

Mimicking the learning process of humans and animals on complex tasks, Curriculum Learning (CurL) provides a step-wise learning approach. Specifically, by decomposing the final task into sub-tasks it gives the RL agent a more achievable learning goal in each of them. Each subsequent curricular step uses knowledge gained in the previous one resulting in the agent learning the ultimately complex task in a more gradual manner. Specifically, CurL has been proven successful in numerous fields, such as system control and complex games [5,13]. Ng et al [5] have used curriculum learning to gradually train an RL agent to perform non-linear helicopter maneuvers. In 2016, Alpha Go defeated the – at the time – world champion in the game Go. It was trained using curriculum learning enhancements to facilitating the learning process of the complex

game [4, 13, 14]. On a high-level and abstract basis, a curriculum obeys two statistical properties [2] which are detailed in a subsequent paragraph.

The design of a curriculum has an important impact on the overall stability and convergence of the RL agent's learning process [2]. As defined by Bengio et al [2], curricula are subject to two rules (summarised in Eq. (3)) in order to ensure a progressive learning strategy. One relates to a monotonically increasing statistical entropy, $H(Q_\lambda(x))$, throughout the curriculum, increasing the diversity. The other requires a monotonically increasing weight function, $W_\lambda(x)$, to add states to the state space made available to the RL agent. In the equation below, the subscript λ denotes the curricular step index and ϵ is a positive integer added to the former to denote a subsequent curricular step.

$$\begin{aligned} &1) \quad H(Q_\lambda(x)) < H(Q_{\lambda+\epsilon}(x)) \quad \forall \epsilon > 0, \forall x \in \mathcal{S} \\ &2) \quad W_\lambda(x) \leq W_{\lambda+\epsilon}(x) \qquad\quad \forall \epsilon > 0, \forall x \in \mathcal{S} \end{aligned} \tag{3}$$

Statistical entropy is a mathematical expression for designating the amount of randomness in a statistical process. It can be seen as a metric indicating how predictable a process is. In the case of the first condition given in Eq. (3), the entropy is tied to the probability of occurrence of a certain state or observation, $Q(x)$. In essence, given a state space, the function $Q(x)$ is the probability density function of that state space. If only a few states are likely to occur, the predictability is high. By adding entropy the uncertainty of that distribution is increased and the occurrence predictability of a state or observation is reduced.

The second condition proposed in [2] and repeated in Eq. (3), relates to weighting function allocated to an observation. By increasing $W_\lambda(x)$ throughout the curriculum, the state space available in the environment is enlarged. Consequently, the probability density function flattens. Furthermore, using knowledge about the topical RL problem, more *difficult* situations can be added in a later stage of the curriculum. The aforementioned difficulty is problem dependent.

Inherently the two conditions are closely related. By increasing both the entropy and the weighting function throughout the curriculum, the diversity in the state space in increased.

Recent research [4] has found that curriculum learning approaches can be categorised in three main branches: reward shaping [15], variable task complexity and reversed problem space. All of which adapt one or multiple parts of the MDP's tuple in the design of the curriculum.

As the name suggests, reward shaping introduces variations in the reward function, $\mathcal{R}$, of the MDP. Resembling dog shaping, the agent is given higher rewards for good actions taken at a given state. Over the course of the curriculum, the bias introduced in the reward function is gradually removed, to converge towards the true reward function of the MDP. One example of the benefits of such approach is the non-linear helicopter maneuvering capabilities learned by the agent in Ng et al [5].

Gradually increasing task complexity is another approach to construct a curriculum which is more closely related to the way the academic systems are designed. Indeed, as a child, simple math and language principles are taught, after which the difficulty and, more importantly, the complexity is increased over time. The increasing difficulty is mirrored in the academic phases known as primary school, high school and university, for example. The benefit of a curriculum becomes clear when using the example of a child being taught university level courses. The learning process would be inefficient in two aspects: the amount of knowledge gained by the child and the time required for that knowledge to be assimilated. Instead, with a structured teaching approach the learning process's efficiency is increased [16].

Finally, there is the reversed problem space approach. The perspective being that the agent is initially placed close to the goal state. Throughout the curriculum, the agent is removed further away from the goal state. As such, it has to find a path to its comfort space, which it learned in previous curricular steps. Humans learn to play certain board games, such as Chess or Draughts in this manner. Intrinsically these approaches increase the size of the state space and even the action space in each curricular step. As such, they allow the conditions presented in Eq. (3) to be satisfied.

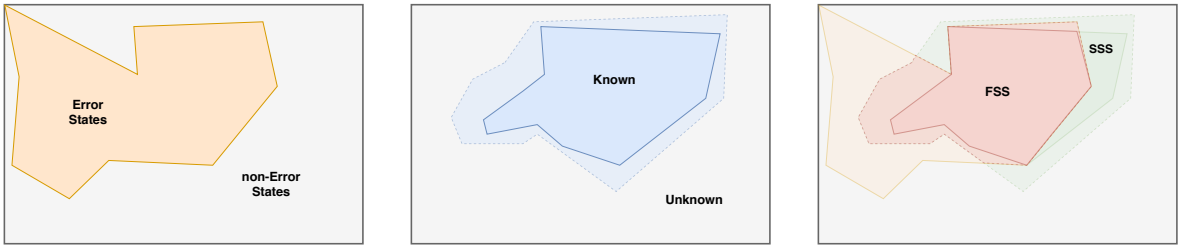
When moving from one curricular step to another, the purpose is to retain the knowledge and experience acquired thus far in the curriculum. Transferring knowledge between two sub-tasks is achieved using methods contained in Transfer Learning (TL) [17]. In RL, the gained experience and knowledge are generally contained in an agent's internal representation of the MDP. Based on the state and action space representations of the MDP by the agent, the knowledge transfer, or mapping, has to be adapted. When state and action spaces are represented equally in both curricular steps between which the knowledge transfer is to be performed, the mapping is straight forward. Indeed scaling is possible and no intricate mapping strategy is required.

However, when either one of these spaces is different, mapping strategies become more strenuous. Two ideologies arise in this field [18]. The first focuses on learning the value function, ν_π, which is transferred between source and target task [19]. The other proposes to transfer the policy learned in the previous curricular step to the target task, whilst re-iterating the value function during the new curricular step [20].

II.C. Safe learning

Reinforcement learning comes with the benefit of not being model-based. Indeed, methods such as temporal difference and Monte Carlo are able to find optimal policies without knowledge of part of the MDP. However, the safety of a system is not considered during the learning process. The safety aspect can be a crucial subject when applying reinforcement learning on real-life experiments, particularly on systems that are expensive to build or set up, such as aircraft, cars, boats or even robotic arms used in the production lines of factories [21, 22].

According to [18], safety in learning processes is attributed to either the optimisation criterion, or the exploration process. The optimisation criterion relates to the reward function of the MDP, discussed in previous section. Ensuring safe learning can indeed be addressed by adapting this criterion to be risk sensitive [18, 23]. As such, the reward function can be adapted to give lower rewards when a transition leads to an unsafe state (elaborated upon in the next paragraph). The exploration process can thus be interfered with by either providing external knowledge to the agent and guiding it away from dangerous state spaces [24], or by adding an element of risk into the exploration process. This provides the agent with a risk assessment tool enhancing its ability to chose the optimal safe action.



(a) *Error* and *Non-Error* states within the set of states $\mathcal{S}$

(b) *Known* and *Unknown* states within the set of states $\mathcal{S}$. Dashed line represents the exploration of $\mathcal{S}$ over time.

(c) *Safe State Space (SSS)* and *Fatal State Space (FSS)* within the set of states $\mathcal{S}$. Dashed line represents the exploration of $\mathcal{S}$ by the agent over time. Note that this figure is an overlap of figures 1a and 1b

Figure 1: State space segregation based on different concepts: (left) error-states, (middle) known states, (right) safe states. The grey rectangle represents the complete state space of the task at hand, $\mathcal{S}$.

During the exploration process of the state and action spaces, certain state-action pairs can pose danger to systems. In safe learning research, safety is assessed by associating a given state to a state space with certain properties. In Fig. 1, an overview is given of three approaches used to divide the state space into groups based on certain characteristics.

In a first instance, the state space system can be divided in error and non-error spaces which define the consequence of being in their respective states (see Fig. 1a) [25]. Error states are considered as harmful to the system as they either lead to system termination or to serious damage. Non-error states are considered innocuous to the system. In a second instance, the state space can be segregated based on the experience gained by the agent. Shown in Fig. 1b, states are labeled as known or unknown, depending on the agent having visited the state previously [26]. Finally, combining these concepts, the definition of a Safe State Space (SSS) and a Fatal State Space (FSS) can be given [27–29]. In a SSS, only states that are labeled non-error and known can be included. The FSS includes error states, regardless of whether the agent has visited the state previously. This is shown in Fig. 1c.

In the scope of exploratory safety, the Safety Handling Exploration with Risk Algorithm (SHERPA) and optiSHERPA introduced in [27–29] provide a risk-sensitive and model-predictive exploration method. SHERPA

makes use of the policy reuse concept outlined in [30], in which the probability of using a safe policy is related to the risk sensed at a given state. In SHERPA's terminology, this safe policy is defined as a backup policy, which relies on the ergodicity condition. Discussed in [31], an exploration method based on ergodicity goes into more detail on this regard. When an agent follows its policy, that policy is deemed safe if and only if there is the possibility to safely return to a previously visited state. The latter being part of the SSS as discussed in an earlier paragraph. The way to return to a state, $x \in SSS$, is SHERPA's backup policy. This backup policy has to satisfy a complementary condition that the trajectory to this state is within the SSS at all times. A considerable drawback of the ergodicity condition and algorithms relying on its application is its dependence on knowledge of system dynamics, albeit in the form of of a bounding model.

II.D. Safe Learning for Systems With Unknown Dynamics

Safe learning algorithms rely mainly on external knowledge, in the form of guidance during exploration or sensible adaptions of optimisation criteria [18]. Lacking the availability of system dynamics, albeit certain or uncertain, has a more significant impact on safety methods based in the exploration process of the state and action spaces by the agent. More specifically, methods such as SHERPA and optiSHERPA in which the backbone is constructed on the ergodicity condition [27–29, 31]. By consequence, these paradigms inherently rely on system and input dynamics representations to project trajectories internally. In order to solve the problem in such methods when working with a system where only the state and action orders are known, system identification techniques can be used.

Linear systems mostly make use of a state space model representation for their system and input dynamics. Moreover, their non-linear counterparts can be approximated by time-variant linear state space systems, with the exception of highly non-linear systems due to inaccuracies arising with such models. Consequently, for highly non-linear systems multivariate splines or Artificial Neural Networks (ANN) are preferred for a more accurate model representation. Multivariate splines are model representations that are constructed by a set of piecewise continuous functions, locally approximating the dynamics. Data points are collected to optimise simplex coordinates, which in turn provide a baseline for the piecewise functions. On the other hand ANN's provide a non-linear model approximation by virtue of their activation functions defining their nodes [32].

II.E. Safe Curriculum Learning for Systems With Unknown Dynamics

The basis of the Safe Curriculum Learning paradigm was first introduced in [7], where it was used to regulate a 3-mass-spring-damper system as well as a quadcopter on its six degrees of freedom. Combining the knowledge found in research discussed in previous sections, the paradigm aims at finding a control policy of complex systems whilst considering safety of the learning process. The research provided successful results for a regulation task by using the SHERPA algorithm as a safety filter.

Furthering the aforementioned paradigm, by providing independence of system dynamics knowledge is given in Fig. 2. Here, a system identification module is added to the process to provide the safety filter with an estimate of the dynamics during the online learning process.

In Fig. 2, a schematic formulation of a Safe Learning process for systems with unknown dynamics is given. Each curricular step uses the logic demonstrated by this diagram to find an optimal safe policy for the topical task at that stage of the curriculum and its respective MDP. Mapping the agent's representation of the value function and policy is then used to transfer knowledge between curricular steps. A curriculum can be constructed by repeating the schematic provided in Fig. 2 in a sequential or parallel manner, depending on the curricular requirements of the task at hand.

III. Methodology

The research in this article proposes an online safe curriculum learning paradigm with the purpose of controlling systems with unknown dynamics to track a reference signal. The black box approach assumes knowledge about the order of the state and action spaces to be available. However, system dynamics and input dynamics are not provided in an exogenous manner throughout the learning process. The decisions made during the research regarding the specific methods used to construct the multiple building blocks of the safe curriculum

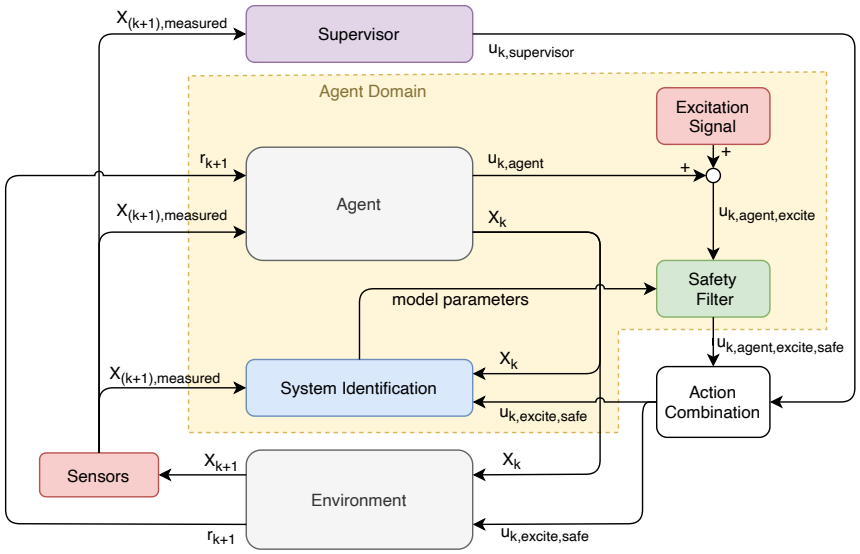


Figure 2: Schematic overview of a single curricular step

learning paradigm are presented in this section. The section is started with the task to be performed by the agent along the RL method of choice, after which the curricular construct is elaborated, followed by the safety filter.

III.A. Learning Framework

Provided that a reference tracking controller is sought, as outlined in the introduction, a tracking task was devised to obtain a high-performance stabilising policy for a system. As mentioned in Section II.A, reinforcement learning processes are guided by their underlying MDP's reward function, $\mathcal{R}$, for optimisation of the policy and value function. In this research the Linear Quadratic (LQ) cost function (shown in Eq. (4)) is selected as it can be used in optimal control strategies in concordance with RL methods such as Q-Learning [33, 34]. The Q and R matrices found in the LQ cost function are diagonal weighting matrices. Their respective diagonals contain a weight expressing the importance of the state and input vector elements' contribution to the cost [34].

$$\mathcal{R} : c = x^T Q x + u^T R u \tag{4}$$

By limiting the scope of the research to discrete-time linear systems, the Linear Quadratic Tracking (LQT) paradigm proposed by [33] was found to be promising. That research uses Q-learning with PI to learn a tracking task. Fundamentally based on the Linear Quadratic Regulation (LQR) model, the authors provide mathematical proof of a RL agent's ability to learn a tracking stabilising policy. Instead of finding a controller that regulates the states to an equilibrium, the LQT task aims to regulate the error between the system's state and the reference signal. The scope being limited to linear systems, it was chosen to use the linear discrete-time state space (A, B, C, D) representation to model the system dynamics in the environment and consequently as the transition function, $\mathcal{T}$. By means of the C matrix of this representation and the assumption that feedforward dynamics are omitted, the error is defined as shown in Eq. (5). This leads to the use of an augmented state, X_k containing both the system's state and the reference's state, x_k^r. Due to the introduction of an augmented state, X_k, the Q matrix of the LQ cost function in Eq. (4) needs to be altered to accommodate for the new dimension of this augmented vector. In [33], a substitute matrix, $Q_1 = C_1^T Q C_1$, is defined to ensure conforming dimensions with the augmented state.

$$e_k = y_k - x_k^r = Cx_k - x_k^r = \begin{bmatrix} C & -I \end{bmatrix} \begin{bmatrix} x_k \\ x_k^r \end{bmatrix} = C_1 X_k \tag{5}$$

The Q-Learning approach that was chosen is not of tabular form. Although the state and action spaces are discrete in time, they are not inherently discrete to fit within a tabular formulation. Instead, the possibilities

of state-action pairs is near-infinite. In order to accommodate such spaces, a continuous Q-function is required. For this purpose, Kiumarsi et al [33] proposes to use a kernel matrix, H, to determine the Q-value as outlined in Eq. (6). The focus of the RL process is to find the optimal kernel matrix for the topical task. In the equations below, the vector, $Z = \begin{bmatrix} X_k & u_k \end{bmatrix}^T$, contains both the augmented state as well as the action at a given time step. Additionally, the constant γ represents the discount factor.

$$q_k = \frac{1}{2} Z_k^T H Z_k \tag{6}$$

$$Z_k^T H Z_k = X_k^T Q_1 X_k + u_k^T R u_k + \gamma Z_{k+1}^T H Z_{k+1} \tag{7}$$

In more general terms, the Q-function satisfying the the Bellman equation [33, 35] for an LQT task is formulated in Eq. (7). This equation holds for discrete-time linear systems. Obtaining the values of the symmetric kernel matrix, H, is done by using sample based VI. By sampling the states and inputs along the system trajectory, an estimate of the kernel matrix can be computed using a simple Ordinary Least Squares (OLS) regression. The estimator is discussed in more detail in Section III.C. Mathematically, the estimate of the subsequent kernel matrix, H^{j+1}, is obtained by solving Eq. (8), in which Z_k is sampled over time. This covers the policy evaluation step of the VI strategy.

$$\left(Z_k \otimes Z_k\right)^T vec(H^{j+1}) = X_k^T Q_1 X_k + u_k^T R u_k + \gamma Z_{k+1}^T H^j Z_{k+1} \tag{8}$$

In order to complete the iterative loop of VI, the policy improvement step has to be performed. Here, the policy takes the form of a control gain, K_1, in control theoretical terms. Based on the findings in [33], the logic within the policy is given by Eq. (9), where H_{uu} and H_{uX} are sub-matrices of the kernel matrix H (see Eq. (10)).

$$u_k = -\left(H_{uu}^{-1} H_{uX}\right) X_k = -K_1 X_k \tag{9}$$

Successful learning convergence is achieved by adding a Persistence of Excitation (PE) signal to the input vector, in order to avoid non-invertible precision matrices in the OLS estimator. Random noise has been selected as the excitation signal of choice. In accordance with Narendra and Annaswamy [36], the PE signal shall not integrate to zero over the course of the sampling period if the PE condition is to be satisfied. White noise is not likely to fail this condition. However, such signal comes with the drawback that it is fundamentally not band limited and the signal power is spread equally over all frequencies.

III.B. Curriculum Setup

The learning task presented in the previous section, although proven to have convergent learning behaviour, suffers from the curse of dimensionality. Increasing the amount of states causes an exponential increase in the dimensions of the kernel matrix. When considering complex systems such as aircraft, states constitute not only the attitudes and rates in the aerodynamic and body frame, but also mappings to other frames of reference such as the inertial frame. Such systems generally adhere to state and action spaces with numerous dimensions. By designing a curriculum, the strain can be relieved from the learning process and positively impact the learning convergence, performance and safety on the aforementioned state and action spaces.

The curriculum learning strategy chosen for this research is a curriculum based on gradually increasing the task complexity. Two staging approaches were considered for this research, namely intra-task and inter-task [7]. For intra-task stages, the agent's representation of the state and action spaces was set to be the same between two consecutive curricular steps. On the other hand, inter-task staging assumes a different representation between the source and target tasks be it in terms of dimensions or in terms of internal dynamics representation. Besides increasing the task complexity, reward shaping was implemented on more complex systems as an additional guide for the agent. By changing the values on the diagonals of Q and R from Eq. (4), the contribution of each state and action element towards the total cost can be adjusted. Its use was primarily prominent when a state and its derivative were present in the state vector. For example, when a position state element is tasked with tracking a reference, the respective velocity component of that state is given a lower value in Q to focus the agent's attention on the position tracking before the velocity tracking.

A second point of attention when designing a curriculum is the transfer of knowledge between curricular steps. Given the learning task explained in Section III.A, the knowledge gained during a particular curricular step is stored in the kernel matrix H. Since this matrix is used to define the Q-value space, it contains the necessary information to dictate the agent's value function and policy. Here, it was opted for a semmantic mapping strategy of the kernel matrix. In essence, the environment and agent of the subsequent curricular step dictate the new dimensions of the kernel matrix for that step as they set the dimensions of the state and action spaces. The basis for the new kernel matrix is an identity matrix onto which the kernel values of the source task are mapped based on corresponding state, reference and action elements. For the sake of clarity, Eq. (10) outlines an overview of the different blocks of the kernel matrix relating to the state, reference and action elements denoted by the subscripts x, x^r and u respectively.

$$H = \begin{bmatrix} H_{xx} & H_{xx^r} & H_{xu} \\ H_{x^rx} & H_{x^rx^r} & H_{x^ru} \\ H_{ux} & H_{ux^r} & H_{uu} \end{bmatrix} = \begin{bmatrix} H_{XX} & H_{Xu} \\ H_{uX} & H_{uu} \end{bmatrix} \tag{10}$$

Mapping semantically requires an operator with knowledge of the system and input dynamics, which comes as a disadvantage of this specific strategy. However, it provides a more reliable starting point for the RL agent in the subsequent curricular step. Furthermore, the specific internal dynamics are not required, only a thorough understanding of the state and action vectors elements' position changes when transferring from source to target task.

III.C. Safety Filter for Systems With Unknown Dynamics

Completing the proposed paradigm in this research is the safety aspect. It is true that safety can be a consequence of using curriculum learning. However, no guarantees can be asserted by these means alone. Consequently, an additional safety module was used to ensure a safe learning process, as is shown in Fig. 2. The safety mechanism of choice is a safety filter fundamentally relying on ergodicity named SHERPA [27–29].

Essential to the SHERPA safety filter are the concepts of the SSS and FSS discussed in Section II.C. In essence, the SSS is an evolving part of the state space that is enlarged through experience. States that have been visited by the agent and are considered safe are added to the space. To push the safety aspect further, a sensor reach is added to the state to act as the risk function required for the SHERPA algorithm as it is defined by Mannucci et al [29]. For example, a radar sensor can be equipped to a quadcopter, allowing it to sense for obstacles up to a certain distance thus giving it a better idea of potential threats in its surroundings. The binary risk function sets risk at 0% when the sum of the state and the sensor reach are outside of the FSS (and conversely sets it to 100% otherwise). Besides the expansion of the SSS over time, the safety of the system is retained by finding backup policies. These policies aim at verifying that the agent is not in Lead to Fatal States (LFS), which are states that are inherently safe, but over time have a 100% probability to transition the agent into the FSS [29]. Additionally, backup policies are validated by satisfying the ergodicity condition [29, 31].

The search for such policies lies at the core of SHERPA's paradigm. Using a form of model representation, the algorithm projects a number of trajectories for a pre-defined number of time steps based on a set of policies. The trajectory generation proposed by Mannucci et al [29] relies on a series of random actions. Instead, this research considers the SHERPA safety filter to be part of the agent domain (see Fig. 2). Consequently, SHERPA has access to the agent's current policy, from which it generates a set of policies used to define the series of inputs taken for the projected trajectories. More specifically, the policy generation is based on the ϵ-greedy [6] approach used to promote exploration in RL. In most cases the agent's current policy parameters are altered by a certain random factor. In the remaining cases, a completely random policy is generated. The projection of these trajectories is performed using an uncertain model of the system and input dynamics that is identified online.

The proposed contribution of this research finds itself in the addition of a system identification module to estimate a model of the system and input dynamics. As model representation, the linear state space formulation was chosen by virtue of its efficiency and fitness for the scope of this research. The model is estimated using sample based OLS, with an estimate update frequency that is different and higher than the VI iteration process. Samples of the state and input at time step k as well as the subsequent state are

collected to construct the regression matrix, P, and regression vector, b, which are then used to find the model estimate as shown in Eq. (11).

$$\hat{\theta} = \left(P^T P\right)^{-1} P^T b \tag{11}$$

The model uncertainty of the estimate is derived from the residuals between the estimated model and the sampled data. Thereby resulting in the absolute parametric uncertainty of the system dynamics estimate, $\hat{A}$, the input dynamics estimate, $\hat{B}$, and the reference dynamics estimate, $\hat{F}$. These uncertainties are crucial for the uncertainty region's propagation in SHERPA's backup policy projection process.

IV. Simulation Results

The safe curriculum learning for unknown systems architecture proposed by this research was first validated on a relatively more tractable cascaded Mass-Spring-Damper (MSD) system bounded at one end. This is followed by an example of the application on primary flight control on a linearised quadrotor model, which has been limited to attitude and altitude control.[a]

IV.A. Model Validation on Mass-Spring-Damper System

The architecture presented in Fig. 2, was validated on a N-MSD system, where N denotes the number of masses present in the system. This system is used because of its modularity and scalability. Additionally, making a N-MSD system unstable can be achieved by making either the spring constant, k, or the damping constant, c, negative. The curricular construction is centred around inter-task staging steps where the agent's representation of the system dynamics varies throughout the curriculum. Initially, a single mass attached to fixed bound (1-MSD) is used as an environment for the agent, with stable open-loop characteristics. The agent's task is to have the mass position track a sinusoidal reference signal. Throughout the curriculum, more masses are added between the wall and the previously existing system. Moreover, the system is gradually made unstable to conform with the statistical conditions of entropy and diversity that are needed for a curriculum. This can be achieved by making either the spring constant, k, or the damping constant, c, negative. The ultimate goal is to have the agent track a sinusoidal reference signal on a $3 - MSD$ system with the last mass's position, x_3, whilst the other masses in the system are regulated. All the while keeping the former within safety bounds. Finally, the actions taken by the agent are bounded by saturation limits set to 5N in either application direction.

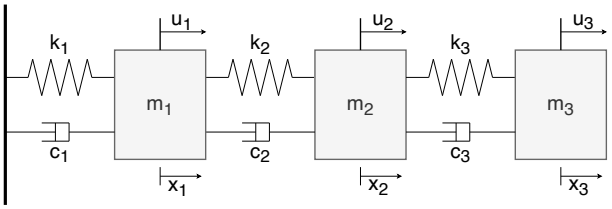


Figure 3: Figure showing a cascaded 3-Mass-Spring-Damper system

Table 1: Characteristic parameters of the unstable 3-MSD system shown in Fig. 3

Mass	m $[kg]$	k $[N/m]$	c $[N/(ms^{-1})]$
1	0.3	-1	6
2	0.8	3	-1
3	0.4	4	3

Table 2: Characteristic parameters of the persistence of excitation signal and measurement noise

PE signal		Measurement Noise	
Parameter	Value	Parameter	Value
μ_{PE}	0.0	μ_{MN}	0.0
σ_{PE}	1.5	σ_{MN}	$2.0 \cdot 10^{-3}$
f-band	$[-\infty, \infty]$		

[a]Full code of the implementation used in theses simulations can be found at `https://github.com/DBdiego/SafeCurriculumLearning.git`

The focus of this validation is not only to prove that the agent can safely learn but also to show that the learning efficiency is improved when complementing the agent with a safety filter and pacing it through a curriculum. Therefore, the results of the proposed architecture are compared to flat learning of the aforementioned tracking task on the full 3-MSD system. With the characteristic parameters of the 3-MSD system and excitation signal presented in Table 1 and Table 2, respectively, the agent converged to a stabilising policy without leading the system in the FSS in a curriculum learning framework. Consequently, completing the input's excitation signal, a measurement noise, with characteristic hyperparameters outlined in Table 2, was added to the state time series to provoke unstable kernel and policy updates by the agent. Indeed, without the addition of measurement noise, the learning process stays safe even without a safety filter. By adding measurement noise, the benefits of SHERPA can be showcased more clearly. Consequently, to make the results comparable, both the flat learning benchmark and the safe curriculum implementation were complemented by measurement noise.

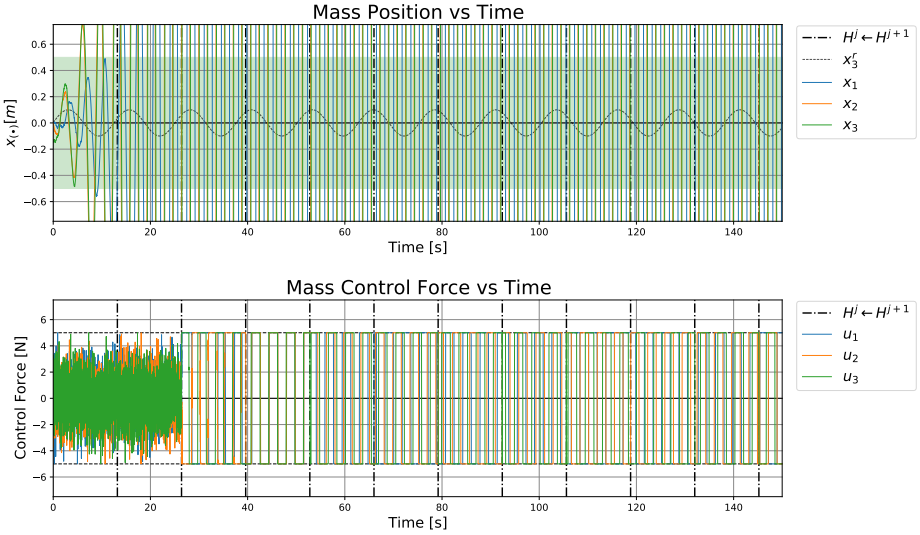


Figure 4: Evolution of mass positions and control forces over time; Online VI; Flat Learning; No SHERPA; Gaussian sensor noise, $\sigma = 0.0012$; $x_{init} \in [-0.2, 0.2]$; $\dot{x}_{init} \in [-0.25, 0.25]$; Green is the SSS for x_3

When neither the curriculum and the safety filter are activated, the agent's learning behaviour is volatile and divergent. This is reflected in Fig. 4. It takes two kernel updates ($H^j \leftarrow H^{j+1}$) for the agent's policy to dictate actions that are continuously saturated. Furthermore, once the position of the three are well beyond the SSS (green area in Fig. 4), the tracking task becomes more arduous. Conversely, when using the architecture proposed in this research, the resulting learning process is stable and convergent. The convergence rate however varies with the respective amplitudes of the PE and measurement noise signals. In Fig. 5, the position state and action force propagation of all masses is given, showing an adequate tracking of the reference by x_3. The total learning time for the safe curriculum learning implementation was set to equal the flat learning benchmark's total learning time. Each curricular step provides 50 seconds to the agent to learn a stabilising tracking policy.

Furthermore, the system identification module (blue box in Fig. 2) gradually lowers the parametric absolute uncertainty during the simulation. By assuming that the system to be modelled is linear, the estimated model parameters are time-invariant. As such, the sliding window principle can be used for system identification. The initial estimate is the only one that is delayed as it requires sufficient samples to be collected. During this sampling, SHERPA does not need to be activated since no uncertain model estimate is available for the backup policy projection. The purpose is to reduce the system identification module's initial sampling time as much as possible whilst keeping the estimate's uncertainty relatively low. The evolution of these absolute parametric uncertainties are shown in Fig. 7. It can be observed that the estimate becomes available at $t = 5.8$ seconds. Finally, SHERPA's interventions are shown in Fig. 6. Here, the grey area provides a better understanding of this architecture's major drawback, namely, that the system is prone to go into the FSS during this time interval.
This concludes the validation of the proposed architecture on a simple system such as a Mass-Spring-Damper

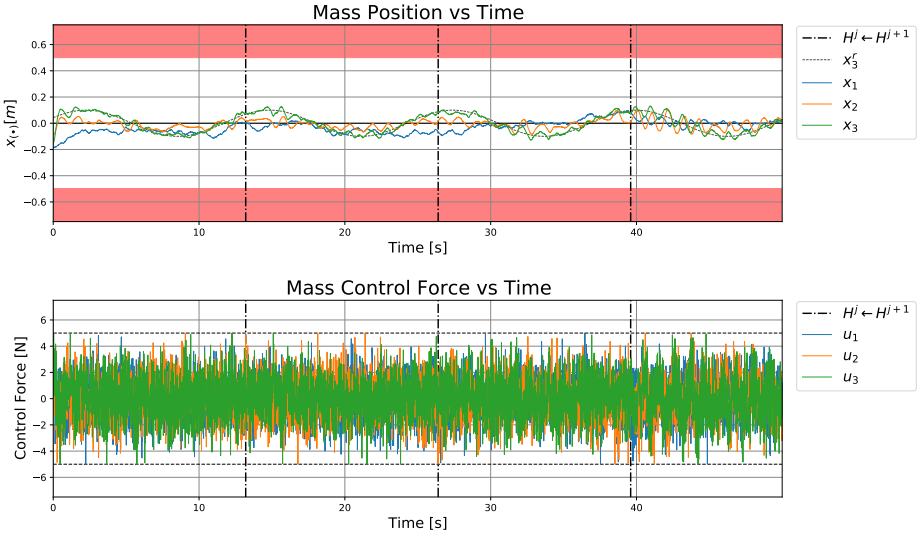


Figure 5: Evolution of mass positions and control forces over time; Online VI; Safe Curriculum Learning (step 3); SHERPA activated; Gaussian sensor noise, $\sigma = 0.0012$; $x_{init} \in [-0.2, 0.2]$; $\dot{x}_{init} \in [-0.25, 0.25]$; Red is the FSS for x_3

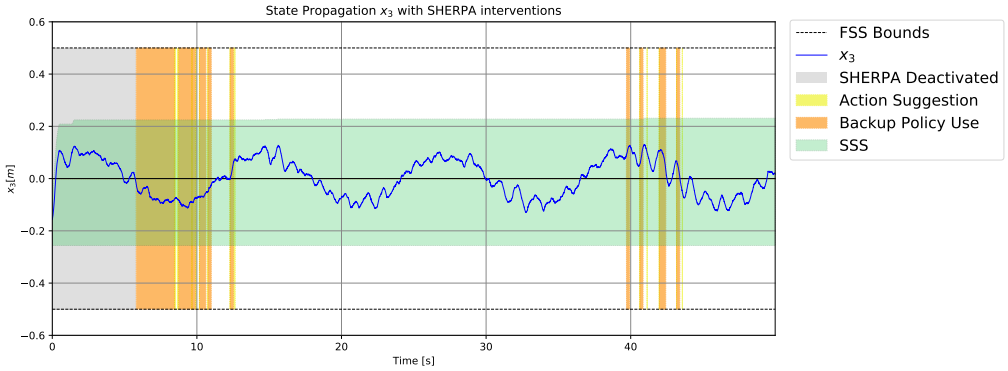


Figure 6: Evolution of the SSS over time along with SHERPA interventions

system. Due to the inter-task staging strategy being easy to understand by an operator, no supervisor was required to ensure convergent learning behaviour.

IV.B. Quadrotor Primary Flight Control

Having verified the proposed paradigm, it is possible to apply it to more complex systems such as a quadrotor for the proof-of-concept. More specifically, the goal is to find a stabilising policy that can track references in all four Degrees Of Freedom (DOF) available in this model. These are constituted of the pitch, roll and yaw attitudes, and the altitude.

A linearised continuous-time quadrotor model was derived based on the state and action vector formulations of Eq. (12) and is shown in Eq. (13). Due to the paradigm's limitation to discrete-time dynamics, the state space system shown in Eq. (13) was discretised using the zero-order hold method. By assuming small angles for ϕ, θ and ψ, the rates p, q and r can be assumed to be the same as $\dot{\phi}$, $\dot{\theta}$ and $\dot{\psi}$. Furthermore, the equilibrium point used for linearisation is hovering flight.

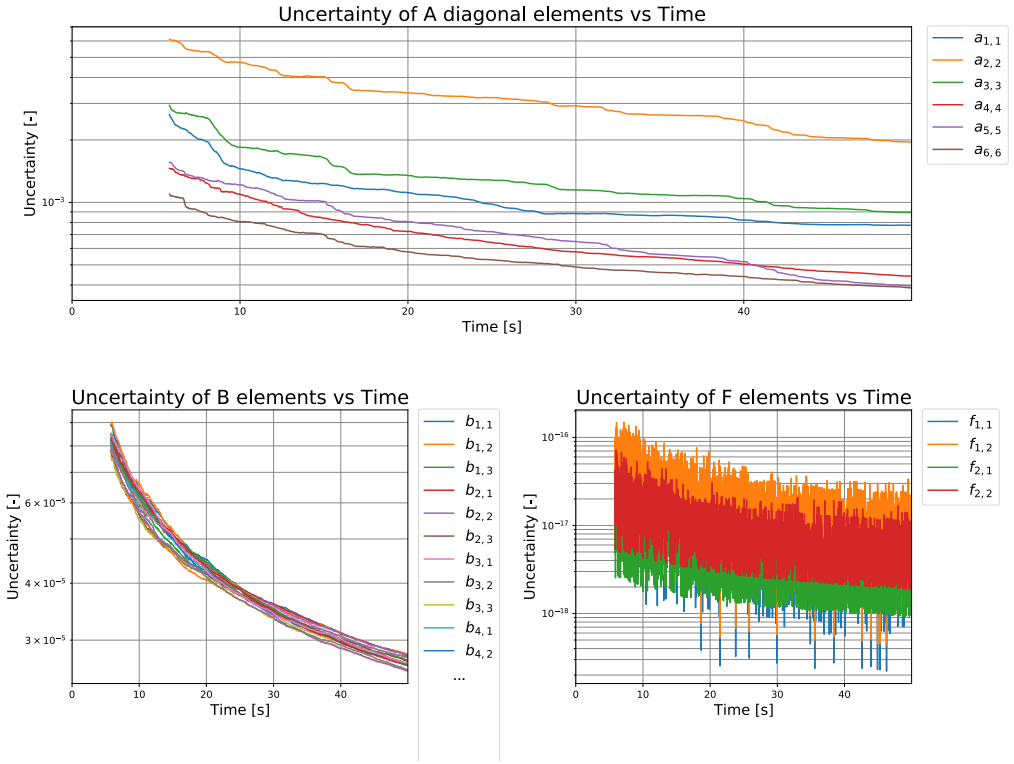


Figure 7: Evolution of the absolute parametric uncertainty of all matrices forming the uncertain model estimate used in SHERPA, $\hat{A}$, $\hat{B}$ and $\hat{F}$; Online VI; Safe Curriculum Learning (step 3); SHERPA activated; Gaussian sensor noise, $\sigma = 0.0012$.

$$
\begin{aligned}
x &= \begin{bmatrix} \phi & \theta & \psi & p & q & r & w & z \end{bmatrix}^T \\
u &= \begin{bmatrix} f_t & \tau_x & \tau_y & \tau_z \end{bmatrix}^T
\end{aligned}
\tag{12}
$$

$$
A = \begin{bmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
\quad
B = \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & \frac{1}{I_{xx}} & 0 & 0 \\
0 & 0 & \frac{1}{I_{yy}} & 0 \\
0 & 0 & 0 & \frac{1}{I_{zz}} \\
-\frac{1}{m} & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\tag{13}
$$

In this implementation, a combination of intra-task and inter-task staging strategies was used, where the agent keeps the system dynamics representation constant throughout the curriculum. However, the representation of the input and reference dynamics varies. This was chosen based on the task complexity being relatively high in the curricular steps presented in Table 3. The agent gains control of an additional input at every step. The other inputs are controlled by an LQR supervisor supervisor. The gains of the supervisor were deduced using an LQR theory with the Q and R matrices given in Eq. (16). Along with a new input dimension, comes one or multiple new degrees of freedom to be controlled by the agent. Consequently, a reference signal is added for each step as well. The curricular setup has been summarised in Table 3 to

provide a better overview. The order in which the inputs are switched from the supervisor to the agent has no specific importance other than the hyperparameters shown in Table 5 should be altered. Indeed, the attitude angles that are not learned have to remain small in order to satisfy the small angle assumption made when linearsing the model.

In essence, each input has respective states that it controls. Not only does this provide better tractability during the learning phases, but it also allows for an easier separation between the LQR supervisor and the agent's control actions. Indeed, this approach avoids having the supervisor controlled actions' and states' contribution to the cost function (Eq. (4)) during the learning process, giving the agent a better understanding of the influence of its actions on the system.

The fifth curricular step focuses on mapping the tracking policy found in the first four steps, into a policy that uses the rotor RPMs as input vector instead of torques and forces along the respective degrees of freedom. In this last step, no learning occurs; only state propagation checks to ensure the learned policy can track along all degrees of freedom with minimal error ($RMSE < 0.05$).

The policy learned by the agent in the first four curricular steps applies to any quadcopter configuration as long as there is a mapping, ξ, to a rotor RPM input vector, Ω. For sake of simplicity the experiment conducted in this research has assumed a linear relationship between a rotor RPM and the thrust force, $f_{t,i} = b \cdot \Omega_i$, and torque, $\tau_{z,i} = d \cdot \Omega_i$, it generates. This assumption facilitates the mapping process and proof of concept. Notably, however, non-linear effects such as blade flapping and aerodynamic disturbances are not considered in this experiment. With this assumption, the mapping matrix, ξ, for an X-configuration quadrotor can be defined as outlined in Eq. (15). This map is derived from a basic understanding of forces on a quadrotor shown in the form of three free body diagrams in Fig. 8.

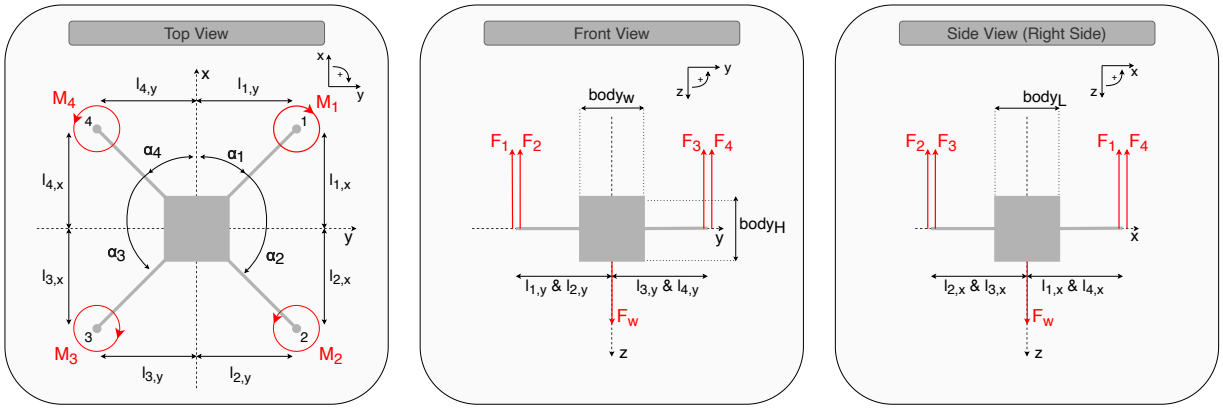


Figure 8: Views of the quadcopter forces and other important dimension.

$$\pi_\Omega = \xi \cdot \pi_{f,\tau} \qquad (14)$$

$$\xi = \begin{bmatrix} b & b & b & b \\ -bl_{1,y} & -bl_{2,y} & bl_{3,y} & bl_{4,y} \\ bl_{1,x} & -bl_{2,x} & -bl_{3,x} & bl_{4,x} \\ d & -d & d & -d \end{bmatrix} \qquad (15)$$

Table 3: Curriculum setup. Red are the states and inputs controlled by an external *supervisor* LQR-controller. Green are the states and inputs controlled by the agent. Underlined states track a reference.

Curricular Steps		1	2	3	4	5
Inputs	Thrust force	f_t	f_t	f_t	f_t	Ω_1
	Torque around x	τ_x	τ_x	τ_x	τ_x	Ω_2
	Torque around y	τ_y	τ_y	τ_y	τ_y	Ω_3
	Torque around z	τ_z	τ_z	τ_z	τ_z	Ω_4
States	roll	ϕ	$\underline{\phi}$	$\underline{\phi}$	$\underline{\phi}$	$\underline{\phi}$
	pitch	θ	θ	$\underline{\theta}$	$\underline{\theta}$	$\underline{\theta}$
	yaw	$\underline{\psi}$	$\underline{\psi}$	$\underline{\psi}$	$\underline{\psi}$	$\underline{\psi}$
	roll rate	p	$\underline{p}$	p	p	p
	pitch rate	q	q	$\underline{q}$	q	q
	yaw rate	$\underline{r}$	r	r	r	r
	vertical velocity	w	w	w	$\underline{w}$	w
	z-position	z	z	z	$\underline{z}$	$\underline{z}$

Table 4: Characteristic parameters for the quadrotor system

Parameter	Value	Unit
I_{xx}	0.045667	$[m^4]$
I_{yy}	0.045667	$[m^4]$
I_{zz}	0.090667	$[m^4]$
h_{body}	0.1	$[m]$
w_{body}	0.1	$[m]$
l_{body}	0.1	$[m]$
l_{arm}	0.3	$[m]$
m_{body}	0.2	$[kg]$
m_{arm}	0.1	$[kg]$
m_{motor}	0.2	$[kg]$
m_{quad}	1.60	$[kg]$
α_{arms}	45	$[deg]$
b	2.04366e-3	$[N/RPM]$
d	2.78893e-4	$[Nm/RPM]$

For a quadrotor, it was found empirically that the amplitude and the frequency of the PE signal both have an essential impact on the learning behaviour; result which was not the case for the N-MSD system used to validate the architecture. The agent was more specifically found to experience more difficulties learning in a convergent manner when the PE signal has a low frequency bandwidth. Therefore, a band-limited excitation signal was used in this experiment. An arbitrary frequency range was defined to be between 50Hz and 100Hz. By increasing the frequency, it was found that the lower derivatives of each DOF (i.e. attitude angles) are more centred around their respective regulated values, specifically zero in this experiment.

Additionally, the frequency and amplitude of the reference signal play an essential role in the convergence and stability of the learning process. They fundamentally define the task that the agent has to learn. Moreover, when their frequency range is similar to the eigenfrequencies of the system, an additional complexity is added to the learning process as the system starts to resonate. The hyperparameters for both signals throughout the curriculum are summarised in Table 5. Notably in this table is the amplitude for the reference signals of the attitude angles diminishing as the curriculum evolves. Only the attitude being learned is allowed a large amplitude to facilitate the differentiation between the states the agent is learning to control from the others [b]. Due to the linearisation of the system, small angle approximation and the assumption that $\{p, q, r\} = \{\dot{\phi}, \dot{\theta}, \dot{\psi}\}$, the angles that are not learned have to remain small in order for the modelled dynamics to be representative of a quadrotor.

The improvements provided by both a curriculum on the learning stability and a safety filter on the learning process's safety are shown by comparing their simulation results to a benchmark flat learning case. The cumulative learning time within the simulation is limited to 170 seconds. Outlined in Fig. 9 are is the evolution of all the attitude state angles and the altitude position for flat learning. It is apparent that the task is too complex for the agent to learn a stabilising tracking policy in the given time frame. Furthermore, the safety-critical roll and pitch attitudes are constantly in the FSS. It was concluded, that a curriculum on such task is necessary to more gradually increase the task complexity and use a safety filter to ensure a safe learning process.

Following the curricular strategy presented in Table 3, the first sub-task was learning to track a reference on the yaw angle, ψ, whilst all the other DOF are under the authority of a regulating supervisor. To guide the agent in the learning process, an additional reference is provided for the yaw rate, r. The learning behaviour is shown in Fig. 10a. In a second instance, control of the roll attitude comes under the agent's authority, and the agent learns a policy that tracks both roll and yaw angle. This is presented in Fig. 10b. This is then followed by the addition of the pitch angle, which is shown in Fig. 11a. Finally, the altitude DOF is controlled by the agent in the fourth curricular step giving the agent full authority on all the system's

[b] Note that the agent has full observation of the system when learning specific states in each curricular step.

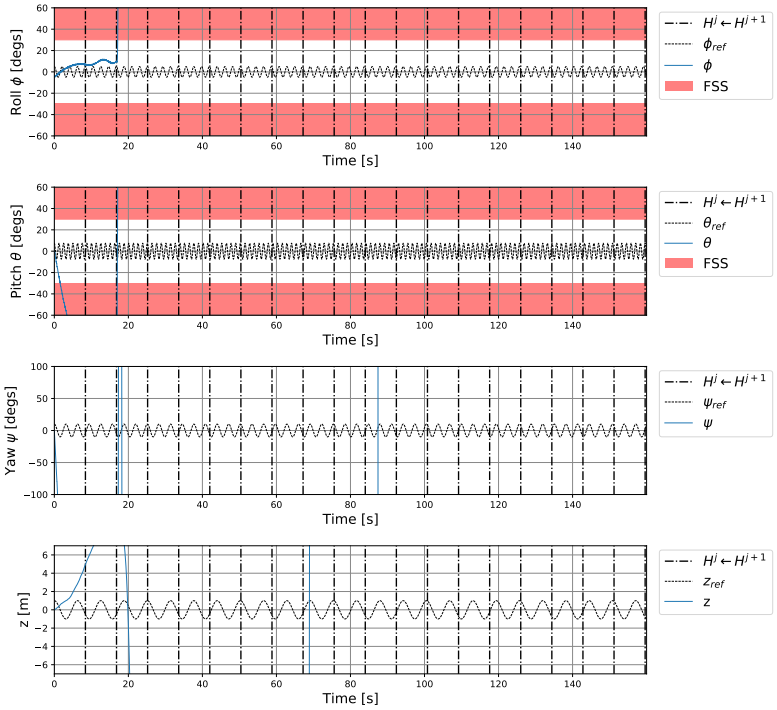


Figure 9: Evolution of attitude and altitude over time; Flat Learning; Online VI; No Safety Filter (SHERPA)

degrees of freedom. The final tracking time series is shown in Fig. 11b.

In Figs. 12 and 13, the evolution of the policy through the curriculum can be seen. The rows annotated with a * contain the LQR supervisor gains, whereas the others contain ones that have been learned during the curricular step by the agent. The policies reflect the curricular setup presented in Table 3, where the states and their respective references that are learned and tracked during a curricular step are highlighted. This means that they receive a gain that is non-zero (zero gains are white).

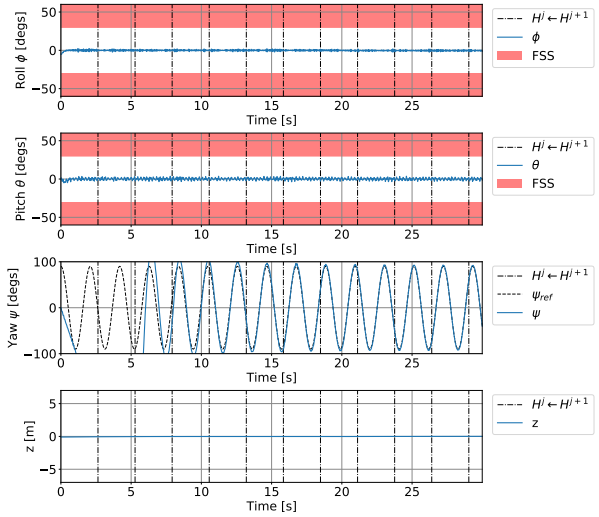
In the experiment proposed by this research only two states define the safety of the system, namely roll and pitch, for which the FSS limits have been set at $\pm 30°$. From figures 10b and 11a, it can be observed that the trajectory in the respective states adventures into the FSS, which is considered to be equivalent to a crash or resulting in an undesired situation. The final module required to complete the paradigm proposed in Fig. 2, is the safety filter SHERPA.

The limitations on roll and pitch angles serve as the principal safety constraints on the system, whereas the position of the quadrotor has been left unbounded. As such, the safety filter is only applied to curricular steps 2 and 3. As seen in figures 10b and 11a, the system goes into the FSS for both the roll and pitch states when the agent learns to control these dimensions. As mentioned in Section II.C, a backup policy has to satisfy two conditions. One is the ergodicity condition where the system has to come back to a previously visited state (or close to the said state). The other is a safe trajectory to reach this previously visited state. As the model used in this experiment consists of decoupled dynamics, the ergodicity can be limited to the set of states directly related to the input learned by the agent at a given curricular step.

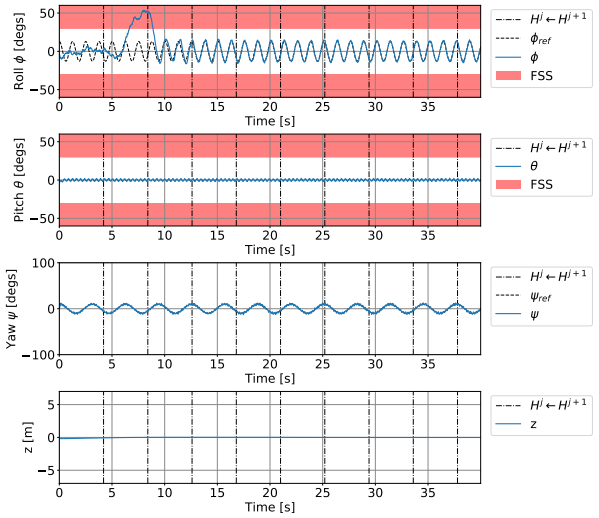
In the second curricular step, the ergodicity and safe trajectory conditions apply solely to state elements related to the input torque around the x-axis, τ_x. More specifically, only the roll angle, ϕ, is checked for inclusion in the SSS. For the ergodicity condition, both roll angle and roll rate, $\{\phi, \dot{\phi}\}$, as well as their respective reference signals, $\{\phi^r, \dot{\phi}^r\}$, are used. Due to the use of augmented states (Section III.A), the reference plays an important role in state propagation and, consequently, the system's safety. The closeness interval is set to $\pm 5°$ and $\pm 7.5°$ for roll angle and roll rate states, respectively. Furthermore, the reference states have a closeness interval of $\pm 7.5°$. As an initial backup policy, SHERPA uses the first policy given to the agent after the knowledge transfer between curricular steps.

In the subsequent curricular step, SHERPA ensures the safety of both the roll angle and the pitch angle.

The ergodicity condition is applied to all states related to the input torques in both directions, τ_x and τ_y. As such, it drastically complicates the process of finding a backup policy. Due to the addition of six dimensions, the space in which to look for these policies becomes relatively large, making the search more computationally expensive [c].

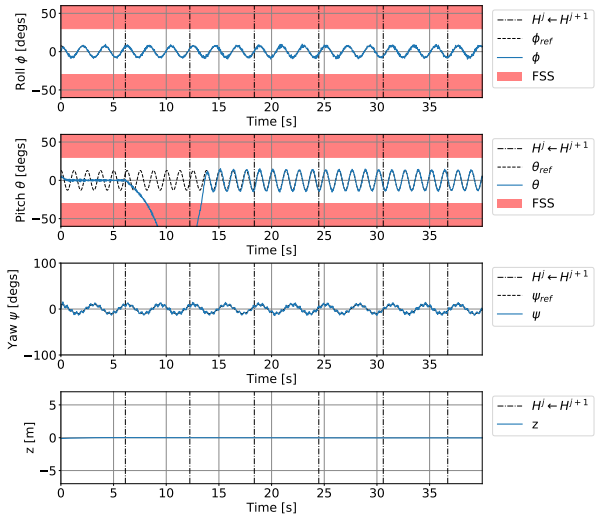


(a) Attitude evolution during curricular step 1; Agent authority = $\{\psi\}$; Learning yaw angle, ψ, tracking.

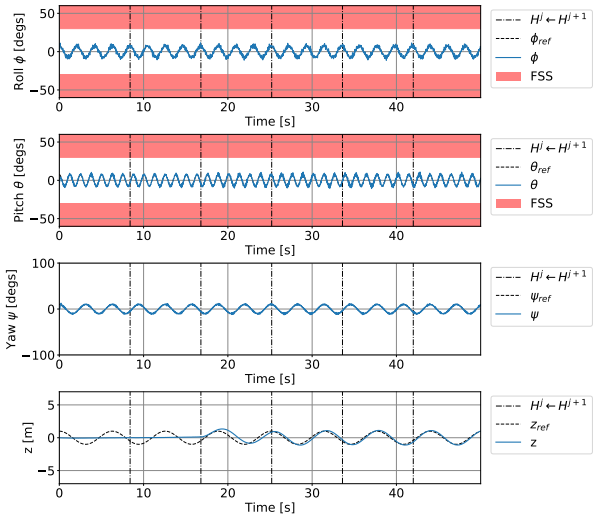


(b) Attitude evolution during curricular step 2; Agent authority = $\{\phi, \psi\}$; Learning roll angle, ϕ, tracking.

Figure 10: Evolution of attitude angles over time for curricular steps 1 and 2



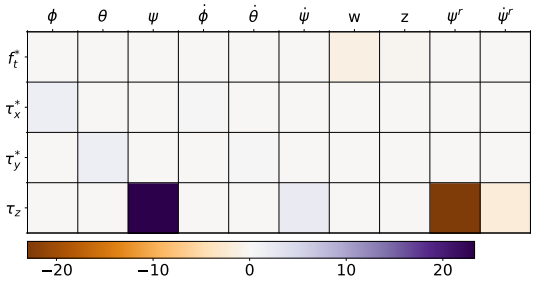
(a) Attitude evolution during curricular step 3; Agent authority = $\{\phi, \theta, \psi\}$; Learning pitch angle, θ, tracking.



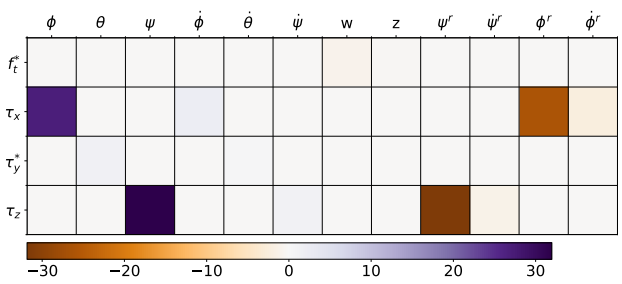
(b) Attitude evolution during curricular step 4; Agent authority = $\{\phi, \theta, \psi, z\}$; Learning altitude position, z, tracking.

Figure 11: Evolution of attitude angles over time for curricular steps 3 and 4

[c]For the 40 second simulation time shown in Fig. 14b the run time was 3h+ on a MacBook Pro Intel i7 2.3 GHz (late 2013)

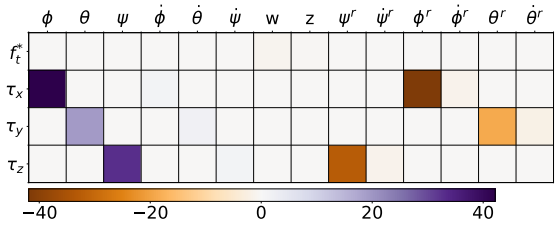


(a) The resultant policy gains after the first curricular step; Online VI; No Safety filter present; * denotes the policy rows that are controlled by the LQR supervisor.

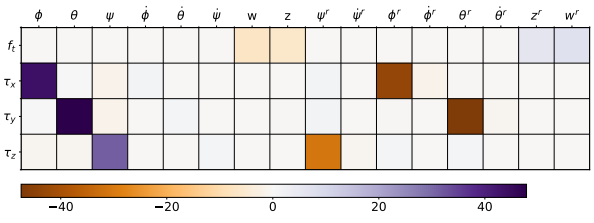


(b) The resultant policy gains after the second curricular step; Online VI; No Safety filter present; * denotes the policy rows that are controlled by the LQR supervisor.

Figure 12: Resultant policy parameters after the first and second curricular step, left and right, respectively.

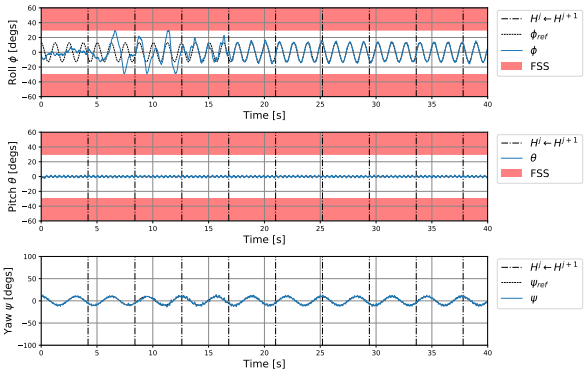


(a) The resultant policy gains after the third curricular step; Online VI; No Safety filter present; * denotes the policy rows that are controlled by the LQR supervisor.

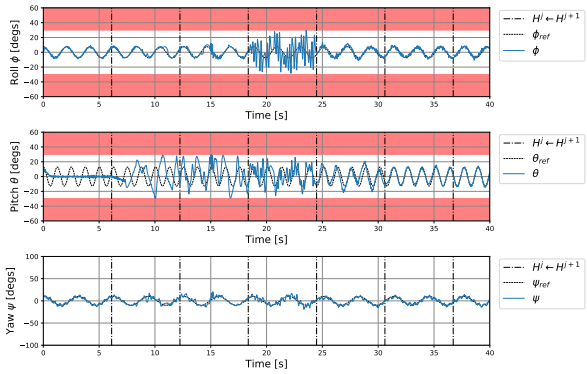


(b) The resultant policy gains after the fourth curricular step; Online VI; No Safety filter present; *denotes the policy rows that are controlled by the LQR supervisor.

Figure 13: Resultant policy parameters after the third and fourth curricular step, left and right, respectively.



(a) Evolution of attitude angle states during curricular step 2; Safe Curriculum Learning; Online VI; SHERPA active on ϕ.



(b) Evolution of attitude angle states during curricular step 3; Safe Curriculum Learning; Online VI; Red is the FSS for $\{\phi, \theta\}$; SHERPA active on $\{\phi, \theta\}$.

Figure 14: Evolution of attitude and altitude over time for Safe Curriculum Learning steps 2 and 3.

The safe curriculum learning process is shown in Figs. 14a and 14b for curricular step 2 and 3 respectively. By comparing the propagation of these states with the ones in Figs.10b and 11a, it can be seen that safety is indeed preserved at all times during the learning process of each respective curricular step. It should be noted, however, that the presence of a safety filter impedes the learning efficiency. This effect can be

attributed to the addition of PE before the action is passed into the safety filter (as shown in Fig. 2). Without an appropriate PE signal, the agent cannot learn correctly, and policy updates are more likely to be unstable.

By mapping the policy using the mapping matrix ξ given in Eq. (15), the tracking task could be defined in terms of rotor RPMs. Ultimately, the goal is to obtain a policy for optimal tracking control of a given quadcopter configuration. As such, no further learning is performed during the last curricular step. Instead, the mapped policy is simulated within the environment.

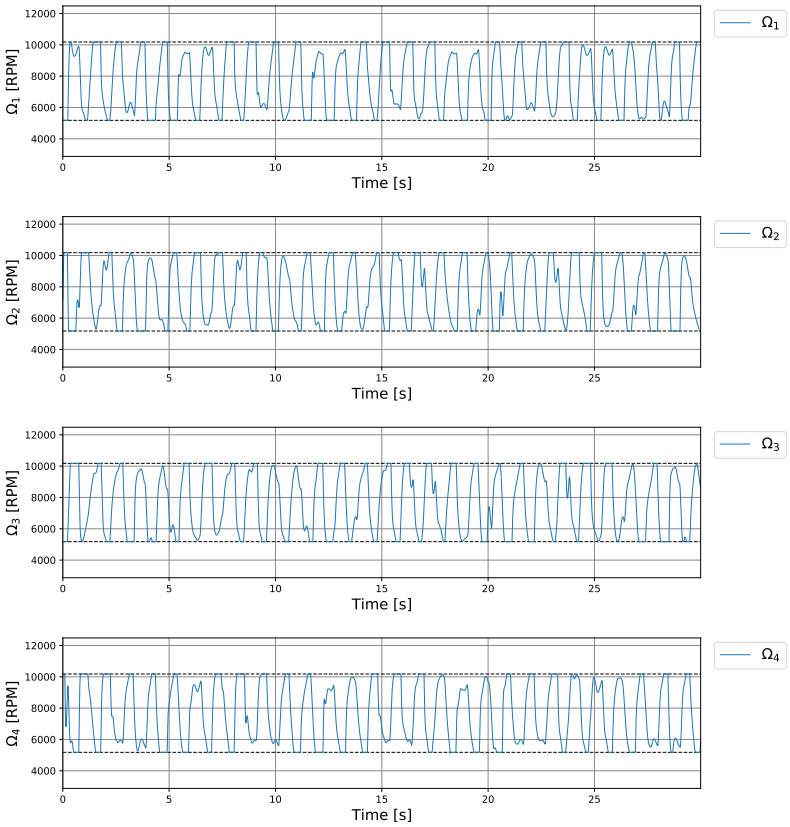


Figure 15: Evolution of rotor RPMs after mapping the policy (step 5); No learning.

The amplitudes and frequencies of the reference signal were changed in the last step to confirm that no overfitting occurs during the curriculum. Furthermore, it allows for showcasing the range in which the obtained policy can be applied. In Fig. 15, the rotational speed of each motor blade is outlined. It can be seen that the rotors move in pairs depending on the reference of each state. Moreover, the tracking error is relatively small, even at extreme reference frequencies and amplitudes, which is determined based on the action saturation imposed on the rotor RPMs.

This section has shown the application case for the proposed paradigm in Fig. 2 on a quadrotor. Due to its larger state and action spaces, this system complicates the agent's task of learning a stabilising and tracking policy. Therefore, a more elaborate curricular strategy was devised to provide the agent with a more gradual learning complexity. Furthermore, more advanced policy search strategies were put in place within SHERPA to accommodate larger and sparse (safe backup) policy spaces.

IV.C. Discussion

The safe curriculum learning architecture presented in Fig. 2 has proven to be effective in learning an optimal tracking control policy on linear systems with unknown dynamics. However, when analysing the results more closely, it is found that the complexity of the system on which the paradigm is applied has an essential impact on the learning stability and the learning convergence. When the comparison is made between a more

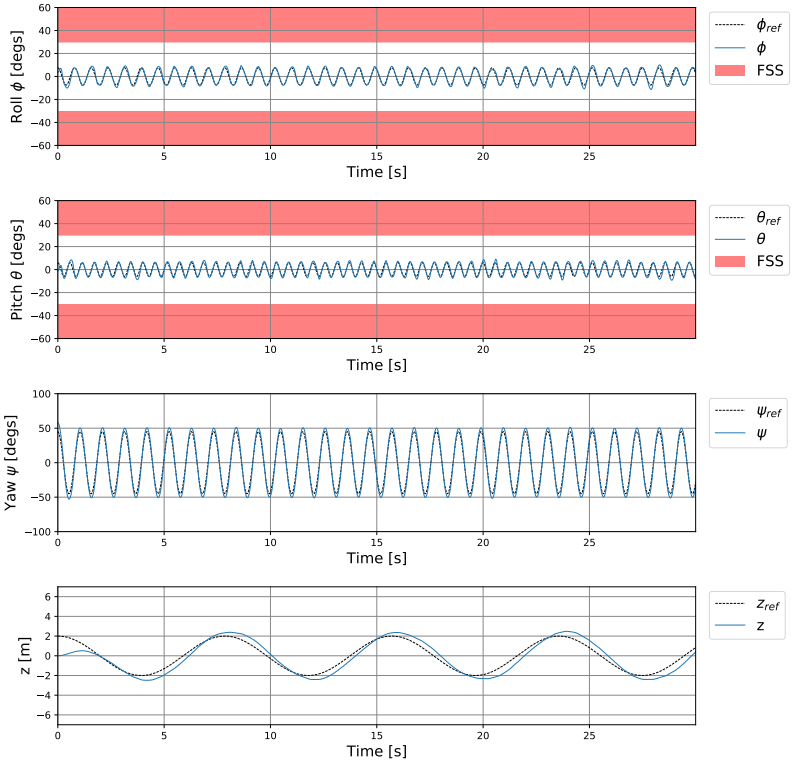


Figure 16: Evolution of attitude and altitude states using π_Ω (step 5); No learning.

simplistic N-MSD system and a linearised quadrotor model, it is noticeable that the curricular strategy and the safe learning one are more elaborated on the more complex quadrotor system. The additional complexity is also reflected in the wall clock times of each learning process, where the quadrotor simulation requires additional computational power.
Additionally, the complexity of the system on which the paradigm is applied defines the hyperparameter sensitivity similar to the effect of the PE signal on the quadrotor. In contrast with the N-MSD system, the PE signal had to be band-limited for the UAV to achieve conceivably positive results. Moreover, a more advanced backup policy search strategy had to be implemented within SHERPA for the quadrotor to deal with the more sparse safe policy space. In this strategy, a log is kept of previously used backup policies tested during the backup search at each time step in the simulation.

An inherent flaw was discovered during the experiments regarding the paradigm presented in Fig. 2, namely the position of the PE module block in this figure. An adequate excitation signal is required for the agent to learn stably. However, the safety filter receives the action combined with the PE signal. Should this combination lead to the FSS, the safety filter overwrites the action and provides a new one. The latter lacks an excitation signal. The process is worsened when a backup policy is followed for a relatively large number of time steps, or if multiple backup policies are followed back-to-back. During these interventions, the agent is deprived of the PE signal essential for its learning stability. Besides the agent, the system identification module in the experiments shown in this paper relies on the PE signal for its model estimate. When that signal is removed, the system identification model estimate is more likely to have increased parametric uncertainty. Higher uncertainty in the model has a consequence in the internal projection algorithm of SHERPA, which it uses to find backup policies. It becomes harder to find backup policies, thus the safety filter tends to follow more backup policies in its repertoire. It fits the description of an unstable loop affecting both the learning stability and the system's safety, with a snowball effect. It is, therefore, important to accurately tune the hyperparameters for more complex systems since the paradigm is particularly sensitive to the persistence of excitation and reference signals for the learning efficiency. Additionally, depending on the dynamic stability of the system, the severity of SHERPA's hyperparameters regarding the ergodicity condition has to be adapted to promote the quality of the backup policies. Indeed, with backup policies of

high quality, safety can be ensured using SHERPA [29, 31].

Finally, the design of the curriculum and the respective staging strategy still relies on the operator's knowledge of the system's approximate dynamics. In order to successfully transfer knowledge between curricular steps, a sensible mapping is required. However, the agent inherently is not aware of the dynamics when simulated in the environment.

V. Conclusion

The insurgent needs for system controllers that can optimally track a reference signal have become apparent in recent years. Not only does the range of applications for such controllers expand over time, but the complexity of the systems in these applications is increasing as well. This paper proposes a safe curriculum learning architecture where the the fundamental principles of reinforcement learning, curriculum learning, safe learning and system identification are combined to provide a methodological approach for finding optimal stabilising tracking policies for such applications. Through two different experiments, the paradigm has been proven effective in stable learning of such policies for systems with unknown dynamics. Although computationally expensive, the results show it is possible to learn a complex tracking task safely. The complexity of the system on which the learning problem is applied was found to have an important impact on the level of strategies to be used and the sensitivity on hyperparameters. Even though the paradigm proposed in this paper shows great potential for a variety of applications, it has only been demonstrated for linear systems with decoupled dynamics thus limiting the conclusions of this research to linearised systems. However, the results in this paper can serve as a foundation for further research in this direction.

V.A. Recommendations & Future Work

The research presented in this paper is focused on the analysis and experimentation on time-invariant linear systems. Further research should detail the applicability of the architecture proposed in this paper to non-linear systems and time-variant linear systems. For such implementations, more advanced reinforcement learning approaches are required such as an actor-critic architecture [6]. A system identification module could be added to Pollack and Van Kampen [7], for example. Furthermore, the system identification module requires adaptation as well to allow accurate model representations of non-linear systems.

Additionally, another limitation that became apparent in the experiments is the dependence on the persistence of excitation signals for stable and convergent learning. In the paradigm presented in this paper the placement of a safety filter between the PE signal and the state propagation in the environment poses a risk to the inclusion of such a signal. One way this could be solved, would be to define an uncertainty that is representative of the PE signal and include it into the model estimate uncertainties coming from the system identification module. Adding PE uncertainty would allow SHERPA to perform its policy projections with the inclusion of PE and return an action with excitation.

Furthermore, the safety filter used in the experiments proposed in this paper, namely SHERPA is relatively computationally expensive. Consequently, the learning process can not happen in real-time on conventional computers, and certainly not onboard. However, the Mannucci et al [29] proposes another method, optiSHERPA, which has lower computational complexities.

Finally, it should be investigated if this paradigm shows equivalent potential when applied on systems with coupled dynamics. Coupled dynamics require a different approach in curricular and safe learning strategies when a supervisor is present. This is due to the actions computed through the policy gain being partially defined by the agent and partially by the supervisor. Consequently, the supervisor has an impact on the cost function in the agent's learning process.

Appendix

The table below contains the hyperparameters for the persistence of excitation signal and the reference signal throughout the curriculum of the quadrotor. Providing a better overview and the possibility to reproduce the experiments if desired.

Curricular Steps			1	2	3	4	5
PE	f_t [N]	Amplitude [N]	10.5	13.0	13.0	21.0	-
		Frequency band [Hz]	[50, 100]	[50, 100]	[50, 100]	[50, 100]	-
	τ_x [Nm]	Amplitude [N]	10.5	5.0	5.0	21.0	-
		Frequency band [Hz]	[50, 100]	[50, 200]	[50, 200]	[50, 100]	-
	τ_y [Nm]	Amplitude [N]	10.5	5.0	5.0	21.0	-
		Frequency band [Hz]	[50, 100]	[50, 100]	[50, 100]	[50, 100]	-
	τ_z [Nm]	Amplitude [N]	90.0	15.0	15.0	21.0	-
		Frequency band [Hz]	[50, 100]	[50, 100]	[50, 100]	[50, 100]	-
Reference	ψ [rad]	Amplitude [rad]	$\frac{\pi}{2}$	$\frac{\pi}{18}$	$\frac{\pi}{18}$	$\frac{\pi}{18}$	$\frac{\pi}{4}$
		Frequency [Hz]	3	2	2	2	6
		Phase [rad]	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	$[-\frac{\pi}{2}, \frac{\pi}{2}]$
	ϕ [rad]	Amplitude [rad]	-	$\frac{\pi}{14}$	$\frac{\pi}{24}$	$\frac{\pi}{24}$	$\frac{\pi}{24}$
		Frequency [Hz]	-	5	3	3	8
		Phase [rad]	-	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	$[-\frac{\pi}{2}, \frac{\pi}{2}]$
	θ [rad]	Amplitude [rad]	-	-	$\frac{\pi}{14}$	$\frac{\pi}{24}$	$\frac{\pi}{30}$
		Frequency [Hz]	-	-	5	3	11
		Phase [rad]	-	-	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	$[-\frac{\pi}{2}, \frac{\pi}{2}]$
	z [m]	Amplitude [m]	-	-	-	1	2
		Frequency [Hz]	-	-	-	1	0.8
		Phase [rad]	-	-	-	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	$[-\frac{\pi}{2}, \frac{\pi}{2}]$

Table 5: Hyperparameters of persistence of excitation signal and reference signal for each curricular step.

Additionally, this appendix holds the Q and R matrices used in the LQR theory for deducing the supervisor gains. The respective matrices are given in Eq. (16). The energy in the attitude angles is penalised harshly to promote the regulation of these states. The same logic was applied to the altitude state. The rates have received a reduced penalty as they may require to get high initially to regulate the attitude angles and altitude position. With regard to the inputs (the R matrix), these are not penalised as severely for the same reason as the aforementioned rates.

$$Q = \begin{bmatrix} 1e5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1e5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1e5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3e0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3e0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3e0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1e1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1e3 \end{bmatrix} \quad R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{16}$$

References

[1] T.J. Ludeña Cervantes, S.H. Choi, and B.S. Kim. Flight Control Design using Incremental Nonlinear Dynamic Inversion with Fixed-lag Smoothing Estimation. *International Journal of Aeronautical and Space Sciences*, 21(4):1047–1058, 2020.

[2] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th International Conference On Machine Learning*, ICML '09, pages 41–48. Association for Computing Machinery, 2009.

[3] A.M.C. Helmer, C.C. De Visser, and E. Van Kampen. Flexible heuristic dynamic programming for reinforcement learning in quad-rotors. In *AIAA Information Systems-AIAA Infotech at Aerospace, 2018*, number 209989, 2018.

[4] J. West, F. Maire, C. Browne, and S. Denman. Improved reinforcement learning with curriculum. *Expert Systems with Applications*, 158, 2020.

[5] Andrew Y. Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In Marcelo H. Ang and Oussama Khatib, editors, *Experimental Robotics IX*, pages 363–372, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[6] R. S. Sutton and A. G. Barto. *Reinforcement learning: An Introduction (2nd edition draft; 2012)*. MIT Press, 2012.

[7] T.S.C. Pollack and E. van Kampen. Safe curriculum learning for optimal flight control of unmanned aerial vehicles with uncertain system dynamics. In *AIAA Scitech 2020 Forum*, volume 1 PartF, 2020.

[8] John G. Kemeny and J. Laurie Snell. *Markov chains*. Springer-Verlag, New York, 1976.

[9] B. Langmead. Markov chains. University Lecture 22, John Hopkins Whiting School of Engineering, 2013.

[10] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

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

[12] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel. Trust region policy optimization. In *32nd International Conference on Machine Learning, ICML 2015*, volume 3, pages 1889–1897, 2015.

[13] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, T. Graepel, and D. Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[14] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[15] J. Burden and D. Kudenko. Using uniform state abstractions for reward shaping with reinforcement learning. In *ALA 2018 - Adaptive Learning Agents - Workshop at the Federated AI Meeting 2018*, 2020.

[16] Y.J. Lee and K. Grauman. Learning the easy things first: Self-paced visual category discovery. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1721–1728, 2011.

[17] M.E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

[18] J. Garcıa and F. Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[19] M.E. Taylor and P. Stone. Representation transfer for reinforcement learning. In *AAAI Fall Symposium - Technical Report*, volume FS-07-03, pages 78–85, 2007.

[20] M.E. Taylor, P. Stone, and Y. Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8:2125–2167, 2007.

[21] L. Wang, E.A. Theodorou, and M. Egerstedt. Safe Learning of Quadrotor Dynamics Using Barrier Certificates. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2460–2465, 2018.

[22] Subin Huh and Insoon Yang. Safe reinforcement learning for probabilistic reachability and safety specifications: A lyapunov-based approach. *arXiv preprint arXiv:2002.10126*, 2020.

[23] A. Basu, T. Bhattacharyya, and V.S. Borkar. A learning algorithm for risk-sensitive cost. *Mathematics of Operations Research*, 33(4):880–898, 2008.

[24] L. Torrey and M.E. Taylor. Help an agent out: Student/teacher learning in sequential decision tasks. In *Proceedings of the Adaptive and Learning Agents Workshop 2012, ALA 2012 - Held in Conjunction with the 11th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012*, pages 41–48, 2012.

[25] P. Geibel and F. Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence Research*, 2005.

[26] J. García and F. Fernández. Probabilistic policy reuse for safe reinforcement learning. *ACM Transactions on Autonomous and Adaptive Systems*, 2019.

[27] T. Mannucci, E. Van Kampen, C.C. de Visser, and Q.P. Chu. SHERPA: A safe exploration algorithm for reinforcement learning controllers. In *AIAA Guidance, Navigation, and Control Conference, 2013*, 2015.

[28] T. Mannucci. *Safe Online Robust Exploration for Reinforcement Learning Control of Unmanned Aerial Vehicles*. Delft University of Technology, 2017.

[29] T. Mannucci, E.-J. 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, 2018.

[30] F. Fernández and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the International Conference on Autonomous Agents*, volume 2006, pages 720–727, 2006.

[31] T.M. Moldovan and P. Abbeel. Safe exploration in Markov decision processes. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, volume 2, pages 1711–1718, 2012.

[32] L. White, R. Togneri, W. Liu, and M. Bennamoun. *Introduction to neural networks for machine learning*, volume 783. 2019.

[33] B. Kiumarsi, F. L. Lewis, H. Modares, A. Karimpour, and M. B. Naghibi-Sistani. Reinforcement Q-learning for optimal tracking control of linear discrete-time systems with unknown dynamics. *Automatica*, 2014.

[34] F. L. Lewis, D. Vrabie, and V. L. Syrmos. *Optimal Control*. John Wiley & Sons, 3 edition, 2012.

[35] B. Kiumarsi, F.L. Lewis, M.-B. 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, 2015.

[36] K.S. Narendra and A.M. Annaswamy. Persistent excitation in adaptive systems. *International Journal of Control*, 45(1):127–160, 1987.

II

Literature Study

2

Reinforcement Learning

The field of Reinforcement Learning (RL) is part of what is known as Machine Learning (ML) depicted in Figure 2.1. In ML, an algorithm learns to perform a specific task through experience, which resembles the way humans gain understanding about solutions to problems in their daily lives. From a theoretical standpoint ML is divided in three categories [30], namely *supervised learning*, *unsupervised learning* and RL. These methods have their respective benefits and drawbacks, however these will not be discussed in this literature study as they overstep the scope of this research. In order to form a better perspective to the division presented in Figure 2.1, a brief introduction on supervised and unsupervised learning will be given.

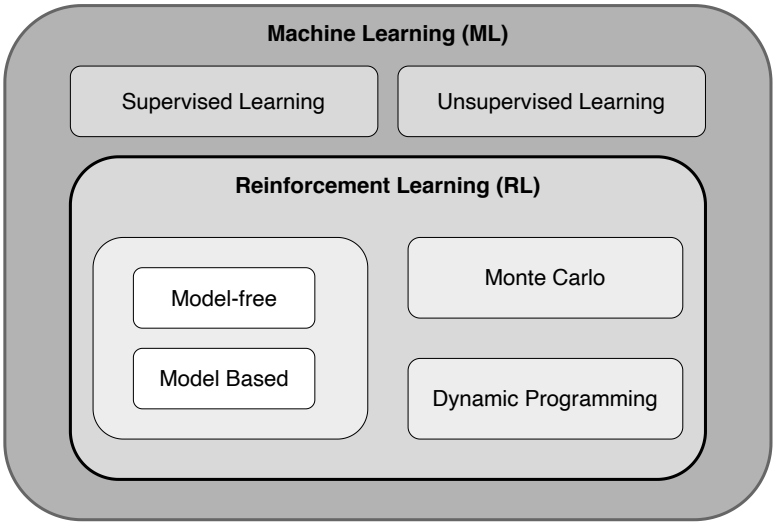


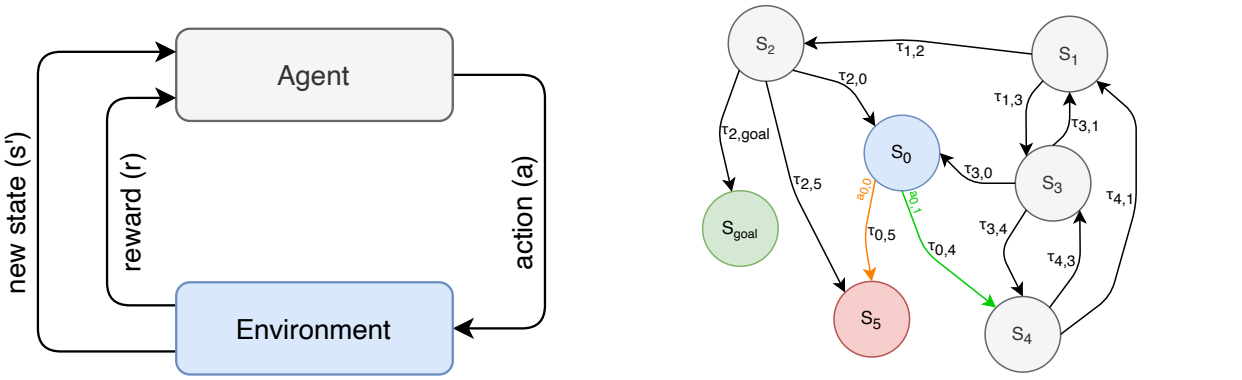
Figure 2.1: Machine Learning (ML) segmentation (interpretation of figure 1.3 from [30])

When talking about supervised learning, literature states the concept of gaining experience with supervision [24]. Where the term supervision indicates that the data on which the learner trains is labeled and categorised in some way prior to the training phase. Usually, the experience of the learner consists of finding patterns in the input data based on its labels. Two main uses are found in literature, the first of which is classification, where the learner categorises data in classes, and secondly, regressions, where on the basis of input data the learner predicts a single output. An example of this is the model construct presented in Section 6.3.3 using measurements to learn how to model a system.
On the other hand, there is the concept of unsupervised learning, where the goal is to discover patterns independently [24]. Mostly, the unsupervised learning is represented in clustering techniques and association algorithms, which seek correlations in the input data in order to group it.
Finally, there is RL. In contrast with the aforementioned methods, no data is required for the learner to gain experience. Instead, the learner will *reinforce* behaviour that leads to more optimal ways of reaching a specified goal within an environment [52]. Although this approach of learning is praised for many reasons, it has considerable drawbacks as well.

This chapter will provide some fundamentals with regard to the concepts comprised within this field of RL in Section 2.1. Furthermore, the mathematical basis of RL will be formulated in the form of an Markov Decision Process (MDP) in Section 2.2 accompanied with solutions for these processes in Section 2.3. Finally, an overview of two existing methods, SARSA and Q-learning, with their pro's and con's will be covered in Section 2.5 alongside more advanced RL methods in Section 2.6. The purpose of this chapter is to gain insight of RL and to discover its advantages when faced with complex systems.

2.1. Fundamentals

The most important concept of RL is that experience is gained by the learner through interaction with an environment [52]. In RL the learner is called an *agent* and it is put in an environment which will provide a new state, s', as well as a reward, r, based on the agent's action a. The process is represented by a flowchart in Figure 2.2a. In essence, the agent starts at an initial state, s_0, and has to find the most optimal way to a set goal state, s_{goal}, by choosing an action, a_i. All available actions to the agent are collected in the set $\mathcal{A}$. Likewise, all the possible states, s, are said to be part of the set $\mathcal{S}$, which is known as the State Space (SS) in control theory jargon.



(a) Overview of a Reinforcement Learning (RL) process

(b) A simple Markov Decision Process (MDP) Schematic; Blue is the initial state; Red is a representation of a terminal state; Green depicts the goal; $\tau_{i,j}$ represents the transition from i to j; $a_{i,p}$ is the p^{th} action at state i

Figure 2.2: Fundamental concept of RL overview with its translation in MDP form.

In Figure 2.2a, only the variables that define the interactions between the agent and the environment are displayed. However, this only lays the foundation for what the learning process entails. Indeed, only the playing board has been described so far and the focus of RL is to understand the player of the game, and its decision making process. The way an agent decides what action, a, to select at a given state is defined by a policy, π, serving as a translation between the state received by the environment and the action taken by the agent. On the bottom part of Figure 2.2a, there is the environment, mapping a state-action (s, a) combination to a new state with a respective reward by means of a transition, τ. All possible transitions are extracted from the transition function $\mathcal{T}$.

2.2. Markov Decision Process (MDP)

Although the fundamental concept of RL was presented in Section 2.1, it lacked a more formal and mathematical basis to construct a methodical approach, essential for further understanding of this learning method. Here enters the theory of an MDP. Based on Markov chains [21], an MDP represents the environment by a collection of interconnected blocks, each representing a state $s_i \in \mathcal{S}$ in which an agent could end up. Each block is connected to a subset of other blocks by links which represent state-transitions, $\tau_{i,k}$ generated by a transition function $\mathcal{T} : \mathcal{A} \times \mathcal{S} \to \mathcal{S}$. Dependent on the action, $a_{i,j} \in \mathcal{A}$, selected by the agent's policy at a state, one of the available transitions at that state is followed. These principles are depicted in Figure 2.2b. Each time the agent changes states, it receives a reward, r, according to a reward function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$, measuring the benefit gained by the current transition towards the final goal. In general, an MDP is defined by a tuple containing the aforementioned domains and functions, $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$.

Characterising an MDP, is the Markov property, informally stating that *the future is independent of the past given the present* [25]. In essence, meaning that the current state contains all the information relating to previously visited states, thus ascertaining a conditional independence on states visited in the past. The importance of this property cannot be underestimated, due to the policy's absolute dependence on the current state for the selection of an action. Indeed, the choice of a specific action at a given state (called a state-action pair) is formed by the agent's policy $\pi(s) = a$. Moreover, according to [52], the probability of an agent transitioning to a subsequent state s' is given by the probability $p(s', r|s, a)$, in which r stands for the reward that comes with this transition. Clearly, the reward given to an agent is dependent on the current state-action pair (s, a). Furthermore, the consideration could be made to include a dependence on the subsequent state s', based on how desirable that state would be. As an example, if s' were to be a state where the agent would be damaged or broken, called a terminal state (see definition 2.1), that state should be avoided (this concept will be elaborated on in Chapter 5). Consequently, in some cases the reward is additionally dependent on s', and described by $r(s, a, s')$.

Definition 2.1: Terminal State

A terminal state is a state for which all consequent states, whatever the action taken, will result in a reward $r_i = 0$ (with $r \in \mathbb{R}_0^+$).

Based on the explanation so far, the probability of reaching a given state can be expressed as an expected value problem given in Equation (2.1), which transcribes the probability of reaching a subsequent state, s', when the state-action pair, (s, a), occurs. Likewise, the formulation for the reward function can be described in this form, as shown in Equation (2.2). Since the reward is linked to a transition, the probability of receiving a given reward, is directly linked to the probability defined in Equation (2.1). When the subsequent state is known, such as is the case for methods like $SARSA$, the expected reward value is defined by Equation (2.3).

$$p(s'|s,a) = Pr\{S_t = s'|S_{t-1} = s, A_{t-1} = a\} = \sum_{r\in\mathcal{R}} p(s', r|s, a) \tag{2.1}$$

$$r(s,a) = \mathbb{E}\left[R_t|S_{t-1} = s, A_{t-1} = a\right] = \sum_{r\in\mathcal{R}} r \sum_{s\in\mathcal{S}} p(s', r|s, a) \tag{2.2}$$

$$r(s,a,s') = \mathbb{E}\left[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'\right] = \sum_{r\in\mathcal{R}} r\frac{p(s', r|s, a)}{p(s'|s, a)} \tag{2.3}$$

The above equations [52], solely considers the definition of an MDP environment. As shown in Figure 2.2a, however, the agent should be defined as well to complete the RL process. Note, throughout this chapter the *learner* and the *agent* are used interchangeably.
The aim for the agent is to achieve a predefined goal with the highest cumulative reward or lowest total cost depending on how the problem is defined. By definition, the cost or reward are ways of communicating the problem to the agent by the operator. It is understood that it should reflect the optimisation that is sought. An additional concept is the *expected return*, G_t, defined by [52] as representing the sum of all the future rewards, from the state, s, at time t. In case a reward is given to the agent (and not a cost), the agent's aim is to maximise G_t. Thus, considering the dependence of the reward function on the state-action pairs, the expected return highly depends on the agent's policy as well. Undoubtedly, this is an objective way of defining how valuable a state is to the agent's progression towards finding the optimal path. The issue with expected return, however, is its dependence on knowledge about future states. This poses a problem in tasks with an infinite horizon (that have no terminal state), or no predefined episodes, since the time required to end the task could be infinite, resulting in $G_t = \infty$. In order to avoid such situations, a *discount factor*, γ, is implemented (Equation (2.4)), which will discount the importance of rewards far away in time with respect to the expected return of the current state s.

$$G_t = \sum_{i=1}^{\infty} \gamma^i R_{t+1+i} \tag{2.4}$$

The remainder of this section covers the Bellman equation in Section 2.2.1. After which two characteristics of MDP's are discussed. First the difference between discrete and continuous MDP's is detailed in Section 2.2.2, followed by the probabilistic nature of transition functions in Section 2.2.3.

2.2.1. Bellman Equation

As mentioned, the learner will base its path through the MDP on the reward function through its policy. In order to shape the policy, state-action pairs are given a score, called *value*, reflecting the expected return from this pair. The function defining the value of all state-action pairs in an MDP is called the Bellman equation.
In order to understand the use of the Bellman equation, first, the concept of state value will be further introduced. The value of a state, v_π(s), given an agent's policy, π, is defined by Equation (2.5), being the expected value of G_t at a state s. This equation can be converted to be dependent on the value of the subsequent state,$v_\pi(s')$, as is proven in [52], resulting in Equation (2.6). This equation uses Equation (2.2) to find the reward, given a certain action a was taken. The latter having a probability of occurrence accorded by the policy, $\pi(a|s)$.

$$v_\pi(s) = \mathbb{E}_\pi\left[G_t|S_t = s\right] \tag{2.5}$$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\left[r + \gamma v_\pi(s')\right] \tag{2.6}$$

The equation above is known as the Bellman equation, and represents the basis policy evaluation laid out in Section 2.3.1. In this equation, the discount factor is given by γ.

2.2.2. Discrete and Continuous Markov Decision Processes

A distinction is to be made between continuous and discrete MDPs, where the difference lies in the segregation of states for discrete processes. This comparison closely resembles the difference between discrete and continuous statistics. In discrete processes, the amount of states for an agent to visit is finite, with a certain resolution of state space segregation. For example, when all state values are rounded to predefined decimal floating point precision. This comes with the advantage that state values can be compared easily, or tabular formulations can be made of the MDP, facilitating the path optimisation by the agent.

$$d(s_{t'}, s_{t''}) = \sqrt{\sum_{j=0}^{n} (s_{t',j} - s_{t'',j})^2} \tag{2.7}$$

On the other hand, there is the continuous MDP, where the amount of possible states is infinite. Although more common, it is also renders the more simplified tabular version of learning impossible to implement, as the table would require an infinite amount of cells. Consequently, a continuous value function has to be found, satisfying the Bellman equation (Equation (2.6)). Furthermore, the direct comparison of states becomes impossible, which calls for the definition of the euclidean distance between two states in Equation (2.7). When comparing states, one might want this distance to be below a certain threshold, as is the case for the closeness condition used by the Safety Handling Exploration with Risk Perception Algorithm (SHERPA) outlined in Section 5.3.3.

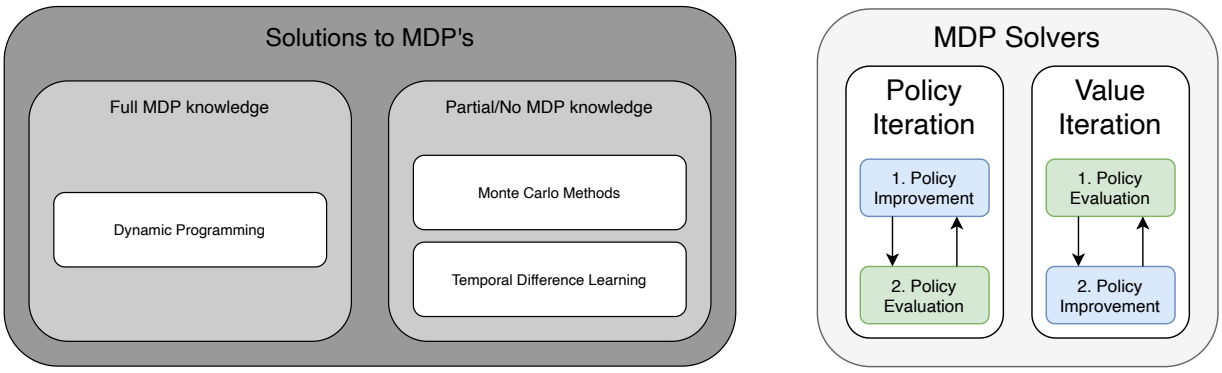
2.2.3. Stochastic and Deterministic Transition Functions

Transition functions, $\mathcal{T}$, can be categorised in two different groups based on how they connect states in the scope of MDP's. Transition functions are dependent on the current state as well as the selected action by the agent. In the case of deterministic transition functions, the state-action pair has a strict mapping to a new state, s', according to the transition function.
In case of stochastic transitions, the state-action pair will result in multiple possible outcomes. In this way, the transition function has a given probability to move the agent to a given state, s'_a. The transition could, however with a different probability, lead the agent to s'_b. Although in the example given, solely two possible outcomes are given with their corresponding discrete probability, in some cases the possible outcomes are plentiful, or even infinite which would require a respective continuous probability distribution.

Stochastic transition functions will lead to more difficult path optimisation by the agent, due to the more difficult estimation of the transition mapping.

2.3. Solution to Markov Decision Processes

By definition, *solving* an MDP is the feat of the agent finding an optimal path towards a predefined goal, based on the rewards acquired on that MDP. In other words, this is an optimisation process, aiming to find the best policy, π, which will guide the agent optimally through the MDP towards its goal. For this purpose, the taxonomy presented in Figure 2.3a shows three different approaches based on the level of knowledge of the MDP.



(a) Breakdown of the optimisation methods found in MDP theory, based on required knowledge of the MDP.

(b) Schematic representation of Policy Iteration (PI) and Value Iteration (VI).

Figure 2.3: Breakdown of the optimisation methods found in MDP theory (left) and the two methods they use internally: PI and VI (right).

All these methods make use of the same principles defined as *policy evaluation* and *policy improvement*. Based on the available knowledge about the topical MDP, these principles can be combined appropriately to solve the MDP optimally. Their internal process will be outlined in subsections 2.3.1 and 2.3.2, for policy evaluation and policy improvement, respectively. Followed by a brief summary of the methods presented in Figure 2.3a, which use the building blocks previously mentioned. A more elaborate explanation can be found in [38, 52].

2.3.1. Policy Evaluation

The purpose of policy evaluation is to find the true value function, $v_\pi(s)$, which determines the value of a given state, based on all future rewards conditioned by the agent's policy π. Basically, it means that this value function indicates how well a policy is doing on the MDP at hand. For achieving the true value function, an iterative process is used, which considers the value of the subsequent state, s', as shown in Equation (2.8) and resembling the Bellman equation, with the exception of, k, denoting the iteration index.

$$v_\pi^{k+1}(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma v_\pi^k(s') \right] \tag{2.8}$$

Resulting in the convergence of the value function at iteration k, $v_\pi^k(s)$, towards the true value function for a given MDP, $v_\pi(s)$, as $k \to \infty$. The convergence criterion in policy evaluation is given by Equation (2.9)

$$\forall s \in \mathcal{S} : \left(v_\pi^k(s) - v_\pi^{k+1}(s) \right) < \epsilon \tag{2.9}$$

When initialising the iteration process a random value function, $v_\pi^0(s)$, is defined from which future value functions will be derived.

2.3.2. Policy Improvement

Policy improvement is a similar process to policy evaluation, with the difference that the goal is to find the true policy, π, for the MDP at hand. The idea behind policy improvement is that at a given state, an action dictated by the current agent's policy, $\pi^k(s)$, might not be leading to the most optimal path. Instead, an action dictated by a modified policy, $\pi'(s) \neq \pi^k(s)$, might lead to a more optimal path in the long term. The latter is measured in terms of the respective value functions of both the current and modified policies, v_{π^k}, and $v_{\pi'}$. Should $v_{\pi^k} \leq v_{\pi'}$, then it is understood that by the definition of the value function, the modified policy is better at this state, and consequently overall, than the current one. A more formal mathematical formulation is given in Equation (2.10)

$$\pi^{k+1} = \begin{cases} \pi' & if \quad v_{\pi'}(s) \geq v_{\pi^k}(s) \\ \pi^k & if \quad v_{\pi'}(s) < v_{\pi^k}(s) \end{cases} \tag{2.10}$$

Should the case arise where both value functions are equal, then the true policy for the MDP has been reached. In general, this equality is hard to ascertain, requiring a convergence criterion, defined as given by Equation (2.11).

$$\forall s \in \mathcal{S} : \left(v_{\pi^k}(s) - v_{\pi^{k+1}}(s) \right) < \epsilon \tag{2.11}$$

Not the difference in position of iteration step superscript, k, compared to Equation (2.9), here denoting the iteration step of the policy, not the value function.

2.3.3. Policy Iteration and Value Iteration

Policy evaluation and policy improvement each assume a constant policy and value function, respectively, whilst iterating the other. Pushing this concept further, they could be interlinked where the value function and the policy are optimised to their respective true values concurrently. For this purpose, two methods are commonly used, namely PI and VI. How these methods use policy improvement and policy evaluation is shown in Figure 2.3b. Policy Iteration (PI) is initiated with some policy, π_0, which it will use to derive the true value function of that policy. However, the true value function can only be achieved in the limit. Assuming that the true value function, v_π^*, is found, PI will then perform policy improvement using the that value function. Due to the true function only being available in the limit, the policy evaluation step can be computationally exhaustive.
Therefore, PI's counterpart, Value Iteration(VI) can be used as alternative. In VI, the value function is updated by going at least once through all the states. This is called a *sweep* by [52]. In a single sweep, the value function is considered *true* and the policy evaluation step is considered finished. Which, like PI, means that policy improvement is performed, after which the cycle is restarted.
Both VI and PI are extremes of general policy iteration. It should be noted that the amount of refinement in each policy improvement and policy evaluation process is dependent on their respective inherent iteration steps, k. In some cases, for algorithm performance, the respective number of iterations can be set to high values, or in case of more stochastic MDP's, these steps can be set to lower values to avoid overfitting the policy and/or value function.

2.3.4. Dynamic Programming

The method of dynamic programming comes from the optimisation algorithm jargon. It is an optimisation method in parallel with linear programming (presented in [19]). This method requires full knowledge of the MDP, with the concordant perfect model of this MDP. Full knowledge will serve the update of both value function and policy. Respective optimisation processes will be performed over a given amount of states before switching to the other process. Ultimately, all information about the reward function and the transition function is known to the agent in order to use Equations 2.8 and 2.10.
Dynamic programming can make use of either PI or VI, depending on the MDP at hand. Usually VI is chosen for its ability to initiate optimisation without the knowledge of an initial policy. However, if the latter should be known, policy iteration should be used in order to optimise the guess of the next iterations within policy improvement.
Dynamic programming comes with a major flaw, namely the *curse of dimensionality*[2]. Due to the nature of its use of policy evaluation and policy improvement iteration processes. Indeed, dynamic programming covers all the possible states through the iterations, in order to have full knowledge of the MDP. Although accurate, surely, this induces a more elaborate iteration process as the state-space dimensions increases. The latter is known as the curse of dimensionality, which was first introduced in [2].

2.3.5. Temporal Difference

In contrast to dynamic programming, Temporal Difference (TD) methods are used when little to no knowledge is available about the MDP. As such, agents solving an MDP using TD methods will optimise the path to the goal state, s_{goal}, from experience. Thus, involving a rigorous sampling process, to collect expertise on the topical MDP.
Methods under the TD umbrella, will use PI or equivalently VI, depending on what the task requires. For the iterative updates of the value function and policy used inherently by PI and VI, the more simplistic TD method, $TD(0)$, will use only the sample of a subsequent time step for the updates. The formulation of the value function is given in Equation (2.12), with k indicating the iteration step of the value function, γ is the discount factor and α is the learning rate.

$$v_\pi^{k+1}(s_t) = v_\pi^k(s_t) + \alpha\left[R_{t+1} + \gamma v_\pi^k(s_{t+1}) - v_\pi^k(s_t)\right] \tag{2.12}$$

The name of this method was shortly mentioned as $TD(0)$, in which the 0 denotes that this is the one-step TD method. As can be deducted from Equation (2.12), only a single subsequent time step is sampled and considered in the update. A vaster, more general theory of $TD(\lambda)$ methods exist where more sampling is used before an update of the value function or policy is performed. Although these methods would overstep the scope of this literature study, the reader is invited to find more details in [5, 49, 52].

In TD methods, the policy is improved in accordance to the value function, since the policy is defined as Equation (2.13), making the agent take the action that will lead to the state with the highest value. Consequently, once the value function is updated, the policy will automatically take its new form.

$$\pi(a|s) = \underset{a}{\operatorname{argmax}}\left(\mathbb{E}[v_\pi(s_i)|a = a_i]\right) \tag{2.13}$$

To complete the jargon of TD methods, the concept of TD-error is defined as $\delta_t = R_{t+1} + \gamma v_\pi^k(s_{t+1}) - v_\pi^k(s_t)$, which appears on the right hand side of Equation (2.12). The error represents the correction of the value function, v_π^k, based on the acquired experience between time steps t and $t+1$. The TD-error is closely related to bootstrapping [52].

2.3.6. Monte Carlo

A different method to consider when a perfect model of an MDP is lacking, is the Monte Carlo (MC) method. Indeed, when only partial or no knowledge about the MDP is accessible, MC methods can be utilised to optimise the path to the goal state, s_{goal}. The latter is derived from the statistical theorem of large numbers, where it is used to average sample outcomes [44]. Applying this method on MDPs for policy evaluation, enables to estimate the value of given state by averaging the reward of subsequent states. Over time, state-action pair rewards will become closer and closer to the their respective true rewards. By definition, (see Equation (2.8)), this means that the value function of the MDP will indeed average out towards its true instance.
Regarding the policy improvement step, the optimal policy is found by initially finding the average reward of state-action pairs which, as outlined in the previous paragraph, will converge to their respective truth with accruing samples. Based on the current estimate of those rewards, the policy is defined as the agent taking the action with the highest estimated reward.

The amount of samples required for the update of both policy and value function depends on the type of task at hand. For episodic tasks, where a terminal state (see definition 2.1) is guaranteed, the former are respectively updated based on Equations 2.10 and 2.8. The update is performed once all the samples, until either a terminal state or the end of episode were reached, are collected. When a continuous task is known to be performed by the agent, a predefined number of samples, called a *batch*, are collected before the policy evaluation and policy improvement are performed using the newly estimated state-action pair rewards. This specific regulation of samples is what differentiates MC methods from TD methods.

2.4. Back to Reinforcement Learning

The understanding of MDPs is of importance for reinforcement learning, more specifically for the understanding of the environment in which the agent is embedded during learning. Indeed, MDPs form the backbone of the environment, they are the formal mathematical formulation of the latter. When implementing RL, the task has to be divided in its environment, and its agent (see Figure 2.2a), where the environment is to be translated in the form of an MDP. As a consequence, the environment will be defined by its inherent MDP's tuple, $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$. During the implementation it is of great importance to identify these components, in order to appropriately determine the learning strategy of the agent, discussed in more detail in Section 2.4.1.

The environment has two principal tasks. First of which is the propagation of the agent's state based on a corresponding action taken by the former. When the RL process is done within a simulation, this inquires the use of a state space system, continuous or discrete depending on the topical task. The state space system is the mathematical representation of the transition function. Simulation instances a priori require full knowledge of system dynamics, whereas real-life experiments do not. Although more research-intensive due to the required knowledge, simulations come at a greater benefit due to shunning of damage to the agent when terminal states are encountered (more on this matter in Chapter 5). In contrast, real-life experiments have the advantage of abstinence of system dynamics knowledge, but come with the drawback of damaged or broken agents when terminal states occur.
The second task of the environment is to provide the agent with a reward, r, for the transition resulting from the given state-action pair. The reward function is predefined as part of the environment and translates the task from the operator to the agent. In some cases, such as in Section 3.2.1, reward is swapped for cost which is to be minimised by the agent.

A particular subject for complication is the RL process in continuous state spaces, certainly highly non-linear ones, due to the complex generalisation and poor predictability of the state transitions. Furthermore, the agent's policy is more complex to find, as it has to fit an infinite amount of states correctly.

2.4.1. Types of learning

Learning can happen on a multitude of levels, based on what authority the agent has on the learning process. Although filled with more categories, three will be discussed in this subsection. The comparison will be made between on- and offline learning, the discrepancies between on- and off-policy will be discussed after this, and finally, the concepts of on-board and off-board learning will be elaborated.

Online and offline

A first distinction to be made in the learning methods of an agent is the difference between online and offline learning. The former is closely related to continuous tasks in MDP theory, where no episodes are predefined. In online learning the agent has no authority to restart a simulation, and as a consequence has to learn *on the spot*. On the other hand, offline learning grants the agent the capability of resetting the learning progress to an initial state.
An example will serve as clarification for the aforementioned points. On can imagine an agent performing a task, such as learning to ride a bicycle. For an unfortunate wrong series of actions, due to a non-optimal policy, the bicycle falls over. This would be known as a terminal state, from MDP theory (definition 2.1). If the agent is implemented using online learning, it will be stuck there until a termination criteria, separate from reaching a terminal state, is reached. However, in the case of offline learning, the agent would recognise the finality of the episode and would reset the bicycle to an initial condition. In this example it is clear that using online learning would be a risk, assuming no external help is provided to the agent. It should be noted, though, that in some cases, comparative to real-life implementations, where the agent can not possibly reset the learning process, online learning plays an important role.
Of course a combination of both concepts can be used to benefit from their respective situational advantages. For example, on complex environments, one might initiate the agent offline, in order for it to have basic understanding of the first few time steps. After which online learning could be used to further evolve the understanding, even in real-life experiments.

On-policy and off-policy
The focus of the difference between on-policy and off-policy lies in the formulation of the policy evaluation update (Equation (2.8)) and policy improvement update (Equation (2.10)). More specifically, the variable for policy, π, and the improved policy, π', conditioning the value functions in these equations.
When speaking of on-policy learning, the actions shown in Equation (2.8) are generated by the same policy that conditions the value function. Similarly, for policy improvement, the policy that is improved is the one that was used for comparison in Equation (2.10).
Off-policy, on the other hand, uses one policy, π_{cond}, by which the value function is conditioned in the update of policy evaluation (Equation (2.8)), and another, $\pi_{actions}$ to generate the agent's actions. The conditioning policy, π_{cond}, will is derived from past experience while the secondary policy, $\pi_{actions}$, is generally being experimented with at that time. Once a convenient policy, $\pi_{actions}$, is found, it is then used to replace the conditioning policy, π_{cond}.

On-board and off-board
As the name states, on-board and off-board learning relate to the location the learning process (VI and PI) is taking place. When considering the example of a small quadcopter with limited computation power on-board, the iterative updates of Equations 2.8 and 2.10 are performed off-board. This is called off-board learning. On the contrary, on-board learning will perform the necessary computations for these updates on-board. The latter requires more powerful computers, and is more applicable in larger craft such as the Cessna Citation II.

2.5. SARSA and Q-Learning algorithms

This section will discuss two methods used in RL on MDPs representing linearised dynamics. More specifically the model-free methods, Q-learning and SARSA, accompanied with their respective conveniences and drawbacks. Both methods are derived from TD theory and appear regularly in literature as the standard reinforcement learning implementations. Other methods exist, such as advanced MC algorithms, however these would overstep the scope of this literature study. Consequently they will not be discussed here, though, an introduction to these methods is made in [38], and presented in more detail in [52]. Dynamic programming methods are not discussed as part of RL due to their inherent dependency on full knowledge of the MDP, which for this research is assumed to be unavailable.
In order to understand SARSA and Q-learning fully, the concept of an action-value function is given in Equation (2.14). The formulation resembles the state-value function, previously simply called the value function, $v_\pi(s)$.

$$q_\pi(s,a) = \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{k+t+1} | S_t = s, A_t = a\right] \tag{2.14}$$

The main difference with the state-value function, is that $q_\pi(s,a)$ represents the value of a state-action pair. It not only evaluates how valuable the state is, but how appropriate the selected action is at that state. Thus, representing a more exhaustive valuation of the policy.

2.5.1. SARSA

The first algorithm to be discussed is one known as SARSA. The name stems from the algorithms use of current state, action and reward as well as the subsequent state and action, $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$, spelling *SARSA*. The method is an on-policy method that can be implemented in either on-board or off-board fashion. Furthermore, both online and offline implementations exist.
When using SARSA the aim is to evaluate the near-optimal, action-value function, $Q(s,a) \approx q^*_\pi(s,a)$, using sample based learning, from which the near-optimal policy, $\pi^*(s)$, is derived. The policy at state s_t in SARSA is defined as taking the action for which the estimate of the value function at update k, $Q^{(k)}(s_{t+1}, a)$, is maximal. The maximum is iterated over all the possible actions, $a \in \mathcal{A}$, at that state.
Algorithm 2.5.1, specifies the method used to force exploration of the state space, namely the $\epsilon - greedy$ method, where $\epsilon \in (0, 1]$ is a small number representing the probability of exploration. The remaining probability, $1 - \epsilon$, will dictate how conservative the SARSA algorithm is. That is, the update is considered conservative or *greedy*, when it exploits the knowledge accumulated so far (as stated in definition 1.7). The danger with *greedy* implementations is that they tend to get stuck in local optima. In RL terminology, greedy approaches would not maximise reward in the long term, but in the short term. On the downside, when the algorithm is not greedy enough (i.e. ϵ is too large), the learning process will not find a near-optimum, due to the frequent randomisation.

SARSA is an on-policy algorithm, which can be seen in algorithm 2.5.1, line 9, where $Q^{(k)}(s_{k+1}, a_{k+1})$ uses the action a_{t+1} defined by the current policy estimate, $\pi^{(k)}(s_{t+1})$. It serves as an integral part of the action-value function update. This specific line in algorithm 2.5.1, is reminiscent of the update proposed in TD methods (Section 2.3.5), because SARSA is an application of such method. Similar to TD theory, SARSA implementations that look at more than one sample for the update do exist and are called $SARSA(\lambda)$. These use eligibility traces in time history for the action-value update. Such methods are discussed in more detail in [4], and will not be elaborated in this literature study, as they exceed its scope.

Algorithm 2.5.1: ϵ-greedy SARSA [52]

Input : Step size $\alpha \in (0, 1]$; small $\epsilon > 0$; discount factor γ
Output : $\forall a \in \mathcal{A}, \forall s \in \mathcal{S}$: Near optimal action-value function, $Q(s, a) \approx q^*_\pi(s, a)$, and policy $\pi(s) \approx \pi^*(s)$

1 Initialise $\forall a \in \mathcal{A}, \forall s \in \mathcal{S}$: $Q^{(0)}(s, a)$ arbitrarily except that $Q^{(0)}(s_{terminal}, \cdot) = 0$;
2 Initialise s_0;
3 Initialise $\pi^{(0)}(s_0) \leftarrow \underset{a'}{\operatorname{argmax}}(Q^{(0)}(s_0, a'))$;
4 Initialise $a_0 \leftarrow \begin{cases} \pi^{(0)}(s_0) & \text{probability: } 1 - \epsilon \\ \text{random } a_0 \in \mathcal{A} & \text{probability: } \epsilon \end{cases}$;
5 **for** $k \in [0, 1, 2, 3, 4, ..., T]$ **do**
6 Take a_t, observe R_{t+1}, s_{k+1};
7 $\pi^{(k)}(s_{k+1}) \leftarrow \underset{a'}{\operatorname{argmax}}(Q^{(k)}(s_{k+1}, a'))$;
8 $a_{k+1} \leftarrow \begin{cases} \pi^{(k)}(s_{k+1}) & \text{probability: } 1 - \epsilon \\ \text{random } a_{k+1} \in \mathcal{A} & \text{probability: } \epsilon \end{cases}$;
9 $Q^{(k+1)}(s_t, a_t) \leftarrow Q^{(k)}(s_k, a_k) + \alpha[R_{k+1} + \gamma Q^{(k)}(s_{k+1}, a_{k+1}) - Q^{(k)}(s_k, a_k)]$;
10 $s_k \leftarrow s_{k+1}$;
11 **end**

SARSA algorithms are considered to be on the more conservative side, due to their on-policy nature. This is represented in the *cliff walking experiment* [52]. Looking one step ahead, SARSA algorithms find that a given set of actions at the subsequent state might result in one of the cliff states. Thus, lowering the action-value of those state. Consequently, the policy will be opting for a safer path. This is illustrated in Figure 2.4, where the shortest path to the goal, G, is desired. With each step taken resulting in a reward of -1 except for the landing on the cliff states, resulting in a reward of -100.

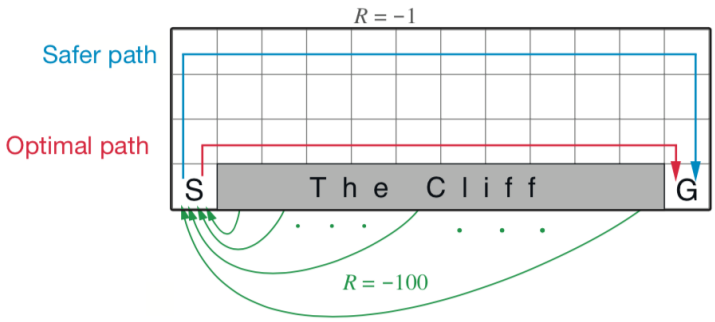


Figure 2.4: Cliff walking experiment discerning safe path (SARSA) and optimal path (Q-learning) (figure taken from [52], p.132)

2.5.2. Q-Learning

Comparatively, Q-learning is the off-policy counterpart of SARSA, thus aiming to find the optimal action-value function, $q^*(s, a)$, and optimal policy $\pi^*(s)$.The general Q-learning method is outlined in algorithm 2.5.2. The difference with SARSA lies in line 9, more specifically the term $\max_{a'}(Q^{(k)}(s_{k+1}, a'))$. The term takes the maximum value for $Q(s_{k+1}, a')$ of an action that is independent of the one selected by the policy in line 8. More specifically, the update of Q-learning's action-value function is independent of the randomness induced by the ϵ-greedy exploration. Since the value for $Q(s_{k+1}, a')$ is not related to the action imposed by the policy, it is an off-policy algorithm.

Algorithm 2.5.2: ϵ-greedy Q-learning [52]

Input : Step size $\alpha \in (0, 1]$; small $\epsilon > 0$; discount factor γ
Output : $\forall a \in \mathcal{A}, \forall s \in \mathcal{S}$: Optimal action-value function, $Q(s, a) \approx q_\pi^*(s, a)$, and policy $\pi(s) \approx \pi^*(s)$

Initialise $\forall a \in \mathcal{A}, \forall s \in \mathcal{S}$: $Q^{(0)}(s, a)$ arbitrarily except that $Q^{(0)}(s_{terminal}, \cdot) = 0$;
Initialise s_0;
Initialise $\pi^{(0)}(s_0) \leftarrow \underset{a'}{\operatorname{argmax}}(Q^{(0)}(s_0, a'))$;
Initialise $a_0 \leftarrow \begin{cases} \pi^{(0)}(s_0) & \text{probability: } 1 - \epsilon \\ \text{random } a_0 \in \mathcal{A} & \text{probability: } \epsilon \end{cases}$;
for $k \in [0, 1, 2, 3, 4, ..., T]$ **do**
    Take a_t, observe R_{t+1}, s_{k+1};
    $\pi^{(k)}(s_{k+1}) \leftarrow \underset{a'}{\operatorname{argmax}}(Q^{(k)}(s_{k+1}, a'))$;
    $a_{k+1} \leftarrow \begin{cases} \pi^{(k)}(s_{k+1}) & \text{probability: } 1 - \epsilon \\ \text{random } a_{k+1} \in \mathcal{A} & \text{probability: } \epsilon \end{cases}$;
    $Q^{(k+1)}(s_t, a_t) \leftarrow Q^{(k)}(s_k, a_k) + \alpha[R_{k+1} + \gamma \cdot \underset{a'}{\max}(Q^{(k)}(s_{k+1}, a')) - Q^{(k)}(s_k, a_k)]$;
    $s_k \leftarrow s_{k+1}$;
end

Due to the lack of impact of the ϵ-greedy method, the action-value function update will not consider the potential miss step in the *cliff walking experiment*, thus opting for the optimal path in Figure 2.4. The downside, however, is that there is more risk involved due to the policy having a randomness factor that might induce the agent to end in cliff states.

2.6. Policy Gradients

Previous section focused on two fundamental algorithms of RL, namely Q-learning and SARSA, which aim to find policy and value functions. Another group of methods in RL focuses on the policy gradient to determine an optimal policy. One such method is the Deep Deterministic Policy Gradient (DDPG), first presented in [28]. DDPG is a form of deep reinforcement learning, specifically designed to optimise an agent's action in large and more importantly, continuous action spaces. Deep reinforcement learning techniques are fundamentally more complex to implement. Other methods part of this category are Trust Region Policy Optimisation (TRPO) [47], Proximal Policy Optimisation (PPO) introduced in [48] and more. Finally, there are actor-critic structures used for continuous state and action spaces [17].

Methods presented in this section have been deemed outside of the scope of this thesis, due to their advanced implementation requirements. As stated previously, this thesis will serve the purpose of determining a safe curriculum reinforcement learning method for systems with unknown dynamics. Consequently, the focus will lie on the identification of system characteristics. Thus, it is opted for a simple RL implementation which still profits of RL's model-free nature.

2.7. Synthesis

Reinforcement Learning (RL) is an interesting field of machine learning by virtue of its independence on external data, such as system dynamics. Indeed, RL requires an interaction between a learner (agent) and an environment. The former will then, by virtue of intelligent trial-and-error, optimise a path from an initial state to a goal state whilst avoiding terminal states. The mathematical backbone of RL is defined in terms of Markov Decision Processes (MDP)s, which formalise the learning process and allow for a mathematical approach to task optimisation. Generally solved using a combination of policy evaluation and policy improvement, where the order is dependent on the topical task. Two methods are prominent in the RL field for MDPs representing linear systems, namely $SARSA$ and Q-learning. The first being on-policy and having the advantage that it is conservative in dangerous environments, with the associated drawback that the path is near-optimal instead of optimal. On the other hand, the Q-learning algorithm, being off-policy, is not as conservative and safe as SARSA, but ensures an optimal path through the MDP.

Furthermore, acknowledgement is given to policy gradient methods, such as DDPG, TRPO and others. However, they were deemed out of the scope of this thesis, due to their more elaborate implementation requirements. Since RL is not the focus of this work's contribution, a less demanding RL method is preferred.

This chapter provides insightful information regarding the advantages RL provides to the control of complex systems. Indeed, it gives the understanding that RL methods do rely solely on the dimensions of system dynamics. They do not, however, depend on a model of the mentioned dynamics, by virtue of their approach to task optimisation. The latter is based on an interaction between an agent and an environment, through which a policy is learned. Ultimately, this policy will serve as the controller for the system. Compared to classical methods which require a model to be constructed of the system, RL methods lack such requirement. Thus, removing a time consuming process of classical controller design methods.

Further research should be performed with regard to non-linear systems and their MDP representation. Moreover, the agent representation of policy and value function will be different as well due to the non-linear format of the system.

3

Linear Optimal Control

Control has become an integral part of engineering during the past century. Evolving from strictly manual control for the vast majority of human history to autonomous controllers in modern era of engineering. As such, enabling comprehensive control of unstable systems such as modern day fighter jets as well as self-driving cars on the road. The rise of automation in control in the past decades has resulted in the expanding impact this field of research has on the current and future world.

This chapter will give a brief introduction to the levels of control, gauging the level of automation and the role of a human operator in Section 3.1. As such, placing Reinforcement Learning (RL) on the ladder of control automation. Furthermore, this chapter serves as a stepping stone between the theoretical model-free structure proposed by RL in Chapter 2 and the practical implementation of actually controlling systems. Complying with the thesis subject, the focus will be set on autonomous control and more specifically on how to characterise optimality for this level of control. Indeed, Section 3.2, will cover two types of tasks, Linear Quadratic Regulation (LQR) and Linear Quadratic Tracking (LQT), which are closely related as will become apparent in these sections. These tasks are compliant with a implementation with RL algorithms such as Q-learning presented in Section 2.5.2. The more particular case of optimised control on linear systems is then addressed in Section 3.3 Finishing the chapter with a conclusion and suggestion for further research in Section 3.4.

Table 3.1: Jargon translation between Chapter 2 and current chapter.

Name	RL Jargon	Control Jargon
State	s	x
Action	a	u

This chapter will use the control theory jargon, for compliance with literature and facilitate the reader's understanding when expecting such vocabulary. However, for the reader coming from Chapter 2, some translation has to be presented. The translation of the concepts used in both the previous and the current chapters are given in Table 3.1. This translation is essential for the comprehension of this chapter as well as the preliminary analysis.

Finally, the scope of this chapter has been limited to linear systems, as the contribution of this work consists in the system identification capability of safe curriculum learning. Consequently, it is desired to simplify the other components of this work, in order to comply with the time frame that is available.

3.1. Levels of Control

Control is a large field of research applicable to many facets of the modern society, ranging from controlling the temperature in a toaster to flying a rocket's second stage to Mars. This section will categorise the many facets of control on a scale of automation, where the role of a human operator is used as a metric. For this specific metric, cybernetic research has an important impact defining the way humans and machines interact.
In more complex systems such as aircraft cockpits, these levels of control are generally blended, to allow for task separation between computer and operator. As such, focusing the concentration of the operator on the most important or urgent tasks. The classification is made based on the pillars[1] used by the faculty of aerospace at TU Delft University of Technology [59].

[1] `http://cs.lr.tudelft.nl/education/msc-thesis-2/`, accessed on 15-07-2020

3.1.1. Manual Control
When no automation is used in the control of a system, and consequently, the system's behaviour is solely dependent on physics and the human operator, the control is considered manual. Manual control is highly dependent on the operator's behaviour, its situational awareness and its capabilities. Indeed, the controller is external to the system that is to be controlled.

3.1.2. Automatic Control
In automatic control, a human operator does not intervene directly. Generally, built by means of electronic or mechanical systems, it regulates and controls a system by its own. It does require human input to set control parameters and milestones for controller to reach. However, the control is done on a stand alone basis. An example for this, is the altitude-hold mode of an aircraft's autopilot, which takes an altitude as input from an operator and will single-handedly control the aircraft to keep that altitude.

3.1.3. Supervised Control
Taking it a step further is supervised control. The operator continuously receives information from the controller, and can intermittently intervene in the control loop. In essence, the human in the loop is mostly there to supervise what the near-autonomous controller is doing, and will perform corrections when and where needed.

3.1.4. Autonomous Control
Finally, there are controllers that are not dependent on human operators at all. These are in the level of autonomous control, where the computer sets milestones, reaches these milestones and does not specifically require human input. However, information is regularly provided to the operator. The purpose is that controllers within this level have the ability to be completely independent from human help.

3.2. Optimal Control
Optimal control is the field of control that aims to mathematically optimise an objective function from a dynamical system's control. In contrast with RL, optimal control focuses on dynamic systems that are defined by ordinary differential equations, as is shown in Equation (3.1). In this equation, the dynamics are affine in input and state, however, the expression can be generalised further for dynamics that are non-affine in state and/or input. Furthermore, Equation (3.1) in comprised of several elements, where x is the state, $f(x)$ represents the internal system dynamics, $g(x)$ the system's input dynamics which is multiplied with, u, the input.

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

A special case for defining system dynamics, is the state space system. A standard definition of such system is defined in 3.2 and 3.3 for the continuous-time and discrete-time formulations respectively. In these equations and the remainder of the literature study, the subscripts c and d, will accordingly denote "continuous" and "discrete".

$$\begin{cases} \dot{x} &= A_c x + B_c u \\ y &= C_c x + D_c u \end{cases} \tag{3.2}$$

$$\begin{cases} x_{k+1} &= A_d x_k + B_d u_k \\ y_k &= C_d x_k + D_d u_k \end{cases} \tag{3.3}$$

3.2.1. Linear Quadratic Objective
The objective function that is optimised in optimal control, is one related to energy in systems. Given by Equation (3.4) [27], it gives information about the amount of energy the state x and the input u contain. Additionally, Q and R are weighing factors that represent the importance of each respectively. The objective is quadratic in the state and the input, whilst both of these are derived from a linear state space system, hence the name linear quadratic.

$$r(x,u) = x^T Q x + u^T R u \tag{3.4}$$

This objective function represents a cost, that is to be minimised by the optimisation process. Mathematically, the optimal value function for Equation (3.4) is defined by Equation (3.5). Note the discrete nature of this equation. Finally the optimal policy for linear quadratic problems is defined as the one generating the input u for which the value function at state x is minimised (i.e., $\frac{\partial V^*(x_k)}{\partial u_k} = 0$). Applying this to Equation (3.5), results in the general formulation of the optimal policy, Equation (3.6).

$$V^{(*)}(x_k) = \min_u (r(x,u) + V^*(x_{k+1})) \tag{3.5}$$

$$u_k^* = \frac{1}{2} R^{-1} g^T(x) \frac{\partial V^*(x_{k+1})}{\partial u_{k+1}} \tag{3.6}$$

3.2.2. Algebraic Ricatti Equation

In order to find the optimal policy from the Hamilton-Jacobi-Bellman (HJB) formulation, given in Equation (3.6), which is non-linear by nature, the Algebraic Ricatti Equation (ARE) comes into play. This equation has two variants, one is for continuous-time and the other is for discrete-time. This report, and consequently this section will focus on the Discrete-time Algebraic Ricatti Equation (DARE), due to the nature of its applicability on discrete-time state space systems considered in further implementations of this theory in Chapter 8. Assuming the value function is defined as $V(x_k) = \frac{1}{2} x_k^T P x_k$, the DARE is defined in Equation (3.7) [22, 43]. Where, γ is the discount factor and A_d, B_d are the discrete-time internal dynamics and input dynamics of the system respectively.

$$P = \gamma A_d^T P A_d - \gamma^2 (A_d^T P B_d)(R + \gamma B_d^T P B_d)^{-1} (B_d^T P A_d) + Q \tag{3.7}$$

$$K^* = (R + \gamma B^T P B)^{-1} \gamma B^T P A \tag{3.8}$$

Solving for the kernel matrix, P, will provide the necessary information to find the optimal policy for the given discrete-time state space system. The optimal control gain, K^*, is defined by Equation (3.8), and forms the basis for the optimal control policy, $u^* = -K^* \cdot x_k$ [22, 27].

There are multiple ways of solving the DARE, mainly split in a category where full knowledge of system dynamics is guaranteed, such as Lyapunov iteration, and methods that do require partial or no knowledge of the system, such as Q-learning. The aim of this study is to find the solution to DARE, for systems with unknown dynamics (definition 1.8). Consequently, Q-learning will be the method of choice in this report and Lyapunov iteration will not be discussed. The reader is invited to find more information on the latter method in [27, 38].

3.2.3. Linear Quadratic Regulation (LQR)

Once the basic principles of optimal control are understood, the introduction to linear quadratic regulators will be given in this section. The aim of a controller in an LQR problem is to find a control gain, K, such that the system's state, x, and input, u, contain the least amount of energy possible when combined. In essence the purpose is to find a policy that will get the system to a standstill and keep it at that.

The state space used for LQR is given in Equation (3.3). Since this has the same format as the general linear quadratic optimal control theory, the objective function is defined as in Equation (3.4). Equivalently, the LQT DARE is given by Equation (3.7).

3.2.4. Linear Quadratic Tracking (LQT)

Instead of regulating a system, the LQT task aims to find a policy that will force the system to follow a reference signal. This signal is generated in a discrete-time fashion formulated by Equation (3.9) [22]. The purpose is to minimise the error between the system's output, y_k, and the reference state, x_k^r, at any time step k. Hence, the error is defined as $e_k = y_k - x_k^r$. With the assumption that $D_d = 0$ (from Equation (3.3)), the error is defined by Equation (3.10).

$$x_{k+1}^r = F_d x_k^r \tag{3.9}$$

$$e_k = C_d x_k - x_k^r = \begin{bmatrix} C_d & -I \end{bmatrix} \begin{bmatrix} x_k \\ x_k^r \end{bmatrix} \tag{3.10}$$

A way of understanding the LQT task, is by seeing that minimising the error, e_k, is the same as *regulating* that error, allowing the use of LQR theory [22]. Based on this understanding, the state space system has to be modified in order to make use of Equation (3.4), the linear quadratic objective function [27]. The augmented state space is just that, with the augmented state defined as $X_k = \begin{bmatrix} x_k & x_k^r \end{bmatrix}^T$.

$$\begin{aligned} X_{k+1} &= \begin{bmatrix} A_d & 0 \\ 0 & F_d \end{bmatrix} X_k + \begin{bmatrix} B \\ 0 \end{bmatrix} u_k = T_d X_k + B_{1,d} u_k \\ e_k &= C_1 X_k \end{aligned} \tag{3.11}$$

The discrete-time augmented state space is shown in Equation (3.11), where the matrix $C_1 = \begin{bmatrix} C_d & -I \end{bmatrix}$. The augmented state is propagated in discrete-time and the output of the state space system has become the error between the reference and the system's state. The quadratic objective function has to be adapted as well, which is done in Equation (3.12), where $Q_1 = C_1^T Q C_1$ is the modified Q weighing factor for the LQT task. Finally, the DARE, from Equation (3.7), is adapted to Equation (3.13), changing $A_d \rightarrow T_d$ and $B \rightarrow B_{1,d}$[22].

$$r(X,u) = e^T Q_1 e + u^T R u \tag{3.12}$$

$$P = \gamma T_d^T P T_d - \gamma^2 (T_d^T P B_{1,d})(R + \gamma B_{1,d}^T P B_{1,d})^{-1}(B_{1,d}^T P T_d) + Q_1 \tag{3.13}$$

The discount factor is still given by γ, and solving Equation (3.13) for kernel matrix, P, will induce the optimal control gain K_1^* accompanied with the optimal policy, u^*, given in Equation (3.14).

$$\begin{aligned} K_1^* &= (R + \gamma B_{1,d}^T P B_{1,d})^{-1} \gamma B_{1,d}^T P T_d \\ u^* &= -K_1^* X \end{aligned} \tag{3.14}$$

With the above optimal control policy, the system is proven to converge in a stable manner to the reference state trajectory generated by Equation (3.9) in [22].

3.3. Optimised Control for Linear Systems

Up to this point, it has been assumed that *when* the DARE is solved for the kernel matrix, P, the optimal policy can be found. The solution of that equation, however, is not trivial to find. In this section, Q-learning, that was introduced in Section 2.5.2, will be developed as the RL method of choice to find the solution of the DARE [22, 23]. First the Q-function for optimal control of linear systems is presented in Section 3.3.1, after which implementation of both Policy Iteration (PI) and Value Iteration (VI) algorithms, introduced in Section 2.3.3, in optimal control will be covered.

3.3.1. Q-learning

Introduced in Section 2.5.2, the Q-learning method estimates the optimal action-value function, also called the Q-function, and the optimal policy of the agent. Once the Q-function derived, online VI and PI will be introduced for the LQT task. Accompanied with a deeper look at Persistent Excitation (PE) for Q-learning on Linear Quadratic (LQ) tasks.

The Linear Quadratic Q-function

The Q-function is defined as the sum of the reward/cost of the current (step k) state-action pair, and the Q-value, or action-value function, of the subsequent state. First the Q-value for discrete-time systems with quadratic objective will be derived. From [27], it is known that the Q-value is defined as shown in Equation (3.15). The value of a state at step k is defined as $\frac{1}{2} x_k^T P x_k$, with P being the kernel matrix form the DARE [22, 27].

$$Q(x_k, u_k) = r(x_k, u_k) + V(x_{k+1}) = \frac{1}{2} x_k^T Q x_k + \frac{1}{2} u_k^T R u_k + \gamma \frac{1}{2} x_{k+1}^T P x_{k+1} \tag{3.15}$$

Using Equation (3.3), to replace x_{k+1}, the Q-value is adapted as in Equation (3.16). When re-arranging the terms, the matrix equation, namely Equation (3.17), is the result. The vector $Z_k = \begin{bmatrix} x_k & u_k \end{bmatrix}^T$, is the augmented state-action pair at time step k.

$$Q(x_k, u_k) = \frac{1}{2} x_k^T Q x_k + \frac{1}{2} u_k^T R u_k + \gamma \frac{1}{2} \left[A_d x_k + B_d u_k\right]^T P \left[A_d x_k + B_d u_k\right] \tag{3.16}$$

$$Q(x_k, u_k) = \frac{1}{2} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T \begin{bmatrix} Q + \gamma A_d^T P A_d & \gamma A_d^T P B_d \\ \gamma B_d^T P A_d & R + \gamma B_d^T P B_d \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} = \frac{1}{2} Z_k^T H Z_k \tag{3.17}$$

The matrix $H = H^T$, is the newly formed kernel matrix of the form $H = \begin{bmatrix} H_{xx} & H_{xu} \\ H_{ux} & H_{uu} \end{bmatrix}$, from which the optimal control gain, $K^* = H_{uu}^{-1} H_{ux}$ can be deduced [27]. The optimal policy is now defined in terms of augmented state, $u_k^* = -K^* X_k$. Note that with the newly defined kernel matrix, H, the Q-value is independent of knowledge about system dynamics, allowing for optimal control on systems with unknown dynamics. Indeed, the components of H are composed of partially an estimation of the system dynamics and an estimate of P. As a result the Q-learning method is sample based, as it needs to inspect the systems behaviour over time.

Once the Q-value has been determined, the Q-function is defined according to Section 2.5.2. When translated in optimal control terms, it gives Equation (3.18).

$$\begin{aligned} &Q(x_k, u_k) = r(x_k, u_k) + \gamma Q(x_{k+1}, u_{k+1}) \\ \Leftrightarrow\ &Z_k^T H Z_k = x_k^T Q x_k + u_k^T R u_k + \gamma Z_{k+1}^T H Z_{k+1} \end{aligned} \tag{3.18}$$

The above Q-function corresponds to the one used for LQR tasks. For LQT tasks, however, an adaptation has to be made in the state x_k from Equation (3.15). In the special case of the latter task, the Q-function is formulated by Equation (3.19), using the augmented state X_k and where $Z_k = [\ x_k \quad x_k^r \quad u_k\ ]^T$ includes the reference signal state, x_k^r.

$$Z_k^T H Z_k = X_k^T Q_1 X_k + u_k^T R u_k + \gamma Z_{k+1}^T H Z_{k+1} \tag{3.19}$$

Only the LQT task will be examined for the remainder of this subsection, due to its appropriate focus for the applications described in this work's introduction and consequently, for further applications in this research. The transition of the subsequent methods to LQR tasks is easily achieved, by either setting the reference state to zeros, or by adjusting the state parameter X_k to x_k. Two sample-based online implementations of Q-learning will be presented further in this subsection, namely VI and PI.

Online VI and PI

Online VI is given in algorithm 3.3.1, which shows policy evaluation in line 5 and policy improvement process in line 6. These concepts were introduced in subsections 2.3.1 and 2.3.2, respectively. The policy evaluation step of online VI consists of solving the recursive Q-function for the kernel matrix, H^{j+1}. Due to the sample-based nature of the algorithm, samples are collected over a pre-defined number of time steps. The cost, $X_k^T Q_1 X_k$, as well as Z_k and Z_{k+1} are sampled for the update which is performed using Ordinary Least Squares (OLS) regression. The OLS finds values of a vector based on a regression matrix and a regression vector.

Algorithm 3.3.1: Online sample-based Q-learning Value Iteration for LQT [22, 23]

Input : discount factor γ, small $\eta > 0$;
Output : $\forall a \in \mathcal{A}, s \in \mathcal{S}$: Optimal kernel matrix H, and optimal stabilising control gain K_1;

1 Initialise H^0 arbitrarily;
2 $K_1^{(0)} \leftarrow H_{uu}^{-1} H_{uX}$;
3 $j \leftarrow 0$;
4 **repeat**
5 Solve for H^{j+1}: $Z_k^T H^{j+1} Z_k = X_k^T Q_1 X_k + u_k^T R u_k + Z_{k+1}^T H^j Z_{k+1}$;
6 $K_1^{(j)} \leftarrow H_{uu}^{-1} H_{uX}$;
7 $j \leftarrow j + 1$;
8 **until** $||H^j - H^{j+1}|| < \eta$;

Before getting into the algebraic manipulations required to use OLS, a study has to be made with regard to the vector and matrix shapes. Assuming the system's state, $x \in \mathbb{R}^n$, the input, $u \in \mathbb{R}^m$, and the reference signal state, $x^r \in \mathbb{R}^p$, the vectors $\{Z_k, Z_{k+1}\} \in \mathbb{R}^{(n+m+p)}$. Consequently, by means of matrix multiplication dimensions, the kernel matrix, $H \in \mathbb{R}^{(n+m+p)\times(n+p+m)}$. Since $H = H^T$, the kernel matrix contains $(n+m+p)(n+m+p+1)/2$ independent values. Finally, let $i \geq (n+m+p)(n+m+p+1)/2$ be the number of samples collected.
Lastly, in order to convert the quadratic formulation in Z of $Z^T H Z$, the Kronecker product ($\otimes$) is introduced in Equation (3.20), with $\bar{H}$ being the vectorised form of the kernel matrix, H. Using the Kronecker product, the policy evaluation equation to be solved is rewritten as Equation (3.21).

$$Z^T H Z = (Z \otimes Z)^T \bar{H} \tag{3.20}$$

$$(Z_k \otimes Z_k)^T \bar{H}^{j+1} = X_k^T Q_1 X_k + u_k^T R u_k + \gamma Z_{k+1}^T H^j Z_{k+1} \tag{3.21}$$

With the knowledge of the aforementioned shapes of Z and H, the right hand side of the equation at time step k has a shape of (1×1). On the other side of the equation, the Kronecker factor, $(Z \otimes Z)^T$, will have a shape of $((n+m+p) \times (n+p+m))$.

Sampling for the OLS method will save the outcome of the right hand side of Equation (3.21), as well as the Kronecker factor on the left hand side of that same equation. The regression matrix of the OLS is defined as the stacked vectorised samples of the left hand side and the regression vector is the column vector containing all samples of the right hand side. Forcing the shapes of these regression parameters to be $(i \times (n+m+p))$ and $(i \times 1)$ respectively. However, since the kernel matrix only contains $(n+m+p)(n+m+p+1)/2$ independent values, only the elements of $(Z \otimes Z)^T$ corresponding to these values are sampled, changing the shape of the regression matrix to $(i \times (n+m+p)(n+m+p+1)/2)$. Applying OLS using these sampled regression parameters will give all the independent values of H^{j+1}, and ultimately by using correct mapping, H^{j+1}. From the latter, the policy improvement process is deduced in an algebraically straightforward manner.

Algorithm 3.3.2: Online sample-based Q-learning Policy Iteration for LQT [22]

Input : discount factor γ, small $\eta > 0$;
Output : $\forall a \in \mathcal{A}, s \in \mathcal{S}$: Optimal kernel matrix H, and optimal stabilising control gain K_1;

Initialise $K_1^{(0)}$ as a stabilising control gain;
$j \leftarrow 0$;
repeat
  Solve for H^{j+1}: $Z_k^T H^{j+1} Z_k = X_k^T Q_1 X_k + u_k^T R u_k + Z_{k+1}^T H^{j+1} Z_{k+1}$;
  $K_1^{(0)} \leftarrow H_{uu}^{-1} H_{uX}$;
  $j \leftarrow j + 1$;
until $||H^j - H^{j+1}|| < \eta$;

In a similar manner as VI, the online Q-learning PI for the LQT task is given in algorithm 3.3.2. The single difference between the two algorithms lies in the policy evaluation step, for which PI does not make use of the current estimation of the kernel matrix H^j, for the kernel estimate update. Instead the regression matrix and regression vector are defined based on the left hand side and right hand side of Equation (3.22), respectively. The same dimensional analysis as VI holds for PI.

$$\left((Z_k \otimes Z_k)^T - \gamma (Z_{k+1} \otimes Z_{k+1})^T\right) \bar{H}^{j+1} = X_k^T Q_1 X_k + u_k^T R u_k \tag{3.22}$$

Having understood the online VI and PI concepts, the condition to which both methods have to obey, is still to be described. The following subsection will explain the persistence of excitation condition.

Persistence of Excitation (PE)

For both algorithms discussed previously, the PE condition must be satisfied, to ensure converging learning behaviour. Indeed, due to the unknown dynamics assumed for the Q-learning method discussed in this section, the agent performs system identification as part of finding the optimal policy and Q-function. When performing policy evaluation and policy improvement, the system's input(s) must be persistently excited to promote the agent's exploration of the state space.

The excitation signal needs to be carefully crafted such that singularities can be averted when inverting the precision matrix in the OLS policy evaluation step of both algorithms 3.3.1 and 3.3.2. The placement of PE will be discussed for VI only, but can easily be translated to PI if desired.
According to [35], one way to describe the PE condition for continuous-time systems is to evaluate the input vector, u, over a given time interval. The conditions is laid out in Equation (3.23), stating that the integral of u over an interval $[t_0, t_T]$ must be larger than a certain positive constant $\delta > 0$, under the assumption that $u(t)$ is smooth in that interval. In essence, u can not integrate to zero over that time interval.

$$\int_{t_0}^{t_T} |u_\tau| d\tau \geq \delta \tag{3.23}$$

$$\sum_{\tau=t_0}^{t_T} |u_\tau| \geq \delta \tag{3.24}$$

The importance of the above equation is not to be underestimated when system identification is performed. Forcing the input to be non-zero over time will force the agent to explore the state-space, even when it is close to the optimal Q-function and optimal policy. Its equivalent form for discrete-time dynamic systems is formulated in Equation (3.24) [26], and the same logic applies as for continuous-time dynamic systems.

3.4. Synthesis

The research field of control, and more specifically, the increasingly automated part of the field is gaining traction in the modern world. The focus of this chapter has been set on tracking a reference signal by an autonomous controller. For this purpose, optimal control theory was used, with the use of a linear quadratic objective function. The LQT task has been proven to converge on both VI and PI processes. Thus, proving that LQT is a concept that works when combined with RL methods. This is one of the objectives outlined in the research scope in the form of research sub-question 2. The convergence is, however, highly dependent on the PE condition being fulfilled. Note that the discussion held in this chapter was focused on linear systems. Further research is required to adopt autonomous controllers able to tackle such tracking tasks for non-linear systems.

4

Curriculum Learning

The idea of Curriculum Learning (CurL) for Reinforcement Learning (RL), is the construction of a task curriculum on which the agent is trained before tackling the ultimate more complex task. Use is made of curriculum learning on tasks for which the state space is too large, or the amount of decisions presented to the agent is relatively large. For such problems, the training time of an agent would be objectively large and convergence of learning could be complicated to predict, should convergence even be possible. One example of this problem was solved in [50, 51], where the game of Go was tackled using a curriculum learning approach. To diminish complexity of the said tasks and in the hope the agent will find any sort of optimum, a sequence or multiple threads of tasks are designed through which the agent is gaining understanding of sub-tasks. As such, the focus is laid on the agent's understanding of building blocks. In [36], they manage to learn a control policy to perform non-linear helicopter maneuvers by providing a more gradual learning framework.
It should be noted that CurL does not have much impact on tasks that lack a minimum complexity. Instead, complicating the learning process, the implementation and slowing learning convergence of simple tasks. Therefore, a task must be scrutinised for its complexity in order to avoid implementing a curriculum for a task that does not require one.

The task this report is considering, is the RL agent-environment interaction on a tracking task, as is more clearly seen in Figure 1.1. The two previous chapters, chapters 2 and 3, have focused on defining the characteristics of such task and this information can be used in this chapter.

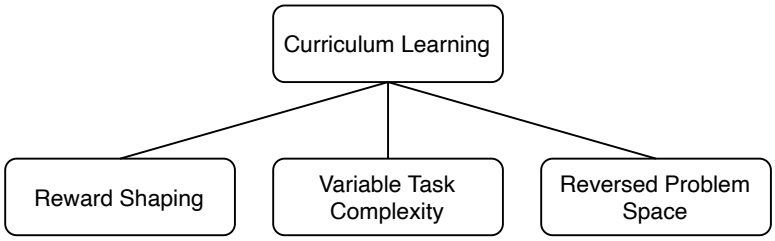


Figure 4.1: Curriculum Learning Taxonomy

According to [60], CurL is divided in three categories, namely, reward shaping, variable task complexity and reversed problem space. A taxonomy repeated in Figure 4.1 for sake of overview and clarity. The partition is based on how the curriculum followed by the agent is constructed. Thus, heavily influencing the sub-task selection.
Reward shaping relates to the theory presented in Chapter 2, where it is explained how a state transition is accompanied with a reward, based on an action selected by the agent's policy. Although in Chapter 2, the reward function, $\mathcal{R}$, is invariant throughout the task, it can be changed to bias the agent to follow a given path. A reward shaping curriculum, consists of the adaptation of the reward function between tasks, or even episodes, in such a way that the agent is guided through the state space. This method has been proven successful in [37], where a controller is trained to take cinematic shots of a person. Based on the abstract definition of curriculum learning given in Section 4.1, the task has to gain entropy as the agent progresses through the curriculum. Requiring a decreasing bias of the reward function over the course of the curriculum. The higher the bias, the more predictable the outcome will be, and consequently the least entropy the learning process will have. The main drawback of such methods is the required knowledge of said reward function in order to have the ability to *shape* $\mathcal{R}$ properly for an effective learning convergence.
Variable, usually increasing, task complexity is commonly mistaken as a synonym for curriculum learning. Even though it represents a large portion of the curricular learning approaches, it is not all-encompassing. In this category, a curriculum is constructed by finding a group of tasks with different levels of complexity, yet related to a shared target task. By the general definition of curriculum learning presented in Section 4.1, the task complexity should increase in order to have increasing entropy.

Finally, a more recent development of this field of research, is the reversed problem space. Initiating the agent close to the goal state (s_{goal}, from Figure 2.2b) and moving it further as the progress is made through the curriculum. The technique is based on how humans learn to play board games, such as chess [11]. It comes with the requirement to know what the goal state is. The latter knowledge is not always available, certainly not in case of a continuous-time infinite horizon Markov Decision Process (MDP).

All the aforementioned categories (Figure 4.1) rely on a variation in a certain aspect of an MDP throughout the curriculum. Inherently, creating a set of different MDP's, in a sense. Transitioning between mentioned MDP's requires an introduction to a specific field of research called Transfer Learning (TL), which will be presented in Section 4.2.

4.1. Abstract Concept of Curriculum Learning

A curriculum can be generally defined as a group of tasks which satisfy two conditions, given in Equation (4.2) [3]. In order to understand these conditions, a couple of concepts need to be presented.
First, let's assume that the purpose of the ultimate task for which the agent is trained, is to minimize the cost function $C_1(\theta)$. However, this cost function is complex, most likely non-convex in θ, and thus this problem is hard if not impossible to solve. Furthermore, it is assumed that a different cost function, $C_0(\theta)$, is easy to minimize. The purpose is to have gradual transition, $C_0(\theta) \rightarrow C_1(\theta)$, by using intermediate cost functions, $C_\lambda(\theta)$. Here, λ denotes the curriculum step, or how much the agent has progressed in the curriculum between C_0 and C_1.
In RL terms, the agent learning process happens on states, $s \in \mathcal{S}$, which have a certain probability of occurrence, $P(s)$, in an arbitrary task's state space. When considering a curriculum, that probability can be tampered with using a weight function, $W_\lambda(s)$, in a way where the occurrence of a given state is given less or more likelihood. Here, λ denotes the affiliation of the weight function to the cost function of curriculum step λ. The occurrence probability distribution of the state space at step λ, can consequently be defined as being proportional to the weight function, $W_\lambda(s)$, as given in Equation (4.1). Not to be confused with the Q-function used in chapters 2 and 3, the notation used in Equation (4.1), $Q_\lambda(s)$, represents the occurrence probability distribution of the state space at the curriculum step λ.

$$\forall s \in \mathcal{S} : Q_\lambda(s) \propto W_\lambda(s)P(s) \tag{4.1}$$

The last concept to introduce is the statistical entropy of a probability distribution, denoted by $H(\bullet)$, which is closely, associated with randomness of a process. In this case, entropy is interpreted as being a gauge for the predictability of state occurrences. The more entropy is present, the more diversity is expected in the state space.

Having discussed the abstract concepts contained in curriculum learning, the general definition of a curriculum can be further elaborated. More specifically, the conditions which a set of tasks has to satisfy to be considered a curriculum shown in Equation (4.2) [3].

$$\begin{aligned} &1) \quad H(Q_\lambda(s)) < H(Q_{\lambda+\epsilon}(s)) \quad \forall \epsilon > 0, \forall s \in \mathcal{S} \\ &2) \quad W_\lambda(s) \leq W_{\lambda+\epsilon}(s) \qquad\quad \forall \epsilon > 0, \forall s \in \mathcal{S} \end{aligned} \tag{4.2}$$

The first condition specifies that the entropy increases over the course of the curriculum. That is for a curricular step further than λ, namely $\lambda + \epsilon$ (with $\epsilon > 0$). In essence, the purpose of curriculum learning is to transition towards $C_1(\theta)$, the cost function that is initially hard if not impossible to optimise. As such, by initially limiting the diversity in the state space, the cost function's domain is smaller. However, the target is still to have a cost function enveloping the entire state space, requiring the diversity, and consequently the entropy to increase as the curriculum progresses.
The second condition is closely related to the first, in that the probabilistic weight, $W_\lambda(s)$, is responsible for the diversity in the state space distribution. Indeed, $W_\lambda(s)$ is multiplied with the probability distribution of the final state space, $P(s)$, in Equation (4.1). Therefore, if the diversity is to be increased, so is $W_\lambda(s)$.

The step size taken between two curricular steps, ϵ, has to be selected such that the agent would not get stuck in a local optimum when transitioning to the subsequent step after that.

4.2. Transfer Learning

Transfer learning is used to map knowledge gained by the agent in a certain curricular step to another. Due to the variability in curriculum structures, TL is a vast field of research, mostly exceeding the scope of this report. However, an introduction is provided in this section. The main focus will lie on the representation of the state space and the action space amongst curricular steps, and how these discrepancies can be tackled.

Before diving into the combinatory variations of state and action spaces, the basis of transfer learning needs to be covered. The ultimate goal is to transfer as much useful knowledge as possible between curricular steps. One should note that a curricular step can be composed of multiple tasks, in which case the knowledge of multiple tasks has to be mapped to the subsequent curricular step. In TL jargon, the tasks composing the initial curricular step are called *source tasks* and the tasks in the subsequent curricular step are called *target tasks*. The transfer of knowledge between mentioned tasks requires task mappings when state and/or action spaces are not represented equally. As the name suggests, task mapping is the translation of the state and/or action space from the source task to the target task. When the representations are equal, no task mapping is required. A multitude of facets in TL can be discussed, ranging from source and target task selection, to diverse task mapping strategies. In this report, the structure presented in [54] will be partially summarised as it gives an effective overview of the field of TL. Only discrepancies between action and state space representations will be discussed in this report. However, more Markovian parameters can vary, such as transition function $\mathcal{T}$ or the reward function $\mathcal{R}$. These are discussed in more detail in [54] with as purpose to give an overview of the field, and in more detail in for example [42] and [1], respectively.
Transfer learning task mappings will be segmented based on state and action space representations between source and target tasks denoted as, $\{\mathcal{S}_{source}, \mathcal{A}_{source}\}$ and $\{\mathcal{S}_{target}, \mathcal{A}_{target}\}$, respectively. Essentially, four possible combinations exist as the basis of the chosen segmentation, given in Table 4.1. However, these can be reduced to two more general cases. Either the space representation is constant throughout the curriculum, or one of the two or even both are not fixed.

Table 4.1: Possible combinations of action and state space representation discrepancies between two curricular steps

$\mathcal{A}$	$\mathcal{S}$
$\mathcal{A}_{source} = \mathcal{A}_{target}$	$\mathcal{S}_{source} = \mathcal{S}_{target}$
$\mathcal{A}_{source} = \mathcal{A}_{target}$	$\mathcal{S}_{source} \neq \mathcal{S}_{target}$
$\mathcal{A}_{source} \neq \mathcal{A}_{target}$	$\mathcal{S}_{source} = \mathcal{S}_{target}$
$\mathcal{A}_{source} \neq \mathcal{A}_{target}$	$\mathcal{S}_{source} \neq \mathcal{S}_{target}$

The remainder of this section will cover further explanations about the four aforementioned cases. Additionally an overview of metrics to measure transfer performance will be given.

4.2.1. Constant Action and State Space Representation

When the action and state space representation by the agent is fixed along the curriculum, the options for transferring knowledge are objectively easier due to limitation in methods that can be used to do so. First, the meaning of fixed space representation has to be clarified. In terms of control theory, a state s is a vector constructed by parameters defining the system in an environment for a certain task. These parameters are discretely identifiable, in a system where knowledge of system dynamics are understood. Correspondingly, the action space is defined, not by system dynamics parameters, but by actions. Each action available to the agent is represented in the action state [1].
For fixed action and state spaces in a curriculum, all components of the respective vectors are considered throughout. Therefore, knowledge is transferred in an optimal manner using an intelligently defined mapping of the agent's understanding of the task between curricular steps.

A common occurrence of the cases presented in this subsection is in the reversed problem space curriculum (see Figure 4.1), where an agent is gradually removed from the goal state. The curriculum adds entropy by elongating the path to the goal state over curricular steps, whilst giving the agent access to most if not all the action and state space parameters from the start. Allowing the agent to put more focus on the understanding of these respective spaces in the initial curricular steps and optimising trajectories when it is initiated further from the goal.

4.2.2. Different Action and/or State Space Representation

Contrary to Section 4.2.1, this subsection will discuss the more complicated case that is a changing internal representation of the action and state spaces by the agent. With the assumption that at least one of the spaces is represented differently, a mapping is required to transfer the knowledge between two subsequent curricular steps. When the former assumption is valid, the tasks are considered separate, resulting in what is known as *inter-task* mapping.

[1] In terms of control theory, the action is defined by the vector u.

The purpose of mapping is in essence to translate the knowledge from one representation to another. The mapping logic is generally designed by a human operator [53–55, 57]. Two trends are introduced by the related works presented in [53] and [55], both assuming the possibility of considerable change of $\mathcal{A}$ and $\mathcal{S}$, between source and target tasks. One elaborates on learning the value function, v, which is transferred between tasks. Consequently, the policy has to be re-iterated in the target task. As long as a sensible mapping is provided, the method shows advantageous results. The other paper researched policy learning, where the policy is transferred from a source to a target task. In the latter, it is the value function that requires iteration, whilst the policy is fine-tuned. With policy learning, [55] proved that knowledge is transferable between domains as well.

4.2.3. Transfer Learning Metrics

In order to measure the efficiency of the knowledge transfer, multiple metrics can be used. In [54], these metrics are used to evaluate and compare methods. The metrics are summarised in Table 4.2, and briefly covered in this subsection.

Table 4.2: Metrics to measure the effectiveness of knowledge transfer between source and target task.

Nr.	Metric	Unit
1.	Jumpstart	[-]
2.	Asymptotic Performance	[-]
3.	Total Reward	[-]
4.	Transfer Ratio	[-]
5.	Time to Threshold	[s]

The first metric relates to the advantage gained by the agent in the target task from the knowledge acquired in the source task. Jumpstart is hard to define generally, due to its dependency on the task at hand. However, the collective reward between the start of a target task and a given time, t, in that task can be improved based on knowledge transferred from a source task.
Asymptotic Performance measures the improvement of the performance between source and target task after convergence is appearing. In general the agent's performance will tend to an asymptote, called the asymptotic performance. With transfer learning, the purpose is to improve that performance.
The third metric is accurately named as it compares the total reward or cost collected by the agent when it performs the task. To be more precise, it is the cumulative reward gathered between the initial state and the terminal state (goal state or not). The comparison is made between source and target task.
Transfer ratio compares the total reward of an agent on which TL has been applied with one that does not follow a curriculum. The idea is that a curriculum, might not be required in the case of tasks that are not complex enough. Up to this point, no time constraint has been considered for the metrics. Leading to the last metric in Table 4.2, time to threshold.
The last metric measures the time required for an agent to achieve a certain performance level. Comparing the source and target task, again, gives insight to the transfer efficiency. This metric is of great use when considering tasks that require rapid optimisation by the agent. Of importance in stabilisation problems, for example.

4.3. Synthesis

Inspired by the learning methods humans use, during school curricula and game play, the field of Curriculum Learning (CurL) aims at facilitating learning of complex tasks. Divided in three main ways of doing so, the field is, however, prone to a more standardised definition. Based on two criteria, a curriculum can be designed. First, each curricular step has to increase entropy in the system, by adding diversity in the state space. In doing so, it has to increase the probabilistic occurrence of states, by adapting their probabilistic weight. Which is the second condition. These form the key aspects defining a curriculum, answering research question 3.1.
In order to pass on knowledge between curricular steps, Transfer Learning (TL) is used. Two divisions were found based on variability in state and action representation throughout the curriculum. When variation in representation is found, a mapping is required to transfer knowledge between steps. The subject of TL is concluded with a brief discussion on how to assess the success of knowledge transfer. Which is of great importance if one would like to verify whether the use of a curriculum is actually required in the topical task.
The design of an effective curriculum requires attention to the design of tasks as was outlined in the intro of this chapter. These strategies allow for a variety of possible curricula. Moreover, the importance of TL is not to be undermined as it is an important tool for transferring knowledge between curricular steps. Consolidating the research scope set in Section 1.2, a curriculum can indeed be designed for a RL task.

5

Safe Learning

As discussed in previous chapters, the potential for intelligent controllers through Reinforcement Learning (RL) methods is tremendous. One major drawback of these methods is the exploration of the state-space, which at times is randomised to increase the probability of finding a more optimal policy, is the relatively high probability to end in a terminal state in the early learning stages. In certain applications such as robot training, the State Space (SS) contains states that are resulting in damage to the agent or even absolute failure (fatal terminal states, definition 2.1). An example of such a situation would be a car driving task, where the agent crashes the car. The state at which the car is crashed, is said to be fatal.
Safe learning algorithms are added to the RL interaction discussed in previous chapters in a modular fashion. They are considered applicable in a curriculum learning setup as long as their presence conforms with the rules for curricular design presented in Section 4.1. In this chapter insight is provided towards the field of safe learning. More specifically, mapping the current trends in Section 5.1, as well as how safety is defined in literature in Section 5.2. Finally, in alignment with the topic of this report, safe learning algorithms that rely on a form of system dynamics are sought. These form the subject of analyses in Part I and Part III of the report as well as in future work.

5.1. An Overview of Existing Methods

Research in Safe Reinforcement Learning (SRL) started to get attention in the early 2000's, when implementation of an intelligent controller using RL was tried on remote controlled helicopters. Since then, the field has grown into a large web of inter-related theories. As readily outlined in [12], the field is considered to be split into two approaches. The first of which is making the learning process safe by shaping the *Optimisation Criterion*. A second approach is to adapt the way the *Exploration Process* is performed. An overview of what both approaches further incorporate is given in Figure 5.1.
The *Optimisation Criterion* branch seen in Figure 5.1 groups a variety of methods. Their common characteristic is their modification of the optimisation criterion used in the Markov Decision Process (MDP). As was presented in Section 2.2, the expected return is used commonly as optimisation criterion for MDP's. The methods proposed in the aforementioned branch of Figure 5.1, aim to diverge from the expected return as an optimisation criterion and specify numerous approaches. The case is made to constrain the criterion using Chernoff bounds by [34] or making the optimisation criterion sensitive to risk, such as presented in [14, 46]. Although the methodology presented in the respective papers regarding the mentioned approaches is promising, the scope of this thesis is restricted to the simplicity of expected return. The reason for scrapping the optimisation criterion is two-fold. First, variation in the optimisation criterion will be considered as a variable in Curriculum Learning (CurL). Thus, adhering to a fixed optimisation criterion during the curricular steps and only allowing change between said steps. Second, the main motivation behind this thesis is to find a safe exploration method for systems with unknown dynamics (as defined by definition 1.8). Consequently, the passionate reader would be invited to consult [12] for further details on safety through adaptation of the optimisation criterion.

On the other hand, the *exploration process* could be used to ensure safe learning. The exploration process is defined according to definition 1.6. In most implementations where exploration is adjusted to incorporate safety, external knowledge is required to determine the actions' safety. A group of methods do so by providing this knowledge through initialisation values of certain parameters within the agent. Other methods require the presence of a teacher, where the relation between the agent and the teacher can take multiple forms [15, 41, 56]. The first of which is the agent understanding its limitations at a certain state and requesting advise from the teacher. Second, the teacher determines whether the agent should use advice, based on the state it is in and its current experience. Finally, the agent can learn from demonstration from either a human teacher or an existing controller that is sub-optimal and consequently requires improvement.
There are, however, approaches that adopt a risk-directed exploration. These accord a risk level δ to a given state-action pair. One such method, namely the Safety Handling Exploration with Risk Perception Algorithm (SHERPA) [30], will be discussed in more detail in Section 5.3.3.

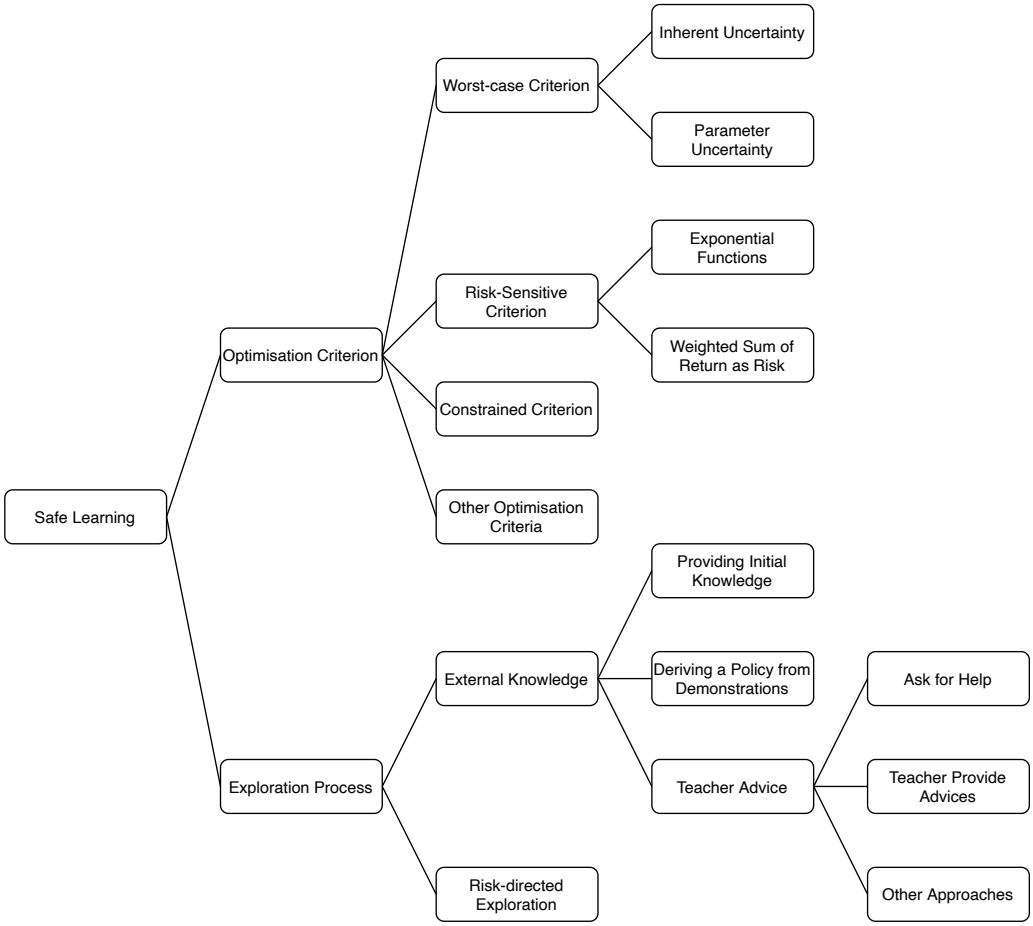


Figure 5.1: Outline of the field of safe reinforcement learning (interpretation of Table 1 from [12])

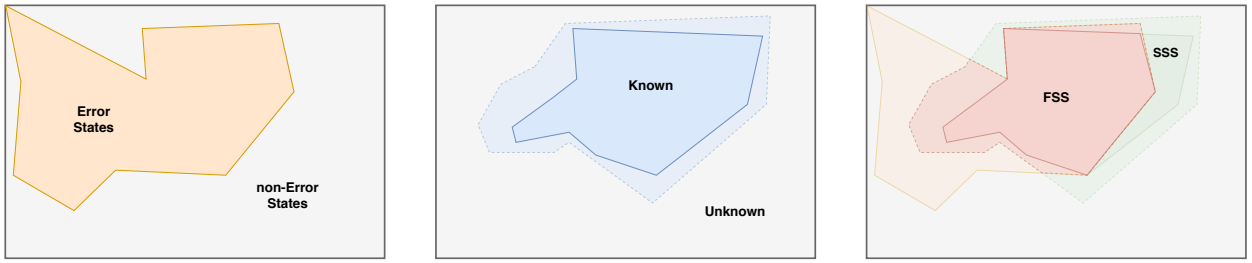
The concepts introduced in safe learning are highly dependent on the understanding of given terminology to states with specific characteristics. The division of the state space in the multiple types of states and sub-sets is first given in Section 5.2. Followed by specifying safe learning algorithms that are inherently dependent on external knowledge in Section 5.3. This section focuses on the concept of a restricted policy space outlined in Section 5.3.2, closely linked to the concepts introduced in Chapter 2. Moreover, the general method of policy reuse is presented in Section 5.3.2 which ultimately leads to SHERPA in Section 5.3.3, a specific case of policy reuse based on risk-directed exploration.

5.2. State Space Segregation

The state-action space of any task with the assumption of a terminal state can be split into two parts. First there is the collection of undesired states, also denoted in literature as *error states* [14], *fatal states* [30] or *unknown states* [13]. Complemented by their respective counterparts, *non-error states* [14], *safe states* [30] or *known states* [13]. All rely on the same concept: a segregation of the state space in regions.
This section will elaborate on these concordant, yet different concepts. First introducing (non-)error states in Section 5.2.1, followed by (un)known states in Section 5.2.2. Finally, the concept of fatal and safe state spaces will be outlined in more detail in Section 5.2.3.

5.2.1. Error and Non-Error states

The conception of (non-) error states was introduced in [14], dividing the state space of a MDP in terminal states and non-terminal states (see definition 2.1). Error states are highly undesirable, due to the cost associated with destruction of learning systems in practical applications such as robot-arm control or control of an Unmanned Aerial Vehicle (UAV). In order to clarify the concepts unambiguously, definitions 5.1 and 5.2 define the concepts of *error state* and *non-error state* respectively.



(a) *Error* and *Non-Error* states within the set of states $\mathcal{S}$

(b) *Known* and *Unknown* states within the set of states $\mathcal{S}$. Dashed line represents the exploration of $\mathcal{S}$ over time.

(c) *Safe State Space (SSS)* and *Fatal State Space (FSS)* within the set of states $\mathcal{S}$. Dashed line represents the exploration of $\mathcal{S}$ by the agent over time. Note that this figure is an overlap of figures 5.2a and 5.2b

Figure 5.2: State space segregation based on different concepts: (left) error-states, (middle) known states, (right) safe states. The grey rectangle represents the complete state space of the task at hand.

Definition 5.1: Error State

A state $s \in \mathcal{S}$ is defined as an *error state* if it is a terminal state in which the agent or external entities sustain damage, injury or destruction. An error state is said to be part of a set of all error states $\Phi \subseteq \mathcal{S}$.

Definition 5.2: Non-Error State

Contrary to error states, *non-error states* can be terminal, as long as no harm is sustained by the agent or external entities. These states are said to be part of the non-error state set $\Gamma \subseteq \mathcal{S}$.

A visualisation of this categorisation is given in Figure 5.2a. As depicted there, the complete state space is split in two time-independent areas, namely error (Φ) and non-error states (Γ). These subsets are characterised by $\Phi \cap \Gamma = \emptyset$ and $\Phi \cup \Gamma = \mathcal{S}$. As a result, a state s can not be part of both sets.
The Time-independence of the categorisation emphasises the fact that an error state will be an error state independently of the time the learner spends in the environment, or whether a state has been visited or not.
The idea presented in [14] is to associate risk with a state s_0 and its subsequent series of states $(s_i)_{i>0}$. The paper introduces a risk function $\rho^{\pi}(s)$ which denotes the probability for a given state, s, and its consequent sequence of states to end in an error state $s' \in \Phi$.

5.2.2. Unknown and Known states

The concept of known and unknown states relies on the experience of an agent in the specific state space of its selected task [13]. An agent can be interpreted as an entity visiting a state s and taking in an action to transition to another state s', for which it will get a reward.

Definition 5.3: Known State

An state $s \in \mathcal{S}$ is considered part of the state set $\Omega \subseteq \mathcal{S}$ of known states if it has been visited by the agent.

Definition 5.4: Unknown State

An state $s \in \mathcal{S}$ is considered part of the state set $\Upsilon \subseteq \mathcal{S}$ of unknown states if it has not yet been visited by the agent.

Initially the complete state space is unknown ($\Upsilon = \mathcal{S}$) since no exploration has been performed yet. When exploration starts, the agent is spawned at an initial state s_0, which is assumed to be a non-error state (see Section 5.2.1), and will transition to other states s'. According to definition 5.3, this means that s_0 is then part of the *known* states within the subset Ω. Additionally, a case base B is constructed, containing not only the known states, but also the action taken at that state and the state-action pair's value. Over time, the complete state space can be explored and learned. This method does not consider error states, but associates risk with the fact that a state is known or not. Definition 5.3 can be further elaborated by suggesting that in case the agent is equipped with sensors, the sensors' range forms a perimeter around the visited state. The states contained in the perimeter can be assumed to be visited and consequently, known.

Definition 5.5: Case Base B

The case base B is defined as a collection of tuples containing a state-action pair and the pair's value.

$$B = \{c_0, c_1, ..., c_n\} \quad with \quad c_k =< s_k, a_k, V(s_k) >$$

All states s_k within the tuple c_k of definition 5.5 are considered to be *known* and thus $s_k \in \Omega$. The method proposed in [13] describes the use of cases c that are in B as much as possible. Exploring unknown states with random probability. When an unknown state s' is encountered, the case base B is checked for a case c_i for which the state s_i is close enough (using Euclidean distance, see Equation (2.7)). In case the distance is below a given threshold θ, the action a_i of the matching case taken and the newly encountered state is stored in B. Meaning that subset Ω grows with time, as is shown in Figure 5.2b. Over time, the state space is explored and the risk map of the state space is perfected, giving a more realistic overview of dangerous parts in the space.

5.2.3. Fatal and Safe states

The concept of Fatal and Safe state spaces is a combination of subsections 5.2.1 and 5.2.2. Fatal states are simply error states and are said to be part of Fatal State Space (FSS) (see definition 5.6). The FSS is initially unknown to the agent which over time, discovers the boundaries of the FSS. Contrarily the Safe State Space (SSS) is defined according to definition 5.7. Although these definitions differ slightly from the ones presented in [30, 31], they replace the additional definitions of that work introducing a further segregation of the state space into a safe (R_{safe}), uncertain (R_{unc}) and fatal (R_{fat}) part of the space.

Definition 5.6: Fatal State Space (FSS)

A state $s \in \mathcal{S}$ is considered part of the Fatal State Space (FSS) when it is considered an error state.

$$\forall s \in \mathcal{S} : s \in FSS \iff (s \in \Phi)$$

Definition 5.7: Safe State Space (SSS)

A state $s \in \mathcal{S}$ is considered part of the Safe State Space (SSS) when it is known and considered a non-error state.

$$\forall s \in \mathcal{S} : s \in SSS \iff (s \in \Omega) \wedge (s \in \Gamma)$$

In Figure 5.2c, the combination of both concepts, presented in Subsections 5.2.1 and 5.2.2, is conceptualised. Due to the state being known, both SSS and FSS are growing over time as the exploration progresses along with the learning. In the limit of time, Equation (5.1)

$$\lim_{t \to \infty} (SSS \cup FSS) = \mathcal{S} \tag{5.1}$$

Finally, in the scope of SSS and FSS, there are lead-to-fatal states that appear in the safe state space, using the current definitions 5.6 and 5.7. Lead-to-fatal states are inherently non-error states, and will be considered known in this section. Whatever the action taken at that state, it will force the agent into the FSS over time. For example, let's take a quadcopter that is flying at low altitude and has saturated control actions. If a downward motion is initiated by the controller, the craft will start moving towards the ground. At some point in time, the agent controlling the quadcopter realises that the craft is too low and commands an action to move upward to counter the low altitude. However, the action that is necessary is above the action saturation limit. This will result in an action that is too weak to counter the downward velocity. The inevitable outcome will be the quadcopter crashing. Clearly the state when the upward action was commanded is inherently a non-error state. However, due to the low altitude and the action saturation, the state has lead to an error state. The example provides a better understanding of the problematic lead-to-fatal states, which at first glance seem to be safe, but ultimately, are not. It is said that a lead-to-fatal state is part of the Lead-to-Fatal State Space (LFSS). Furthermore, the LFSS contains the FSS completely, $FSS \subseteq LFSS$.

The regions defined by SSS and FSS, assume safety is imposed on all elements of a state. It is, however, possible, that not all elements of a state are constrained by such safety bounds. In that case a last concept is introduced. The Restricted State Space (RSS), which restricts the state to the elements that are bound to safety constraints. In this RSS, all regions discussed previously exist. On the contrary, outside the RSS, no SSS and FSS are defined.

This section has lead to a definition of safety in terms of state spaces. Dividing the SS into two categories, namely safe states and fatal states. The division is based on the knowledge of a state and the error/non-error label of that state.

5.3. Safe Learning Founded in External Knowledge

In this section, the focus is set on safe learning trends and methods that have been found to rely on external knowledge as shown in Figure 5.1. More specifically, ones that are part of the exploration branch. Two principal approaches are discussed in this section, as they form the basis for SHERPA. First, the guarantee of safety by restricting the policy space is presented in Section 5.3.1. Another concept reliant on initial knowledge is called policy reuse and is outlined in Section 5.3.2. At last, SHERPA is presented, which uses principles found in the former two subsections.

5.3.1. Restricted Policy Space

Safe learning can be achieved by reducing the policy space. In the same line of thought as previous section, the abstract concept of a policy space can be imagined. Any policy, π, found by the agent (as for example in Section 2.5), is part of the policy space, Π, albeit safe or not. A way to make learning *safe*, is the reduction of the policy space to only contain policies that have a low probability of guiding the agent towards terminal states or lead-to-fatal states ($s \in LFSS$).

Restricting policy spaces to a safe policy space, Π_{safe}, and fatal policy space, Π_{fatal}, allows for reducing the optimisation space of the agent whilst ensuring a safe learning process. Indeed, since the safe policies only form a subset of the complete policy space, $\Pi_{safe} \subseteq \Pi$, the amount of policies an agent has to go through before finding the optimal or near-optimal policy is lower.
One major issue with restricting policy spaces, is that it generally relies on external, a-priori knowledge of the task at hand. The purpose would be to avoid this liability through online learning. Meaning that Π_{safe} is completed over time, though starting as an empty set. However, the latter process would mean no safety guidelines for the agent in the first few steps of policy improvement.

One criterion for classifying a policy, π, as safe, $\pi \in \Pi_{safe}$, is ensuring ergodicity [33]. Ergodicity, in the scope of safe learning, means that from a given state, it is possible to go back to any known state, $s \in \Omega$. Although in discrete state spaces this is no issue, in continuous ones a closeness condition is required. The latter condition checks the euclidean distance (Equation (2.7)), between two states, and verifies that the distance is lower than a given threshold. If that condition is satisfied, the two states are said to be close enough to be considered equal.

5.3.2. Policy Reuse

Suggested by the name, policy reuse describes the selection process of policies from the past compared to new policies. The algorithm is dependent on initial knowledge in the form of an initial policy. In its most abstract form, policy reuse is a way to combine two policies using a probabilistic weighting for choosing which policy to use. The mathematical formulation of policy reuse is given in Equation (5.2) [10, 13]. In this equation the two policies are given by Π_{past} and $\Pi_{present}$, denoting a previously found policy, sometimes called a *backup* policy, and the policy currently being optimised, respectively. The probability that one or the other will be selected is defined by ψ. In risk-directed methods, this probability is set to the risk associated with the state, s.

$$a = \begin{cases} \pi_{past}(s) & prob. = \psi \\ \epsilon - greedy\left(\pi_{present}(s)\right) & prob. = 1 - \psi \end{cases} \tag{5.2}$$

In Equation (5.2) the action taken is either selected by a backup policy or by a current policy on which an ϵ-greedy method is applied (see Section 2.5). The former is exploiting acquired knowledge during a past policy optimisation process, whereas the latter promotes exploration of new actions and even random actions with probability $(1 - \psi)\epsilon$. Policy reuse inherently axes on the known and unknown division of the state space (Figure 5.2b), as it is assumed that the past policy has been constructed and optimised based on visited states. When the probability ψ is related to risk, the additional concept of error and non-error states (Figure 5.2a) comes into play, since risk is associated to the prevention of error-state visitations.

One algorithm applying policy reuse is SHERPA, which determines ψ based in a binary fashion. The value is either 1 or 0 dependent on the risk associated with the propagated bounded state. More information regarding the specifics of this algorithm are found in Section 5.3.3. Although more specific algorithms exist, such as the Policy Reuse Safe Reinforcement Learning (PR-SRL) method introduced in [13] with a continuous risk function, these will not be covered in this report in order to limit its scope.

5.3.3. SHERPA

Introduced in [30–32], the Safety Handling Exploration with Risk Perception Algorithm (SHERPA) combines the concepts of state space segregation and risk-directed policy search. SHERPA acts as a safety filter that aims to find an action from a certain state that ensures ergodicity whilst not going through the fatal state space. The flowchart of the algorithm is given in Figure 5.3.

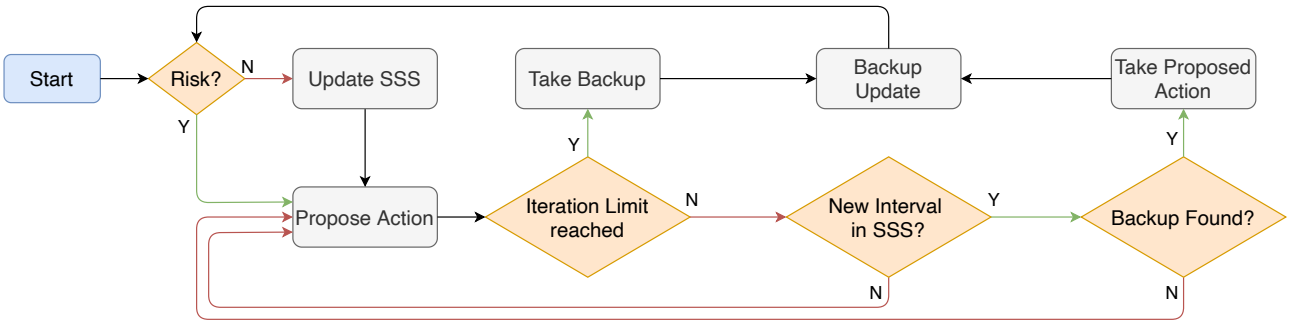


Figure 5.3: Flowchart of SHERPA algorithm; Blue is the start policy; Orange are yes/no question blocks. Chart taken from [31]

.

The SHERPA algorithm is initiated at an initial state, x_0, for which a risk assessment is assumed to be zero, $w(x_0) = 0$. Based on the state and an initial policy, an action is proposed and is then executed internally in SHERPA. The agent does not move from a state to another, but the state propagation is simulated within SHERPA using an uncertain model of the system dynamics. This simulated state propagation is done for a limited number of iterations, within which the states that are encountered have to be part of the SSS (see definition 5.7). If all states encountered are part of SSS and one of them is close enough to a previously visited safe state ($s \in SSS$), then the policy is considered a valid backup policy.

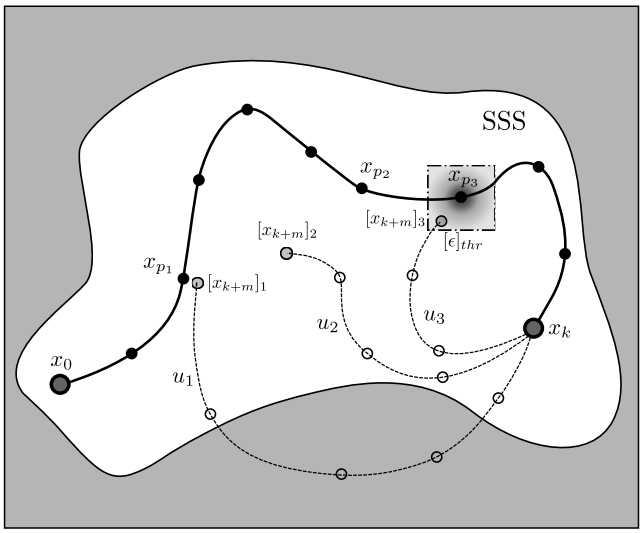


Figure 5.4: Example of SHERPA algorithm working (image taken from [32]). Solid line is the path covered by the agent; Dotted lines are the possible trajectories from step x_k linked to different policies; Grey region around x_{p_3} denotes the closeness condition.

During the state space exploration process, SHERPA constructs a safe policy space, filled with backup policies [30, 32]. These policies are used in case no appropriate action can be found at a certain state, using Equation (5.2) with $\psi = 1$.

An example of the search for a good policy from a given state x_k, is shown in Figure 5.4, where 3 different policies are being considered: $\{\pi_1, \pi_2, \pi_3\}$, with their respective input strings $\{u_1, u_2, u_3\}$. From that figure, it can be seen that $\pi_1 \rightarrow u_1$ leads to a trajectory through the non-SSS region, thus not satisfying one of the conditions for a valid policy. Furthermore, $\pi_2 \rightarrow u_2$, stays in the SSS, but does not satisfy the ergodicity condition. Finally, the last policy shown in Figure 5.4, stays in the SSS and complies with the ergodicity condition in continuous state spaces, as the last propagated state, $[x_{k+m}]_3$, is withing an euclidean distance bound, ϵ, from a previously visited state. Should none of the policies in the figure have complied with the ergodicity and SSS conditions, then a policy from the backup policy space is taken.

The backup policy space is closely related to the safe policy space, Π_{safe}, introduced in Section 5.3.1. Indeed, policies that have complied with aforementioned conditions are saved in this backup space for later use. The aim is to have this backup space filled with all possible safe policies as exploration time tends to infinity.

As briefly mentioned, SHERPA makes use of a bounding model or uncertain model to internally propagate the state when searching for safe policies. Uncertain state propagation requires a different set of algebraic rules. These have been outlined in more detail for the case of a state space model in Appendix A. The reader is invited to read this chapter if interested in the details regarding uncertain state propagation. The uncertain model used within SHERPA is provided by system identification methods outlined in Chapter 6, where three model constructs are presented.

5.4. Synthesis

Reinforcement Learning (RL) methods have a tendency to lead the agent in fatal terminal states relatively frequently in the early stages of learning. In order to avoid damaging an agent in real-life experiments, the field of safe learning has found theoretical approaches to reduce this frequency. In this chapter, a map of the safe learning field was uncovered, in which the trends could be identified. The focus was set on the safe exploration aspect of safe learning. Particularly, the methods that have a dependence on external knowledge. In order to grasp the principles of the methods found in this mapping process, a definition of safety was sought. In Section 5.2, safety is formulated in terms of state space division. On two separate levels, the states are labeled to be either error or non-error states on one level, and labeled known or unknown on another level. When combining these levels, the safety of a state can be expressed. A state is considered safe when it is known and is a non-error state.
In a subsequent section, Section 5.3, three safe learning methods are presented that are relying on external knowledge. Restriction of the policy space provides safety guarantees by limiting the policies an RL agent can use. Separately, the concept of reusing policies is presented in Section 5.3.2. It elaborates on a probabilistic selection between policies that have been used in the past and new ones entirely. Finally, combining these methods is SHERPA, which relies on the guarantee of a backup policy at all times to avoid the FSS.

The chapter provides insights with respect to research question 4 and its respective sub-questions. Giving the reader a better grasp of safe learning and what it entails, as well as the trends in the field. Furthermore, the algorithms fitting the scope of this research are the ones with foundations in external knowledge. More so in the form of system dynamics, as is the case for SHERPA. The final piece of the puzzle to complete a safe curriculum learning algorithm that does not rely on an externally supplied model of the system dynamics is presented in the next chapter.

Further research should be performed on the theory and implementation possibilities related algorithm, optiSHERPA, developed by the same authors.

6

System Identification

One approach to tackle systems with unknown dynamics is to have an internal model of these dynamics be approximated and refined over time. The field of system identification, covers a range of methods to generate estimates of such a model. Within the scope of this thesis, it used to supply the safety algorithms with an approximate model of the system dynamics. System Identification fundamentals are explained in Section 6.1, followed by a presentation of the Kalman filter for state estimation. The latter has been proven useful when the state is not accurately measured, and consequently needs to be estimated or reconstructed instead. The chapter is completed with three model constructs in Section 6.3 with an explanation on their advantages and drawbacks for systems with fixed characteristics, such as their linearity and complexity.

6.1. Fundamentals

In system identification, a model is used to approximating the system dynamics. A model is defined as in definition 6.1. The definition states that the model has as purpose to mathematically represent the system dynamics. A more general picture is given of the system identification process in Figure 6.1.

Definition 6.1: Model [8]

A model is a mathematical abstraction of the system that aims to capture its input-output behaviour while at the same time simplifying and conceptualising its inner workings.

Models are separated in three classes, namely white-box, gray-box and black-box models [8]. The classification is made on the basis of available knowledge about the internal construction of the model. Starting with white-box, the inherent construction of the model is fully known. It is thus possible for a human operator to extract knowledge contained in the model for systems with similar dynamics. White-box models are more complex to construct, as full knowledge of the system behaviour and dynamics is required. Furthermore, analytical or numerical solutions to underlying principles of such systems can be hard to find.
On the other side of the spectrum, black-box models do not require any prior knowledge. Consequently, they are relatively easy to construct. Implementation is, however, dubious due to the lack of understanding regarding the system's behaviour. Indeed, it is hard to verify and validate the model, or even fine-tune the its hyperparameters. A good example of a black-box model is an Artificial Neural Network (ANN), for which the internal logic is generally too complex to be understood by humans and hard to visually represent. More details on the inherent logic of ANN's are given in Section 6.3.3.

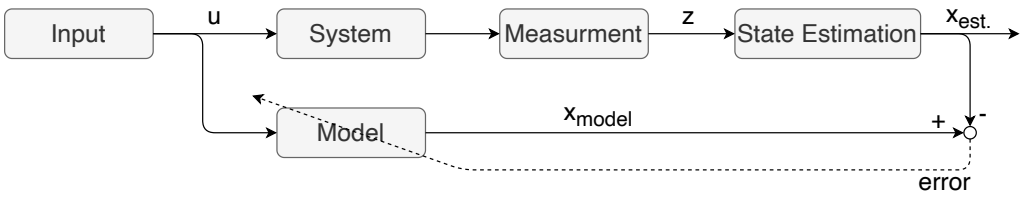


Figure 6.1: Generic flowchart of system identification

Finally, there is a great variety of methods that span between white and black box, called gray-box models. Gray-box models require partial knowledge of the system dynamics, but allow for unknowns in this understanding. They combine advantages of both white and black-box models, by keeping the utility of white-box prediction, whilst allowing a lack of understanding in system dynamics.

For model construction, two methods will be covered, namely the multivariate spline and the ANN, both able to model linear and non-linear systems. Respective models will be covered in more detail in sections 6.3.2 and 6.3.3.

In order to create an accurate model, the error between it and the the actual system dynamics serves the purpose of tuning its internal characteristic parameters. The precision of the aforementioned error is dependent on the accuracy of the system's output. In simulation environments the system output can be generated in a deterministic manner. However, in real-life experiments, these outputs are generally containing measurement inaccuracies and other unaccounted noise. In order to minimize the effect of these unknowns, the output is estimated before computing the error. For state estimation, the most prominent method is the Kalman filter. Multiple extensions exist to the Kalman filter, being designed for linear systems, to accommodate state estimations for non-linear systems [18], these will not be further developed in this literature study. Besides the Kalman filter, simpler systems' outputs can be estimated using the least squares method [20]. Although other state estimators exist, they would overstep the scope of this document and are consequently not outlined here.

6.2. Kalman Filter

Renowned for its ability to converge to correct state estimates based on stochastic measurements in a relatively small time frame, the Kalman filter is an iterative estimation method used when a received state is uncertain to be the true state due to measurement errors. A Kalman filter is an estimator that considers both the uncertainty of measurement results as well as acknowledging that the estimate by the filter is prone to inaccuracies. The purpose is to mitigate them both over time using the Kalman gain, K. Shifting importance of the measurement error uncertainty to the estimate uncertainty. The iteration process occurring within the linear Kalman filter is shown in Figure 6.2, where the five steps are shown in order.

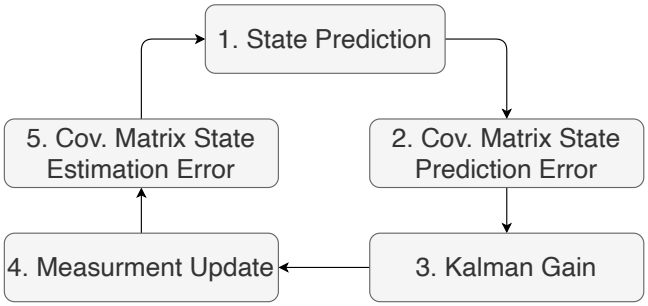


Figure 6.2: Flowchart of a linear Kalman filter's inherent process [40]

Assumption is made that the Kalman filter will start with an approximate idea of the measurement error variance and the prediction error variance as well as their respective inherent state inter-parameter error covariances. In essence, an initial guess of the covariance matrices found in step 2 and 5 are required. These do not have to be necessarily accurate, however.

Mathematically, the steps shown in Figure 6.2, are defined as given by Equation (6.1). This equation assumes a discrete-time time-invariant state space system and contains a number of variables that require context and explanation. The notation used in control theory is followed in this chapter, however a switch to Reinforcement Learning (RL) is simple as given in Chapter 8. Additionally, note the discrete nature of the system required for this filter.

$$
\begin{aligned}
&1.\ \hat{x}_{(k+1,k)} = A_d \hat{x}_{(k,k)} + B_d u_k \\
&2.\ P_{(k+1,k)} = A_d P_{(k,k)} A_d^T + \Gamma_{(k+1,k)} Q_{d,k} \Gamma_{(k+1,k)}^T \\
&3.\ K_{k+1} = P_{(k+1,k)} H^T \left(H_{k+1} P_{(k+1,k)} H_{k+1}^T + R_{k+1} \right)^{-1} \\
&4.\ \hat{x}_{(k+1,k+1)} = \hat{x}_{(k+1,k)} + K_{k+1} \left(z_{k+1} - H_{k+1} \hat{x}_{(k+1,k)} \right) \\
&5.\ P_{(k+1,k+1)} = \left(I - K_{k+1} H_{k+1} \right) P_{(k+1,k)}
\end{aligned}
\tag{6.1}
$$

It is assumed that a guess of the initial state, $\hat{x}_{0,0}$, as well as an initial input, u_0, are given to start the iteration process. The attentive reader notices the double subscript, which denotes the iteration step. The subscript is split into two parts, the first is for the prediction of the state and the second part is for the estimation. The importance of these parts is most prominent in step 3 of the filter, but more on this later in this section.

The first step predicts the next step based on a discrete version of the state space system given in Equation (3.2). The notation for the discrete state space system has been kept from the one given in Equation (3.3), with subscript d denoting the discrete-time nature of the matrices. More detail regarding discrete state propagation is given in Chapter 3. The construction of these matrices can be determined prior to the state estimation in case of white box modelling. On the contrary, when using black box modelling, they are constructed during the process based on sample collection (see Section 6.3). For sake of clarity the discrete state space notation used in this chapter is outlined in Equation (6.2), below. Furthermore, the Kalman filter requires a fully observable system to ensure convergence [40].

$$x_{k+1} = A_d x_k + B_d u_k + \Gamma w_{d,k} \tag{6.2}$$

After a prediction has been made in the first step, the second step will determine the covariance of the prediction error, $P_{k+1,k}$ by considering the discrete system dynamics, A and the discrete weight of the stochastic remainder of the state space system, Γ. Furthermore, the matrix $Q_{d,k} = \mathbb{E}\{w_{d,k}, w_{d,k}^T\}$ is used as well, in which $w_{d,k}$ represents the discrete white noise at time step k. For the second step, an initial covariance matrix, $P_{0,0}$, has to be guessed.
Next, the Kalman gain, K_{k+1}, is computed using the previously calculated covariance matrix of the prediction error, $P_{k+1,k}$ and a mapping matrix H. The latter is used to ensure the correct matrix sizing of K_{k+1} from the covariance matrix. A variable occurs in this step that has not been discussed so far, namely $R_{k+1} = \mathbb{E}\{v_{k+1}, v_{k+1}^T\}$. The discrete stochastic nature in the measurement at step $k+1$ is defined as v_k, in essence denoting the sensor noise. For sake of clarity, the measurement propagation is shown in Equation (6.3). Here, H appears again for the same reason as previously explained.

$$z_{k+1} = H_{k+1} x_{k+1} + v_{k+1} \tag{6.3}$$

The final estimate found by the Kalman filter for step $k+1$, namely $x_{(k+1,k+1)}$, is determined based on the measurement at step $k+1$, more precisely z_{k+1}. The Kalman gain is used to determine the contribution of the error of the difference between the measurement and the estimated measurement found according to Equation (6.3). The larger the gain, the more influence this error will have on the final estimate of the iteration step.
Finally, the covariance matrix of the state estimation, $P_{(k+1,k+1)}$ is computed. All characters for this step have been laid out previously. Consequently, this step is a straightforward calculation using the results of all previous steps. Once this covariance is known, the next iteration step is initiated. Where the first step uses the final estimate found in step 4 instead of the initial state previously used in step 1.

So far the linear Kalman filter has been described, which as the name suggests is valid for linear state space systems only. However, in case of non-linear systems, this filter is no longer valid and needs adaptation. The extended Kalman filter is defined for that purpose. Due to the summarising purpose of this report, the extended Kalman filter will not be discussed in this report, though further information can be found in [18, 40] .

6.3. Model Construction

By using state estimators such as the Kalman filter, a correct, or near-correct state can be provided to system identification algorithms. As such enabling more accurate tuning of the model parameters. Of course, before a model can be estimated, first the basis of that model is required. In this section three model constructs will be developed, namely state space system (Section 6.3.1), multivariate splines (Section 6.3.2) and feed forward ANN's (Section 6.3.3). The reason for only discussing these particular constructs is their range of applicability throughout linear and non-linear systems. Additionally, neither requires initial knowledge of the state space system dynamics, defining the scope of this literature study.

6.3.1. State Space System

The most basic model construct is the state space system. Although limited to linear system representations when the model parameters are strictly time-invariant, it can be used to model non-linear system by allowing the model parameters to be time-variant. In the latter case, the system is modeled linearly by a local estimation around a trim point. Resulting in a piece wise linear time-variant model.
The general definition of a state space model is repeated in Equation (6.4) for clarity and better understanding of the explanation in the subsequent paragraphs.

$$\dot{x} = Ax + Bu \tag{6.4}$$

In the notation provided in the equation above, $\{A, B\}$, are matrices representing the system dynamics and input dynamics, respectively. When identifying a system using the state space representation, the model parameters of A and B are estimated through sampling. One estimator used for this process is Ordinary Least Squares (OLS). In order to understand the estimation process more clearly, an intermezzo for OLS is given in the subsequent paragraph.

In OLS, a linear relationship, $Px = b$, is sought between parameters and the collected samples. The estimation is given by Equation (6.5) [6], where the regression matrix, P, and the regression vector, b, are sampled over a period over time. The regression matrix is conditioned on having full rank in order for the inversion of its product to be possible. Finally, the estimated parameters are contained in $\hat{\theta}$.

$$\hat{\theta} = \left(P^T \cdot P\right)^{-1} P^T b \tag{6.5}$$

The stability of the OLS estimator is directly linked to the determinant, and consequently the rank, of the regression matrix P. By means of linear algebra properties of matrices, the closer to zero the determinant of a matrix is, the more unlikely it's inversion exists. When the determinant of the regression matrix is too close to zero, the parameter estimation can become unstable, even resulting in completely erroneous estimations. In order to avoid such scenario, the number of samples contained in P and b can be increased. Indeed, increasing the samples, increases $|P|$. Finally, noise or another form of excitation is required to ensure a stable OLS estimate.

When using the OLS estimator for estimating a state space system representation of a system, the regression vector, b, becomes a matrix. If a discrete-time example is considered, the state space system can be rewritten in the format given in Equation (6.6). Meaning that when collecting samples, the matrix P will contain a vertical stacking of $\begin{bmatrix} x_k & u_k \end{bmatrix}^T$ and b of $\begin{bmatrix} x_{k+1} & u_{k+1} \end{bmatrix}^T$, also stacked vertically. As such, the resulting estimate, $\hat{\theta}$, is a matrix containing the values of the estimated state space matrices $\hat{A}$ and $\hat{B}$.

$$\begin{bmatrix} x_{k+1} \\ u_{k+1} \end{bmatrix} = \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} \tag{6.6}$$

The OLS estimator aims to minimize the cumulative square error, $\sum \epsilon_i = \sum (x_{t+i+1} - \hat{\theta} x_{t+i})^2$, where ϵ is the collection of the sample-wise errors. In the case of a state space system, ϵ is a matrix. Interestingly, the parametric covariance of θ can be computed by finding the variance of the error, σ_ϵ^2, and multiplying this with the OLS covariance matrix, $(P^T P)^{-1}$. First the variance of the error is calculated as given in Equation (6.7), in which the parameters n and k represent the number of samples and the number of regressor terms, respectively. Note, σ_ϵ^2 is a matrix, in which each element represents the variance of the estimate with respect to each column in ϵ. Since the the error variance is in matrix format, the Kronecker product is required to compute the parametric covariance of $\hat{\theta}$. The resulting equation to compute this covariance for the state space system estimation is given in Equation (6.8), which is adapted from ... to allow for σ_ϵ^2 in matrix format. Finally the parametric variance of the estimates $\hat{A}$ and $\hat{B}$ can be deduced by taking the elements on the diagonal of $Cov\{\hat{\theta}\}$

$$\sigma_\epsilon^2 = \frac{\epsilon^T \epsilon}{n - k} \tag{6.7}$$

$$Cov\{\hat{\theta}\} = \sigma_\epsilon^2 \otimes \left(P^T P\right)^{-1} \tag{6.8}$$

The parametric variance computed so far serves the purpose of finding the uncertainty interval bounds used in uncertain models. The importance of uncertain models is apparent in Section 5.3.3, where Safety Handling Exploration with Risk Perception Algorithm (SHERPA) uses such model at its core. Two probabilistic distributions are considered for each parameter. First, there is a continuous uniform distribution. In essence, this transcribes as a matrix element's value being equally probable within the complete uncertainty interval. If such distribution is bounded by an interval $[a, b]$, then its variance is defined as $\sigma^2 = \frac{(b-a)^2}{12}$ and its mean, $\mu = \frac{b-a}{2}$. Therefore, when conditioning the uncertainty interval to be $[a, b]$, the bounds can be written in terms of the distribution's variance and mean. In the case of continuous uniform distributions, this interval is given as $\left[\mu - \sqrt{3\sigma^2}, \mu + \sqrt{3\sigma^2}\right]$.

In a second instance, a continuous normal distribution was considered. The normal distribution has properties that are more realistic, as the further away from the mean a value is, the lower the probability that the matrix element will take that value. However, it comes with a drawback. Namely, that a normal distribution has no fixed bounds. Instead, distribution bounds can be set by conditioning a percentage of possible values to be within the bounds. In this report, bounds for a normal distribution are defined as a factor, λ, of the standard deviation of the distribution, σ. As such, the uncertainty interval is defined as $\left[\mu - \lambda\sqrt{\sigma^2}, \mu + \lambda\sqrt{\sigma^2}\right]$ when assuming a continuous normal distribution.

6.3.2. Multivariate Splines

Multivariate spline models use piece wise defined functions which are piece wise continuous [7]. The construct is based on polynomial approximation by splines. For sake of overview, first the concept of a 2-D spline will be explained, which serves as a foundation for the more advanced multivariate spline models. After which the concept of a simplex will be elaborated, resulting in the understanding of multivariate splines.

A spline interpolates a function, in the case of Figure 6.3 a polynomial function, at certain intervals using a predefined continuous basis function. In Figure 6.3, a linear spline interpolation is shown, where the basis function is linear, as the name suggests. Although the resulting interpolation is inaccurate with the chosen intervals, the latter can be changed to obtain a better fit. The linear basis function, however, has limited accuracy when it is applied on highly non-linear functions. In order to avoid this limitation, other basis functions such as quadratic and higher order splines exist. Their implementation is harder and requires more data to define the continuous basis function in a given interval. For example a linear spline requires two data points, whereas a quadratic one requires three. As the order of the spline increases, so does the amount of required data points. Additionally, the higher the order, the more the spline interpolation is prone to overfit the function.

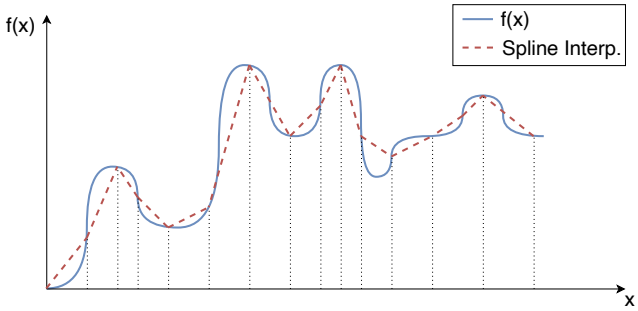


Figure 6.3: 2-D linear spline interpolation with variable step size of an arbitrary function.

Spline interpolations work by saving the parameters of the continuous basis functions at each interval in a table format. When a value, x_k, falls in a given interval, the output value will be extrapolated from the starting point of the interval based on the basis function parameters.

The spline interpolation mentioned so far is applicable for 2-D cases. However, more often than not, state space systems consist of multi-dimensional states. Thus, the state space can be seen as a multi-dimensional function. The linear spline interpolation should, as a consequence, be taken to a multi-dimensional level as well. If the focus is kept on the linear spline interpolation method for purpose of clarity, a linear basis function in an interval is called a simplex. The definition used in [7], is repeated in definition 6.2. According to this definition the linear spline interpolation shown in Figure 6.3, is constructed of 1-simplices (lines) along the intervals. A 2-simplex is a triangle, 3-simplex a tetrahedron, etc. The mathematical principles and coordinates of simplices are beyond the scope of this report and will not be discussed. However, more information in this regard can be found in [7] .

Definition 6.2: Simplex [7]

A simplex is a geometrical structure that minimally spans a set of dimensions.

This leads to multivariate splines, which are constructed by a table of parameters defining the simplices used to interpolate the multi-dimensional polynomial function. Multivariate splines follow the same accuracy principles as mentioned for 2-D linear spline interpolation. The higher the resolution (i.e., the smaller the intervals), the lower the interpolation error. This is of importance for highly non-linear state spaces with cliffs and edges. Higher resolution splines require more computation power and memory as a result of more simplices being used. In fact, the same curse of dimensionality as in Q-learning (see Section 2.5.2) occurs in multivariate splines and ANN's.

6.3.3. Artificial Neural Networks

Compared to multivariate splines, ANN models are applicable to the same variety of cases, with the main difference that it models the state space continuously instead of discretely splitting it into piece-wise continuous intervals as done by the multivariate splines.
Neural networks are based on the idea of interconnected neurons organised in layers. A generic ANN construct is given in Figure 6.4, displaying the three main layers in the structure.

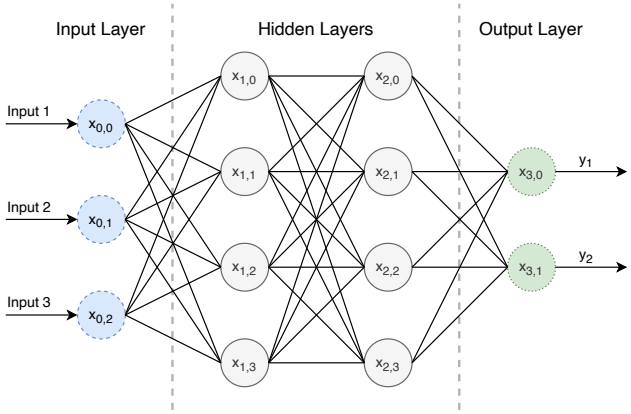


Figure 6.4: Schematic of an arbitrary Artificial Neural Network (Bias neurons not shown for sake of clarity)

Each layer consists of neurons, which individually sum all the inputs they get from the previous layer. Once summed, the result is passed through what is called an activation function, which converts the sum to the neuron's output. Examples of such activation functions are the sigmoid function, the Rectifier Linear Unit (ReLU) function or Radial Basis Function (RBF). The general formulation is given in Equation (6.9), showing the activation function of layer $i+1$, Φ_{i+1}, as well as the weight, $w_{i,j}$, of the link between the j^{th} neuron of previous layer, $x_{i,j}$ and the k^{th} neuron of the new layer $x_{i+1,k}$. Each neuron of the subsequent layer has a bias, $b_{i,k}$ associated with it as well.

$$x_{i+1,k} = \Phi_{i+1}\left(\sum_{j=0}^{M} w_{i,j}x_{i,j}\right) + b_{i,k} \qquad k = 0, 1, 2, ..., N \tag{6.9}$$

If this process, also known as forward propagation, is repeated through each layer, the network's result will be found in the output layer. In the scope of system identification, the ANN is a model of the system dynamics, thus the input is a state which is transformed to a subsequent set of state parameters as output. The transformation is mainly influenced by the weight of each individual linkage and the activation function. Assuming the activation functions are selected initially and stay fix afterwards, the weights remain the only factor that is variable. When a network is *learning*, it is adjusting its weights using a process called back propagation.

Back propagation is a process that contemplates the error of the network's output and compares it to a *true* value, that it receives from an external source (see Figure 6.1). Based on how different these two are, or how large the error is, the weights between the output layer and the last hidden layer are updated. Depending on how much these respective weights are changed, the weights between the layers before that are updated. The deeper the network, the longer this process takes. More specifically, weights are tuned using a method known as Stochastic Gradient Descent (SGD). SGD back propagation determines which weights to change, and by how much, in order to have the lowest possible predicted error. The stochastic part in the name is closely related to the ϵ-greedy implementation explained in Section 2.5. Stochasticity is introduced in the decision making in order to avoid overfitting of data sets. More on this phenomenon can be found in [9].
Mathematically the aforementioned back propagation, mitigating the error, is given by Equation (6.10), where the derivative of the error with respect to the weights is to be minimized. Using the chain rule of multi-variate functions, the derivative can be extended to what is shown on the right hand side of the equation.

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial e}\frac{\partial e}{\partial y}\frac{\partial y}{\partial \Phi}\frac{\partial \Phi}{\partial v}\frac{\partial v}{\partial w} \tag{6.10}$$

$$E = \sum_{j=0}^{N} \frac{1}{2}(y_j - d_j)^2 = \sum_{j=0}^{N} \frac{1}{2}e_j^2 \tag{6.11}$$

In Equation (6.10), E is the error vector, containing the quadratic errors, e_j of each state parameter with its respective counterpart, d (Equation (6.11)). The output of the network is given by the vector y (Figure 6.4 is helpful to visualise this). The activation function from which y is the result is denoted by Φ, in which the weighted sum of all neurons in the previous layer, v, provided as input. The process can be repeated from there to propagate backwards through all layer of the ANN.

$$w_{t+1} = w_t + \gamma\frac{\partial E}{\partial w_t} \tag{6.12}$$

Weights are updated using the same principle as presented in Section 2.3, where the discount factor γ is used to moderate the impact of past error gradients on the weights of the ANN.

6.4. Synthesis

The system identification framework allows for modelling of systems with (partially) unknown dynamics. In this chapter, the general framework is discussed along with system identification fundamentals. The difficulties of gathering truthful data to tune the model were solved using a Kalman filter to estimate the state from measurements. The chapter was ended with three model constructs. First, the more classical approach of state space systems is introduced. Although restricted to linear model representations, the applicability of this model construct shows more statistical tractability with respect to uncertainties. Second, the multivariate spline model was covered, along with the mathematical backbone of such models, namely simplices. This is followed by another black-box model construct, the Artificial Neural Network (ANN). Both constructs have the ability to be added to a RL agent-environment interaction in modular fashion.

7

Synthesis Literature Study

In conclusion of the literature study in this thesis, this chapter combines and unifies the contents of the multiple pillars described thus far. The ultimate purpose of this research is to design a controller for a given task without knowledge of the system and input dynamics. In Chapter 3, two optimal control tasks were uncovered, namely Linear Quadratic Regulation (LQR) and Linear Quadratic Tracking (LQT). To optimally perform these tasks, the choice was made to head for reinforcement learning techniques, such as SARSA or Q-learning. Literature further supports that choice as is the case in [22], for the tracking task and Q-learning. Combining Optimal control and reinforcement learning is not a problem, for linear systems at least. As mentioned in the conclusion of Chapter 3, more research needs to be carried out on the tracking of a reference signal by a reinforcement learning method for non-linear systems.
Rather rapidly, however, when the system's complexity increases, the reinforcement learning agent gets lost in the vast state space in which it finds itself. Optimisation techniques are slow and the cost functions are hard to optimise. This is where curriculum learning comes in. Imposing curriculum learning on an agent charged with a task associated with a cost function that is too complex, is simplifying the optimisation process. Based on literature, the task has to be understood reasonably well by an operator to define a correct curriculum. It can be assumed that based on the theory provided in Chapter 4, the problem of higher dimension state spaces for a Reinforcement Learning (RL) agent is lightened. The tracking task, as a result, has a higher probability of success. Although a curriculum lightens the optimisation process, the safety of the agent is not guaranteed as of yet. Indeed, RL processes have no inherent guarantees of safety preservation. Although the probability of ending in a fatal terminal state has been shown to decrease with curriculum learning, no guarantees have been addressed. Safe learning takes care of this, by keeping a log of all the states the agent has visited. Based on the contents of the log, the safe learning algorithm creates a belief of which states are safe and which are not. All this happens whilst the agent is being paced throughout the aforementioned curriculum. The safe learning algorithm considered in this research, is specifically SHERPA. This method comes with a drawback, namely its dependence on approximate system dynamics. In order to relief the complete algorithm described so far from this dependency, a system identification method is attached to the algorithm. It aims to model the system dynamics as the agent explores the state space. This model is tied to parametric uncertainty of its inherent parameters. When the uncertainty is low enough, the model more accurately represents the system. Consequently, the safe policy search within SHERPA is more probable to be successful.

This brings the literature study to an end. Using the information found in literature, most of the research sub-questions can be answered from a theoretical standpoint. The complete answers to these questions are addressed in conclusion of this report (Chapter 10). Although theoretical concepts provide an insightful idea of the conceptual success of the proposed method, the findings require empirical backing. Therefore, a more practical implementation would be suitable to provide a proof of concept. The second part of this report presents a preliminary analysis to fulfill this requirement.

III

Preliminary Analysis

8

Safe Curriculum Learning for Linear Quadratic Tracking Tasks on MSD System

As a proof of concept, the preliminary analysis found here, contains a practical implementation where the principles presented in the literature study (Part II) are combined. Together, they construct the paradigm proposed by this research. The preliminary analysis supports this purpose, by defining a more comprehensible environment on which to try the fundamentals of safe curriculum learning for systems with unknown dynamics. As environment for the experiment, a cascaded Mass-Spring-Damper (MSD) system of N masses was chosen. In this preliminary research, the analysis will be split in four different implementation phases, provided in schematic form in Figure 8.1.

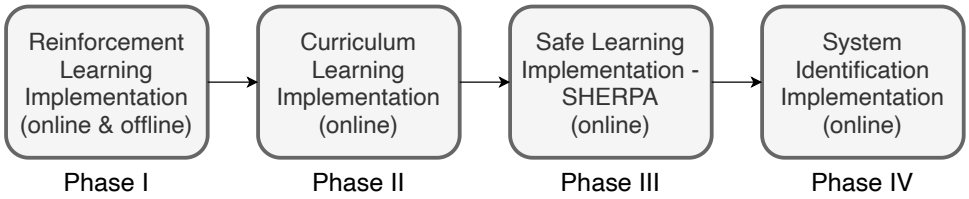


Figure 8.1: Implementation phases of safe curriculum learning for systems with unknown dynamics in the preliminary analysis.

The first phase consists of constructing the environment, an N-MSD system and an agent that can interact and learn within this environment. Both online and offline implementations are trialed to assess the benefits of one or the other. In a second phase, the agent is put through a curriculum, comparing it to a flat learning benchmark set in the first phase. Phase three, is the incorporation of the safety aspect, presented in Chapter 5, in the form of the Safety Handling Exploration with Risk Perception Algorithm (SHERPA). Finally, the last phase will introduce a system identification method to the complete system in order to allow SHERPA to work on systems with unknown dynamics.

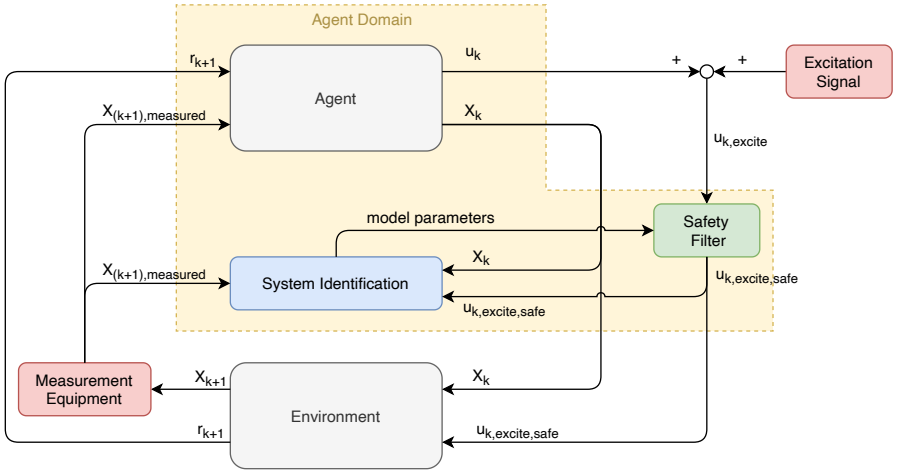


Figure 8.2: Schematic overview of the implementation for a single curricular step. Each block represents a module.

In Figure 8.2, an overview is given regarding the modular nature of the implementation. The schematic represents the core agent-environment interaction of a single curricular step. Each block shows a module incorporated in the final implementation. By virtue of the modular nature, the *Safety Filter*, *System Identification* and *Measurement Equipment* modules can be removed to show their impact on the total learning process.

8.1. Analysis Setup

The choice was made to use an MSD system as a benchmark for the preliminary analysis due to its simple dynamics and its easy scalability as well as its linearity. Besides, the observability and controllability of the system can be adapted to complicate the learning task of the agent.

Table 8.1: Characteristic parameters of the unstable 3-MSD system shown in Figure 8.3

Mass	m $[kg]$	k $[N/m]$	c $[N/(ms^{-1})]$
1	0.3	-1	6
2	0.8	3	-1
3	0.4	4	3

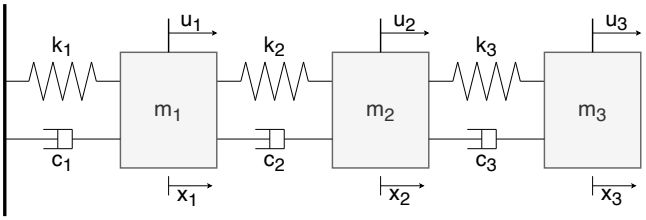


Figure 8.3: Figure showing a cascaded 3-MSD system

The analysis will be performed on a 3-MSD unstable system (Figure 8.3), with characteristic parameters given in Table 8.1 and for which symbolic state space is derived in Appendix B and is repeated in Equation (B.10). This is a linear, continuous-time, fully observable and fully controllable state space system. The observability of the system can be changed by reducing the size of the state seen by the controller. Likewise, the controllability can be modified by changing the amount of masses to which the controller can apply an input force.

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -\frac{k_1+k_2}{m_1} & \frac{k_2}{m_1} & 0 & -\frac{c_1+c_2}{m_1} & \frac{c_2}{m_1} & 0 \\ \frac{k_2}{m_2} & -\frac{k_2+k_3}{m_2} & \frac{k_3}{m_2} & \frac{c_2}{m_2} & -\frac{c_2+c_3}{m_2} & \frac{c_3}{m_2} \\ 0 & \frac{k_3}{m_3} & -\frac{k_3}{m_3} & 0 & \frac{c_3}{m_3} & -\frac{c_3}{m_3} \end{bmatrix} \; ; \; B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{m_1} & 0 & 0 \\ 0 & \frac{1}{m_2} & 0 \\ 0 & 0 & \frac{1}{m_3} \end{bmatrix} \tag{B.10}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \; ; \; D = \begin{bmatrix} 0 \end{bmatrix}$$

The purpose of the exercise is to have the position of the third mass, x_3, track a sinusoidal reference signal, by controlling the available input forces. All the while, aiming to regulate the positions of the remaining masses, $\{m_1, m_2\}$, to zero. A more detailed explanation of the problem and the composition of the required matrices is given in Appendix C, and will not be repeated in this chapter due to the mere size of these matrices. However, the reader is advised to get more profound understanding of the tracking task by reading this appendix.

The optimal policy for the control of the Linear Quadratic Tracking (LQT) tracking task will be found using Reinforcement Learning (RL) techniques, more specifically online Value Iteration (VI) within the on-policy Q-learning framework. The pseudo algorithm for this method is outlined in algorithm 3.3.1. It was opted for an online method, as the ultimate goal is to have the ability to apply the methods used in the preliminary analysis to a dynamic system with unknown dynamics. Meaning that system dynamics models for simulations might not be available for this future system.
Terminology from RL concepts will be translated in terms of control jargon. The state and action from the theory presented in Chapter 2 will be converted as $s \rightarrow x$ and $a \rightarrow u$, respectively. Additionally, the action-value function $Q(s,a)$ from algorithm 3.3.1 is defined as the Q-function formulated in Equation (8.1).

$$Q(x,u) = \begin{bmatrix} X \\ u \end{bmatrix}^T \begin{bmatrix} H_{XX} & H_{Xu} \\ H_{uX} & H_{uu} \end{bmatrix} \begin{bmatrix} X \\ u \end{bmatrix} = Z^T H Z \tag{8.1}$$

The agent's learning process consists of two parts, policy evaluation and policy improvement. Since VI is used, first policy evaluation is performed after which policy improvement will occur. As discussed in Section 3.2.4, policy evaluation will require the solution for H from the discrete-time formulation shown in Equation (3.21). Policy improvement takes the form of deriving the optimal policy from the kernel matrix H, according to Equation (3.6).

$$Z_k^T H^{j+1} Z_k = X_k^T Q_1 X_k + u_k^T R u_k + \gamma Z_{k+1}^T H^j Z_{k+1} \tag{3.21}$$

$$u_k^* = -\left(H_{uu}^{-1} H_{uX}\right) X_k \tag{8.2}$$

For convergence of the online VI algorithm, a Persistent Excitation (PE) condition applies (described in Section 3.3.1). More specifically, the input, u, has to be persistently excited. For the simulation results presented in this chapter, random white noise following a Gaussian bell-curve with $\{\mu = 0, \sigma = 0.5\}$ was used as an excitation signal. White noise was chosen to avoid the case of perfect tracking control (i.e. when the input follows a perfect sinusoidal form), which would integrate to zero in certain time intervals.

The dimensions of the different components of Z are given as n, m and p, for system state $x \in \mathbb{R}^n$, input $u \in \mathbb{R}^m$ and reference signal $x^r \in \mathbb{R}^m$, respectively. In the case of a fully observed and controlled 3-MSD system on the aforementioned LQT tracking task, $n = 6$, $m = 3$ and $p = 2$ (F matrix in Appendix C has size (2×2).

The remainder of this chapter will focus on simulation results. First, a benchmark is set by presenting results of a 3-MSD system where no curriculum learning is applied in Section 8.2. After which, Section 8.3 showcases the results for a curriculum test case with semantic kernel mapping. On the latter, a SHERPA safety filter is added to take the safety aspect of the learning process into consideration. Finally, the chapter is concluded by a hyperparameter sensitivity analysis and a conclusion of the experiment.

8.2. Flat Learning Benchmark

Since a comparison is made between multiple methods, a benchmark must be created with which other methods can be compared. In this section, a closer look will be given to the implementation of online Q-learning as well as its offline counterpart. The total training time is set to 30 seconds for which the discrepancies in learning behaviour will be discussed in this section.
In order to show the simulation results, a couple of additional parameters have to be defined. Namely, the weight matrices Q and R. These are diagonal matrices with $1e5$ and $1e1$ as diagonal values respectively. Implicitly, this means that the difference between the system state and the reference signal, e, has much more importance in the optimisation process than the magnitude of the input forces. The selection of these weights was based on the fact that in case e is large, the agent would explore the influence of large input forces to diminish this error. If R is given higher values, the tendency to use large forces would come at a cost, in a quadratic fashion, which the agent would avoid. On the other hand, the high values in Q, incentivise the agent to close the gap between the reference signal and the state as quickly as possible.
The initial kernel matrix, H^0, is initiated as an identity matrix $I_{((n+m+p)\times(n+m+p))}$, from which the initial control policy is derived (Equation (8.2)).
Finally, the initial conditions for mass positions and velocities, are picked randomly from intervals $x_i \in [-0.2, 0.2]$ and $\dot{x}_i \in [-0.25, 0.25]$, respectively. No initial force is applied externally on the system, except for the random noise added for the PE condition.

First, online VI is discussed. The agent is given full control as well as full observation of the system and 30 seconds of learning time. The result is presented in Figure 8.4. In this stacked plot figure, the first chart shows the positions of all the masses of the unstable system, as well as the reference signal for the last mass. Additionally, the dash-dotted line shows the time at which the policy evaluation and policy improvement steps are performed.

From the first plot in Figure 8.4, it can be seen that although the agent aims to control the system, the tracking task is performed very poorly. Indeed, the position of the third mass error becomes increasingly large. The behaviour of the system is a result of the lack of control during the first sample interval, since the policy's gain is filled with zeros. Consequently, after the first update the agent is none the wiser of the input's influence, thus increasing the gain slightly at every update. Meanwhile, the system reaches states that are close to uncontrollable, rendering further updates more complex for the agent. This is mirrored in the evolution of the closed-loop eigenvalues' real part shown in Figure 8.5a and the evolution of the kernel diagonal values in Figure 8.5b. These time series show a strongly unstable learning pattern, which is undesired. The kernel update executed at 23.1 seconds has a policy leading to all negative closed-loop eigenvalues, which is desired. However, they are accompanied by negative values on the kernel diagonal. That cannot be the case. As such, the following update corrects for the diagonal kernel values, at the cost of getting an unstable policy once again. All the while, unfortunately, the system gets out of control, making updates even more complex. This vicious cycle sees no solution, even when giving the agent a learning time of 100 seconds. In fact, the behaviour described above, is true for 100% of the simulations performed with the aforementioned conditions. The statistic is based on a total of 400 independent simulations, all resulting in unstable policies after 30 seconds of learning. One way of avoiding such behaviour, might be to arbitrarily initiate H^0, such that the initial policy is non-zero.

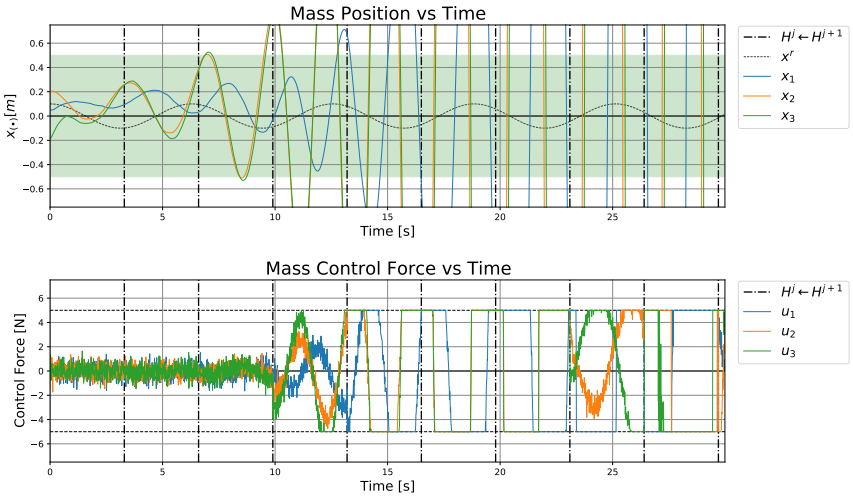
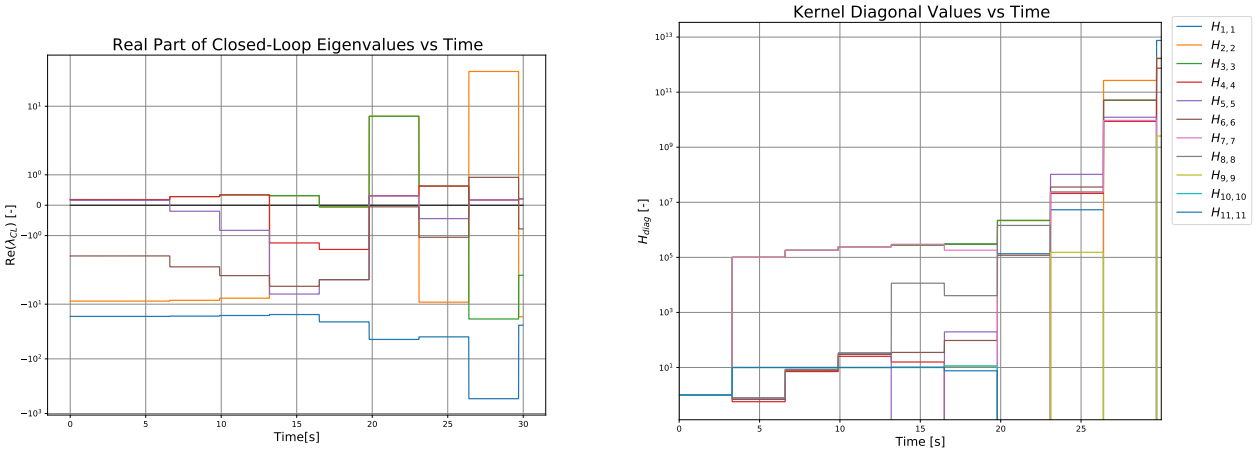


Figure 8.4: Evolution of mass positions over time; Online VI; Flat Learning; No SHERPA; No sensor noise; $x_i \in [-0.2, 0.2]$; $\dot{x}_i \in [-0.25, 0.25]$; Green is the SSS for x_3.



(a) Evolution of the real part of the 3-MSD system's closed-loop eigenvalues; Online VI; Flat Learning; No SHERPA; No sensor noise.

(b) Evolution over time of the diagonal values of H for the 3-MSD system; Online VI; Flat Learning; No SHERPA; No sensor noise.

Figure 8.5: Evolution of closed-loop eigenvalues and kernel diagonal values over time; Online VI; Flat Learning; No SHERPA; No sensor noise.

Another way of avoiding the instability in the system and for it to get out of hand, is offline learning. Where the system is reset to a random initial state and the agent continues learning from there. Notably not online learning, it would prove that the agent has the capacity to learn the optimal policy. For this simulation, it was opted to split the learning into three episodes of 10 seconds each, totalling the same 30 seconds as for the online case. For sake of clarity, only the final episode will be shown in this chapter. The first two episodes have been added in the Appendix D in figures D.1 and D.3.
Although the agent has had the same learning time, the result is objectively much better. The first mass is close to be regulated and so is the second. Furthermore, the third mass is oscillating slightly around the reference signal. Again, this is reflected in the evolution of the closed-loop eigenvalues and kernel diagonal values of this episode shown in Figure 8.7. The first two episodes are presented in Appendix D, more specifically figures D.2 and D.4.

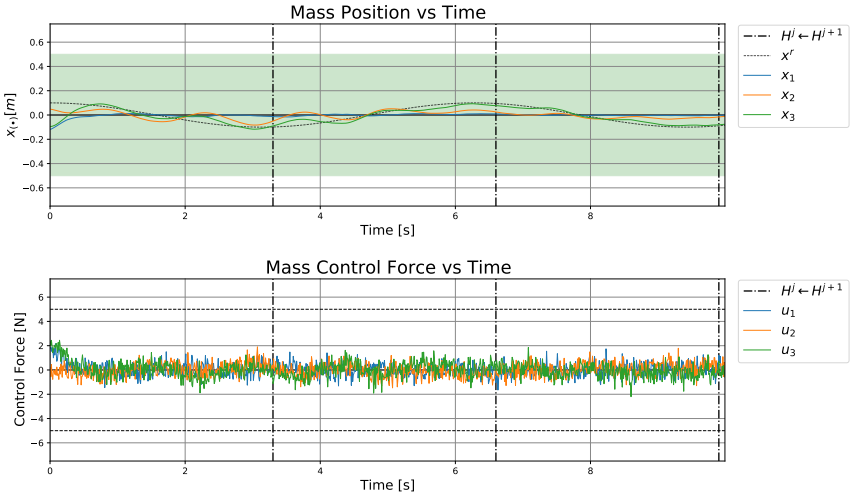
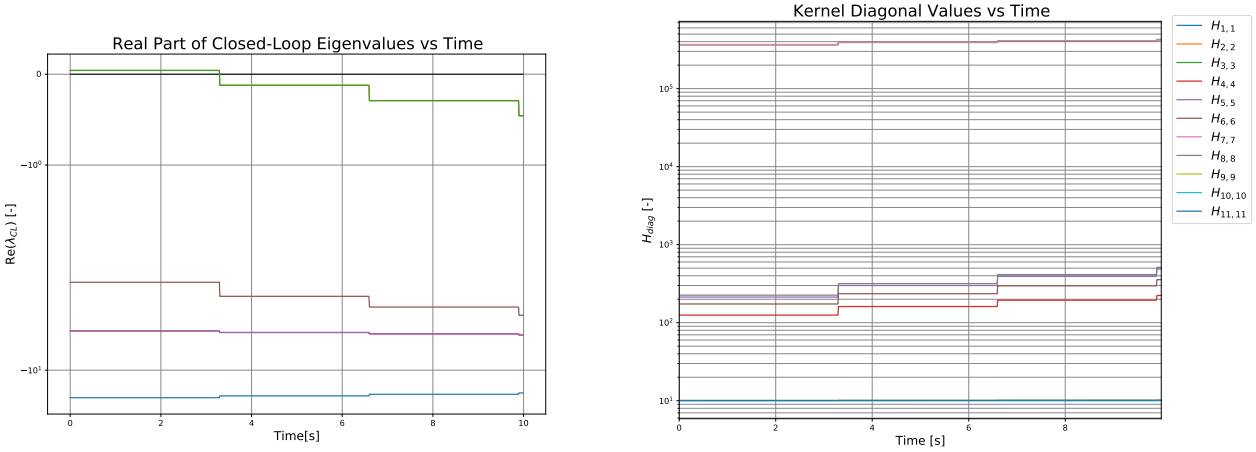


Figure 8.6: Evolution of mass positions over time; Offline VI episode 3 (episodes 1 and 2 are shown in figures D.1 and D.3, respectively); Flat Learning; No SHERPA; No sensor noise; $x_i \in [-0.2, 0.2]$; $\dot{x}_i \in [-0.25, 0.25]$; Green is the SSS for x_3.



(a) Evolution of the real part of the 3-MSD system's closed-loop eigenvalues; Offline VI episode 3 (episodes 1 and 2 are shown in figures D.2a and D.4a, respectively); Flat Learning; No SHERPA; No sensor noise.

(b) Evolution over time of the diagonal values of H for the 3-MSD system; Offline VI episode 3 (episodes 1 and 2 are shown in figures D.2b and D.4b, respectively); Flat Learning; No SHERPA; No sensor noise.

Figure 8.7: Evolution of closed-loop eigenvalues and kernel diagonal values over time; Offline VI episode 3; Flat Learning; No SHERPA; No sensor noise.

Most prominently in the first episode, Figure D.1, it can be seen that when the episode terminates, the agent is suffering from action saturation. If an action is saturated for too long between two updates, the agent has difficulty to find a relationship between its input and the effect it has on the system. However, by terminating the episode after 10s and consequently resetting the system to a random initial state, the time actions are saturated are limited and unstable updates of the kernel matrix H are avoided. When an offline reset is performed, the agent keeps the knowledge it gathered during the previous episode, in the form of its kernel matrix and policy. Offline learning has proven to be more convergent with 96% of simulations (each with three episodes) converged to a stabilising policy. Unfortunately, in this preliminary analysis, an online learning process is sought. Offline learning proved that the agent can indeed learn with VI, when the learning process is reset, which can be seen as a simplification. Using online learning clearly requires some sort of curriculum to simplify task at hand.

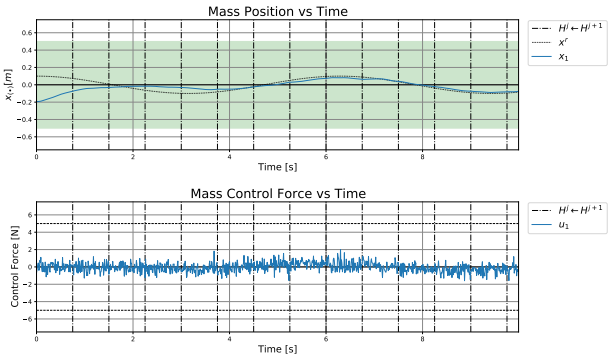
8.3. Curriculum Learning

Due to the unstable nature of the system, a stabilising policy is hard to find with online learning. However, as proven in the previous section, offline learning with equal cumulative training time as online learning, is already converging to a convenient policy. Specifically, one that stabilises the system to have the last mass track the reference signal, and the the two other masses come to a standstill.

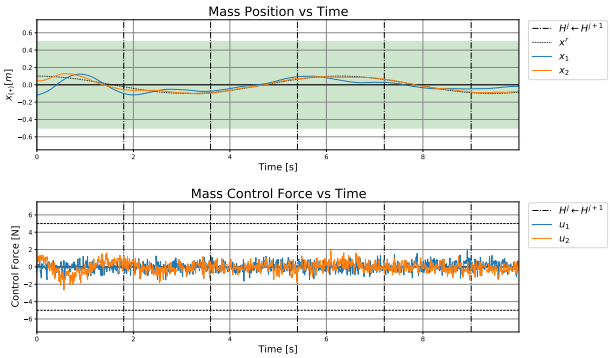
In this section the aim is to converge to such policy in an online learning process by using curriculum learning, discussed in more detail in Chapter 4. The benchmark of 30 seconds of cumulative offline training time is taken as a reference. The curriculum discussed here uses semantic transfer learning as a method of choice to transfer knowledge between curricular steps. Additionally, the entropy in the curriculum in increased by rising the task complexity. Indeed, by increasing the number of masses in the MSD system, the tracking task outlined in previous section is made more complex. Thus, at each curricular step, a mass is added to the MSD system. The order in which masses are added, is the one from Table 8.1 in reverse (starting with the third element and then the second, finishing with the first element). The reason for that specific order is to have a stable system initially, gradually adding instability with new masses.

For transferring knowledge between tasks, a semantic mapping of the kernel matrix, H, is used. In this case, once the optimal policy is found for the previous curricular step, the parameters of the kernel matrix that relate to the mass(es) in the respective places of the system are simply copied over. Whereas the other values of the new kernel matrix are kept at zero.

The results of an exemplar curriculum are presented in figures 8.8a, 8.8b and 8.9, in order of curricular steps. The first curricular step has the controller familiarise with a stable version of the system, since only the third mass m_3 of Table 8.1, with its spring and damper constants, k_3, c_3, constitute the 1-MSD system. As expected for a simple and stable system, the agent learns relatively quickly how to track the reference signal, as can be seen in Figure 8.8a. The kernel parameters found in the the first curricular step are then mapped semantically to a kernel matrix of appropriate size for a 2-MSD system. More specifically, the parameters found in the first curricular step are matched with the kernel parameters of the second curricular step related to the last mass in the system. The latter mass is the one that is tracking the reference signal, and for which a policy has been found already. However, for the other mass, which is to be regulated, no policy has been sought yet. Therefore, the kernel parameters linked to this mass are kept zero.



(a) Evolution of mass positions over time; Online VI; Curriculum Learning (step 1); No SHERPA; No sensor noise; $x_i \in [-0.2, 0.2]$; $\dot{x}_i \in [-0.25, 0.25]$; Green is the SSS for x_1.



(b) Evolution of mass positions over time; Online VI; Curriculum Learning (step 2); No SHERPA; No sensor noise; $x_i \in [-0.2, 0.2]$; $\dot{x}_i \in [-0.25, 0.25]$; Green is the SSS for x_2

Figure 8.8: Evolution of mass positions over time; Online VI; Curriculum Learning (steps 1 and 2); No SHERPA; No sensor noise; Enlarged images are repeated in figures D.5 and D.6 for better readability.

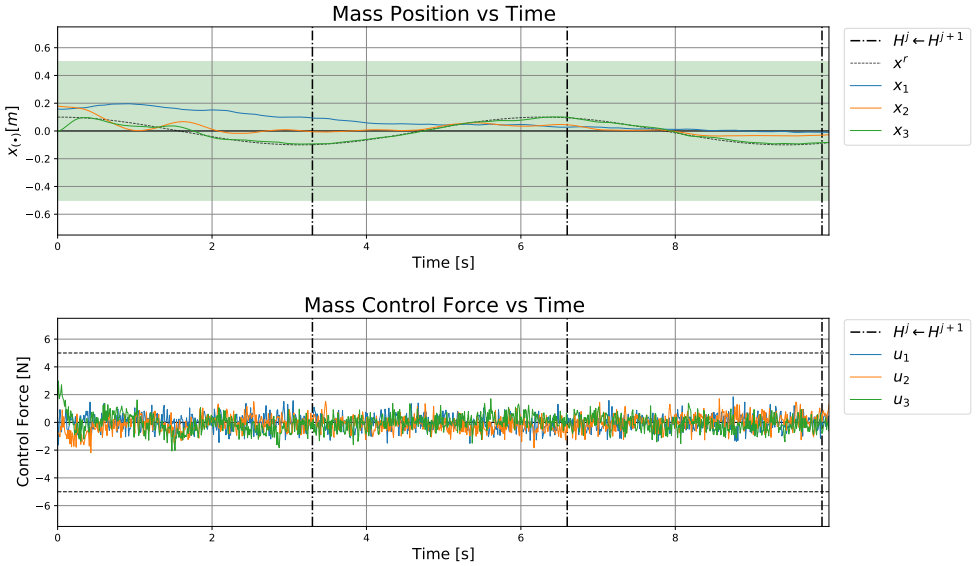


Figure 8.9: Evolution of mass positions over time; Online VI; Curriculum Learning (step 3); No SHERPA; No sensor noise; $x_i \in [-0.2, 0.2]$; $\dot{x}_i \in [-0.25, 0.25]$; Green is the SSS for x_3.

Again, a policy is found to regulate the mass connected to the wall and force the last mass at the end to track the reference signal. After the second curricular step, the kernel matrix, H, contains parameters linked to regulating a mass and to force a mass to track a reference.
For the final curricular step, the kernel parameters are semantically mapped again and a last mass is added between the wall and the aforementioned 2-MSD system. For this last step, the parameters linked to regulation could be mapped to the newly added mass, since it has to be regulated as well. However, in the results presented in Figure 8.9, this is not the case. Like the 2-MSD system, the kernel parameters linked to the new mass are kept as zero. Conveniently, the agent quickly learns to regulate the middle mass, shown in Figure 8.9 as x_2, and to track the reference signal with the last mass, shown as x_3, since their kernel values were mapped from the former curricular step. The mass connected to the wall, x_1, takes a bit longer to be regulated, since no knowledge was transferred for its regulation. With time, however, the complete system is regulated with the exception of the last mass which tracks the reference signal.
In terms of learning, the curriculum learning process exhibits stable convergence towards a policy that is stabilising the system consistently. From a statistical point of view, the convergence rate has been 99% based on 200 runs with three curricular steps each. All converging cases did so consistently across the curriculum. The reason for the 2 exceptions in this statistic are due to a rare occasion where the persistence of excitement condition would not be satisfied and thus the kernel matrix being updated in an incorrect manner (more on this is given in Section 3.3.1).

An additional benefit of curriculum learning for N-MSD systems in the case shown in Figures 8.8 and 8.9, is that from a safety perspective, the end mass stays within its Safe State Space (SSS) (green area) at all times when a stabilising policy is found. Although this is sought, the algorithm presented in this work considers the safety aspect of learning as well. In order to make the system adventure in the Fatal State Space (FSS), noise is added to the state. This is presented in the subsequent subsection.

8.3.1. Addition of Measurement Noise

In order to force the system out of the SSS, it is proposed to add measurement noise or sensor noise to the states. The way this can be interpreted, is that the mass positions and velocities are measured by some type of sensor. Practically speaking, sensors provide measurements with a certain uncertainty, which is generally translated to white noise. In this report, sensor noise is considered to be Gaussian white. In addition to measurement noise, the persistence of excitation signal's amplitude has been tripled to complicate the learning task.

For the example generated in this section, sensor noise is defined with as Gaussian white noise with standard deviation, $\sigma = 0.002$, for all states. After simulating the first curricular step, it was found that the learning process becomes unstable when sensor noise is added to the state. The behaviour is attributed to the noise being equal for both the position states, $x_{(\bullet)}$, and velocity states, $\dot{x}_{(\bullet)}$. For discrete-time state space systems, states which are derivatives of other states should have noise that is a factor of dt. Therefore, in the case of $dt = 0.01$, the noise on the positions states is actually a hundred times more prominent than it should be. Since the purpose of this section is to create a state propagation that leaves the SSS, it is not of importance to adapt the implementation in this regard. Additionally, only the first curricular step is presented, since no stabilising policy is found during this step. As a consequence, subsequent steps receive an initial policy that does not contribute to an improved learning process in those steps.

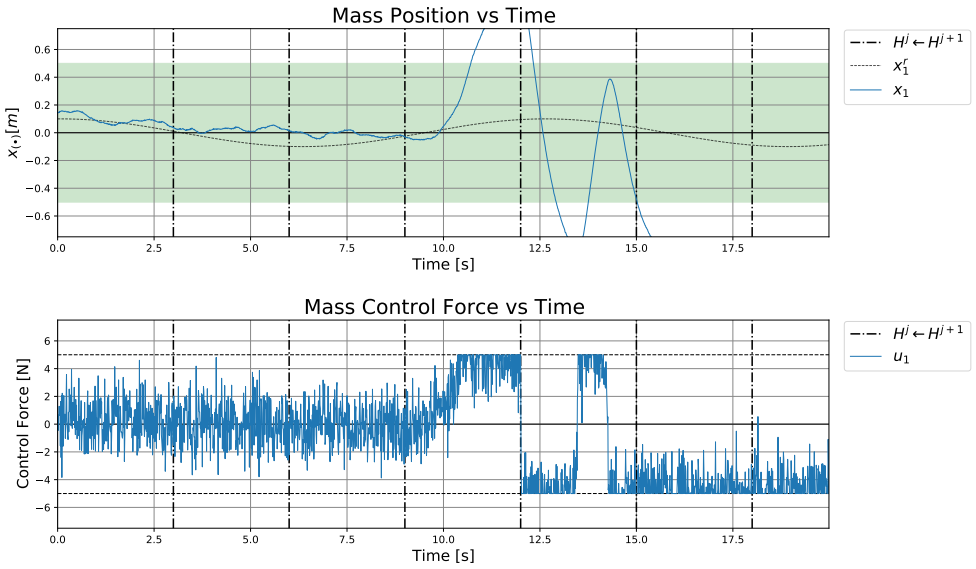


Figure 8.10: Evolution of mass position, x_1, over time; Online VI; Curriculum Learning (step 1); No SHERPA; Gaussian sensor noise, $\sigma = 0.002$; $x_i \in [-0.2, 0.2]$; $\dot{x}_i \in [-0.25, 0.25]$; Green is the SSS for x_1

The conclusion that can be taken from Figure 8.10, is that adding sensor noise results in an unstable learning process where the system does not stay within the SSS. In order to keep the system in the safe region, the SHERPA safety filter is used.

8.4. Safe Learning

In order to keep a system with sensor noise within its safe region, a safety mechanism is required. In this analysis, use is made of SHERPA, which acts as a safety filter on the agent's actions. First, details regarding the assumptions and decisions made during the implementation of SHERPA are laid out in Section 8.4.1. Thereafter, results of the inclusion of the safety filter module are provided in Section 8.4.3, where the benefits of having a safety filter on the example presented in Figure 8.10 is shown. Alongside, the integration of SHERPA in a curriculum with measurement noise is included in this last subsection.

8.4.1. SHERPA Implementation Setup

The safety filter of choice in this analysis is SHERPA. Although the complexity of this algorithm does facilitate implementation, it comes with great benefits regarding the guarantees regarding safety as is laid out in more detail in [32]. Practically speaking, the safety filter is implemented with three modes, namely *action suggestion*, *use backup policy* and *random action*. Each of which is discussed in the remainder of this paragraph.
When suggesting an action, it means SHERPA was not able to find a backup policy for the state-action pair it received from the agent and that the subsequent time step is considered safe by SHERPA. As a consequence, it goes through random actions and tries finding a backup policy for those. In the implementation presented here, this is limited to 10 iterations or tries. It was found empirically, optimising for minimum run time and maximal probability of success in finding a backup policy. More on the policy search in a subsequent paragraph.

In the second mode, *use backup policy*, SHERPA has not been able to find a backup policy for 10 different state-action pairs, including the initial proposal from the agent. As such, the algorithm falls back on a backup policy it found previously. As is described later in this section, the policy search tries a number of policies for which it propagates the state for 20 time steps. Consequently, when using a backup policy, the agent's actions are corrected by SHERPA for 20 time steps, in order to remain consistent with the projected time frame required for a backup policy to correct the trajectory. In addition to following a backup policy, this mode searches of a backup policy at the projected arrival state of current projected trajectory. It should be noted, that for this mode, the projected state at the subsequent time step has to be safe. More on the said safety in a subsequent paragraph.
Finally, there is the last resort is a random action. This mode is solely called into action when the state at the subsequent time step is not safe and no backup policy is found for the given state-action pair, provided by the agent and the 10 other iterations. This mode is undesired as it is difficult to predict its outcome.

As is becoming apparent, selecting the mode SHERPA is using, depends on two main conditions. The first being safety of the projected state at the subsequent time step. Indeed, as presented in [32], the proceedings for SHERPA require the subsequent state to be within the SSS. For a N-MSD system, the SSS and FSS are limited to the position state element of the last mass, which in terms of safe learning is defined as the Restricted State Space (RSS). As such, when mentioning the SSS and FSS, these are in reference to the RSS.
From Definitions 5.6 and 5.7, it is clear that the FSS is fixed and defined externally. In this preliminary analysis, the FSS bounds have been set to $[-0.5, 0.5]$ m. On the other hand, the SSS is expanding over time as the agent explores the state space. Furthermore, the expansion of the SSS is linked to a risk function. Here the risk function is chosen to be binary. More specifically, risk is defined as 0% when the state summed with a sensor reach is not within the FSS. The sensor reach can be interpreted as an ultrasonic radar sensor with a certain reach checking for a wall. The latter representing the FSS. A sensor reach of 0.1 m is defined for the simulations present in this section. In contrast, the risk is 100%, when the sum of the state and sensor reach is within the FSS. When risk is 0%, the state is considered safe and the SSS bounds are updated to contain the state and its sensor reach.
The second condition for mode selection in SHERPA is whether or not a backup policy can be found. Although the backup policy search is defined conceptually in [32], a more practical definition is given here. In this implementation, each time a backup policy is sought, a three-phase sequence is executed:

1. Generate policies
2. Propagate state using a policy and the uncertain state space model
3. Run safety and closeness checks

It was chosen that 10 policies are generated for each policy search. Generating policies uses a logic resembling the ϵ-greedy method explained in Section 2.5.1. As basis, the agent's current policy is used, which is parameter-wise adapted in a random fashion. For greedy changes, each element is multiplied with a random factor $\lambda \in [0.85, 1.15]$. On the other hand, with probability $\alpha = 10\%$, a completely random policy is generated. Forcing SHERPA to explore the policy space increases the chance of finding a near-optimal backup policy. The generate policies are then used one-by-one to project a state propagation using an uncertain model. Leading to the second process in the policy search.
Projecting an uncertain trajectory with state space systems uses uncertain models. A full derivation and validation of the algebra used for these calculations is given in Appendix A. In the N-MSD implementation presented in this section, the number of time steps to be projected is set to 20. Depending on the uncertainty of the model being used, the uncertainty bounds of the arrival state in this projection may vary greatly as is apparent from Appendix A. If input saturation is applicable on the agent, it is also applied in the uncertain state propagation process.
The last process in finding a policy, is assessing its performance. A backup policy is evaluated based on two criteria, firstly on its compliance with the ergodicity condition and second on the safety of its projected trajectory. The safety criterion is determined based on the series of all projected states being within the SSS and inherently related to ergodicity. When a backup policy ends in a state that has been previously visited, the ergodicity condition is fulfilled. However, since the states are continuous, in the sense that they have no rounding limitations nor value bounds, a closeness condition is of application. The closeness condition state that if a state is close enough to a previously visited state, then ergodicity complied with. When working with uncertain states, it has one additional condition to satisfy, namely that the uncertainty interval of the state can not be larger than twice the closeness condition, δ. For position and velocity states, the distance, δ, is set to 0.2 and 0.2, respectively. Which means that the uncertainty interval of the state requires an width of 0.4 or less. Otherwise, not all possible projections can fulfill the closeness condition, and as such the ergodicity condition.

For the sake of overview and clarity the numerous hyperparameters chosen for SHERPA are summarised in the table below.

Table 8.2: Hyperparameters for the SHERPA safety filter

Parameter	Value	Unit
# backup policy search tries	10	-
# policies checked	10	-
# time steps for projection	20	-
FSS bounds	[-0.5, 0.5]	m
Sensor reach	0.1	m
Probability to take a completely random policy, α	10	%
Parametric policy shift factor, λ	[0.85, 1.15]	-
Closeness condition position states, δ_x	0.2	[m]
Closeness condition velocity states, $\delta_{\dot{x}}$	0.2	[m]

Given the presentation of the SHERPA algorithm is finished and the hyperparameters explained, it is possible to present results of such an implementation and deduce the advantages and drawbacks of adding a safety module to the implementation.

8.4.2. System Identification

Inherently, SHERPA is dependent on an uncertain model, which is provided by the system identification module. In this subsection the practical aspects of this module are discussed. In the implementation discussed in this chapter, system identification is performed using Ordinary Least Squares (OLS) to estimate the discrete-time state space system (more details are given in Section 6.3.1). The module has as goal to identify the parameters of A, B and F. Thus, not only the system dynamics, but also the reference dynamics. Consequently the total number of parameters identified is $p^2 + n(n+m)$. As was the case for the policy update process in the agent, the system identification method uses a sample factor to collect enough samples before updating its estimate. In the case of a state submitted to sensor noise, the sample factor is set to 9.
The concept of a sliding window, in sorts, is used in more complex tasks. In such tasks, the impact of a lower uncertainty in the initial time steps has great consequences for the policy search in SHERPA. Consequently, it is desired to have updates, with lower uncertainties, as soon as they are available. It should be noted that in the case of a N-MSD system, which is defined by a linear time-invariant state space system, all samples collected since the beginning can be kept for new estimates. In case of time-variant state space systems, the sliding window can not be used, as the samples collected for a previous update are no longer valid for the current update.

8.4.3. SHERPA results

The inclusion of SHERPA within the bigger implementation is expected to have impact on unstable policy updates by the agent. In essence, keeping the agent at bay when its unstable policy would lead it into the FSS. This subsection is divided into two parts. First the impact of a safety filter is examined on a single curricular step. In a second instance, the inclusion of SHERPA in a full curriculum is inspected.

Initially, the insertion of a safety filter is scrutinised for its impact on a single curricular step with measurement noise. More specifically, the same configuration is used as the one that generated Figure 8.10. Note that the initial state, persistence of excitation and measurement noise are all random processes, which leads to the state propagation not being the same as the aforementioned figure. The resulting learning process is shown in Figure 8.11, in the form of the familiar state propagation plot. One key difference with previous results of this form is the display of the FSS instead of the SSS. Since these results are viewed from a SHERPA perspective, the FSS is fixed over time whereas the SSS is an evolving space. Previously, the SSS was assumed to have been fully explored and thus, known.

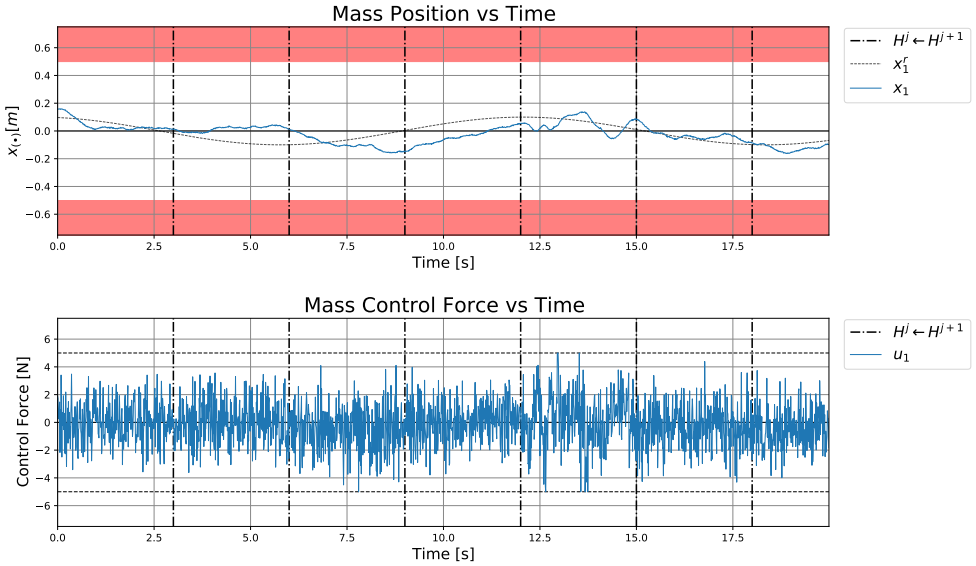


Figure 8.11: Evolution of mass position, x_1, over time; Online VI; Safe Curriculum Learning (step 1); SHERPA activated; Gaussian sensor noise, $\sigma = 0.002$; $x_i \in [-0.2, 0.2]$; $\dot{x}_i \in [-0.25, 0.25]$; Red is FSS for x_1.

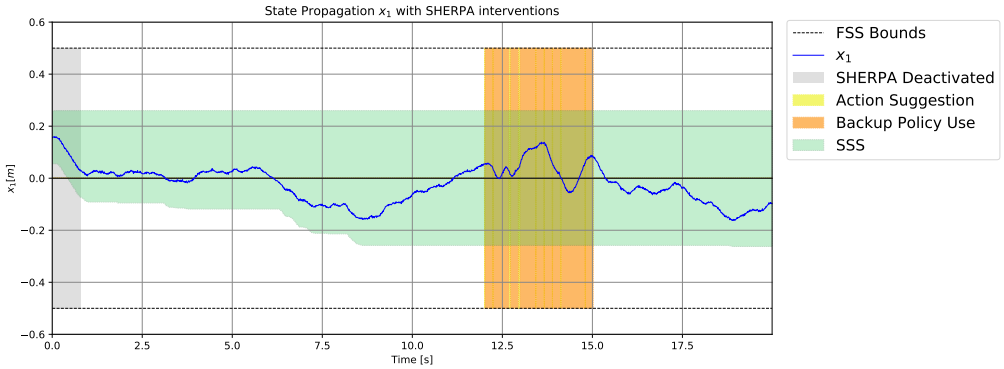
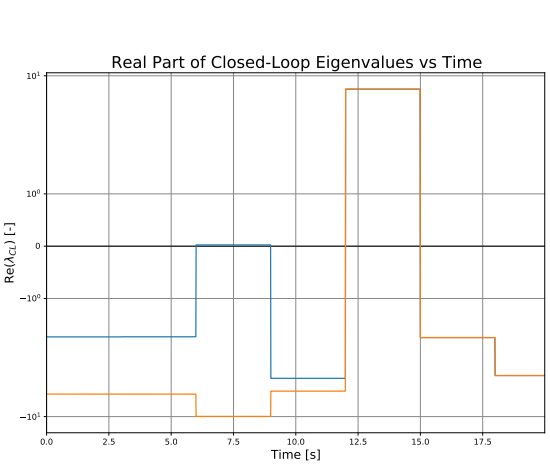


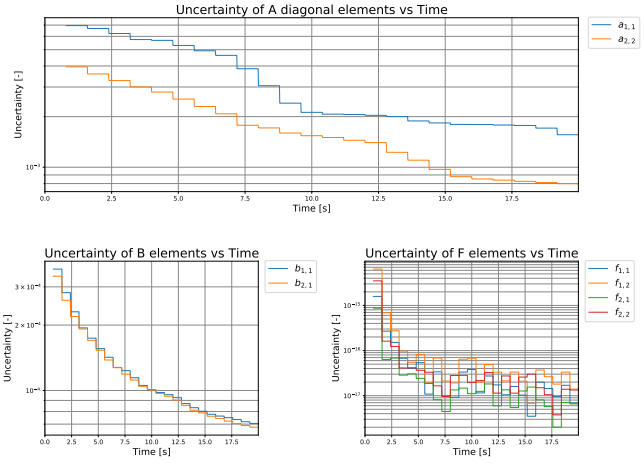
Figure 8.12: Evolution of the SSS over time along with SHERPA interventions for the position of the first mass, x_1; Online VI; Safe Curriculum Learning (step 1); SHERPA activated; Gaussian sensor noise, $\sigma = 0.002$.

A second figure, Figure 8.12, shows the activity of SHERPA throughout the learning process. It should be noted that for the first 0.8 seconds, SHERPA is not active, since no uncertain model is available yet. During this time, the system identification module is collection samples to compute a first estimate of the state space system. As such, SHERPA can not search for backup policies. However, while waiting for an uncertain model, SHERPA simply observes the state-action pairs provided by the agent. It gives the opportunity to update the SSS. The evolving SSS can be seen in Figure 8.12 as the green area. Even during its inactivity, it is shown that the SSS evolves. Last but not least, the safety filter intervenes in the time interval $t = [12, 15]$, corresponding to an unstable policy update, as can be deduced from the positive closed-loop eigenvalues in Figure 8.13a. From Figure 8.12, it can be deduced that mode 2 is used in intervals until a new policy update is performed by the agent. Note, however, that SHERPA does not use the backup policy continuously throughout the time interval. Instead, it follows it for 0.2 seconds, before letting the agent take back the reins. Due to the unstable policy of the agent at that time, the agent quickly pushes the system back towards the FSS. Which is when SHERPA is activated again, and uses its backup policy. In the simulation used as an example so far, the time between using backup policies is generally 0.05 seconds, showing how unstable the agent's policy is.

In addition, it is important to understand how the parametric uncertainty of the model estimate used within SHERPA evolves over time. As shown in Appendix A, the magnitude of the uncertainty has a great impact on the evolution of a state's uncertainty bounds when propagated through an uncertain state space system over time. Important to note, is the assumption that the state and action that are given to SHERPA have an initial uncertainty of 0. In Figure 8.13b, the evolution of the parametric absolute uncertainty of the model used by SHERPA is shown. This image is repeated in Figure D.7 for better readability. From the figure it can be seen that the uncertainty drops as time passes, which is to be expected as more data becomes available. Due to the measurement noise, the uncertainty is around $8e{-}2$, and drops below $1e{-}3$ near the end for the diagonal elements of $\tilde{A}$. The additional assumption that was made in this implementation is that the reference is deterministic, which explains the relatively lower uncertainties of the elements in $\tilde{F}$. These oscillate due to their low magnitude.



(a) Evolution of the real part of the 1-MSD system's closed-loop eigenvalues; Online VI; Safe Curriculum Learning (step 1); SHERPA activated; Gaussian sensor noise, $\sigma = 0.002$.



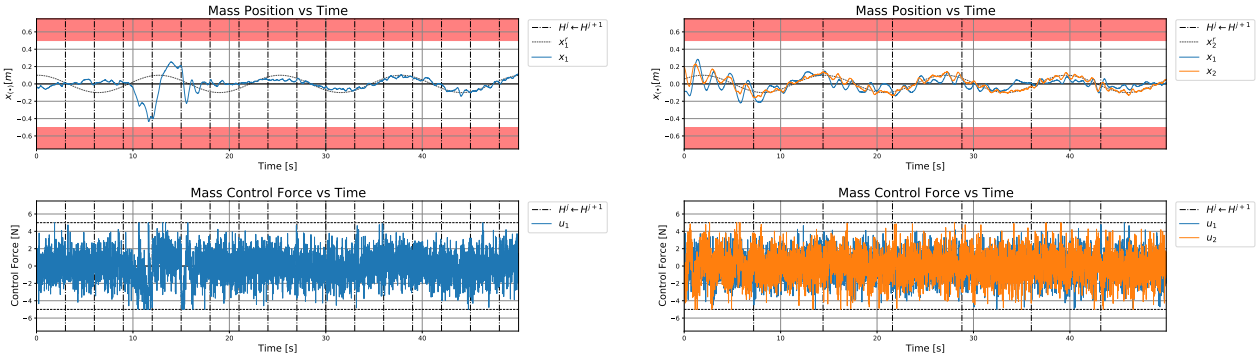
(b) Evolution of the absolute parametric uncertainty of all matrices forming the uncertain model estimate used in SHERPA, $\tilde{A}$, $\tilde{B}$ and $\tilde{F}$; Online VI; Safe Curriculum Learning (step 1); SHERPA activated; Gaussian sensor noise, $\sigma = 0.002$; figure repeated in Figure D.7, for better readability.

Figure 8.13: Evolution of closed-loop eigenvalues and absolute parametric uncertainty over time; Online VI; Safe Curriculum Learning (step 1); SHERPA activated; Gaussian sensor noise, $\sigma = 0.002$.

So far, the safety filter has proven to have a positive impact on the learning process. Indeed, keeping the system within safety even though the agent's policy is unstable. However, SHERPA specifically, requires drastically more computation power. Since at every time step a number of attempts are made to find a backup policy, which in itself projects 10 different policies over 20 time steps. It has been found that the wall clock time for the simulation increases by a factor of 21. Running the implementation with safety filter averages at a simulation time of 150 seconds for the first curricular step. Whereas running the same simulation without safety filter results in an average run time of 7 seconds. Both averages were found after running 200 simulations for each configuration.

Although the results so far are promising, the safety filter has not yet been implemented on a full curriculum, such as the one presented in Section 8.3. Completing this section, two examples of a 3-MSD curriculum with measurement noise and SHERPA as safety filter are presented. The first resembles the curriculum learning behaviour outlined in Section 8.3, whereas in the second solely the last curricular step is discussed, where attention is brought to a particular behaviour of SHERPA.

Firstly, a set of nine curricula was run, with each 3 curricular steps and measurement noise added onto the state. The Gaussian noise's standard deviation was lowered slightly to $\sigma = 0.0012$, in order to fasten the policy search process within SHERPA. When the noise amplitude is lower, the system identification estimates have slightly lower parametric uncertainties. Leading to a slimmer uncertainty bound of the projected state propagation when searching for backup policies. Although the measurement noise is lower, it still impacts the learning process. Even with SHERPA active on the system, sometimes the policy is unstable to such an extent that a backup policy can not change the system's trajectory sufficiently for the latter to stay out of the FSS. From the nine curricula, a total of 7 find a stabilising policy for a 3-MSD system. However, 100% of the simulations found stabilising policies up to the second curricular step.



(a) Evolution of mass position over time for a 1-MSD system; Online VI; Safe Curriculum Learning (step 1); SHERPA activated; Gaussian sensor noise, $\sigma = 0.0012$; $x_i \in [-0.2, 0.2]$; $\dot{x}_i \in [-0.25, 0.25]$; Red is FSS for x_1.

(b) Evolution of mass positions over time for a 2-MSD system; Online VI; Safe Curriculum Learning (step 2); SHERPA activated; Gaussian sensor noise, $\sigma = 0.0012$; $x_i \in [-0.2, 0.2]$; $\dot{x}_i \in [-0.25, 0.25]$; Red is FSS for x_2.

Figure 8.14: Evolution of mass position over time; Online VI; Safe Curriculum Learning (steps 1 and 2); SHERPA activated; Gaussian noise, $\sigma = 0.0012$. Enlarged images of both sub-figures are repeated in Figures D.8 and D.10 for better readability.

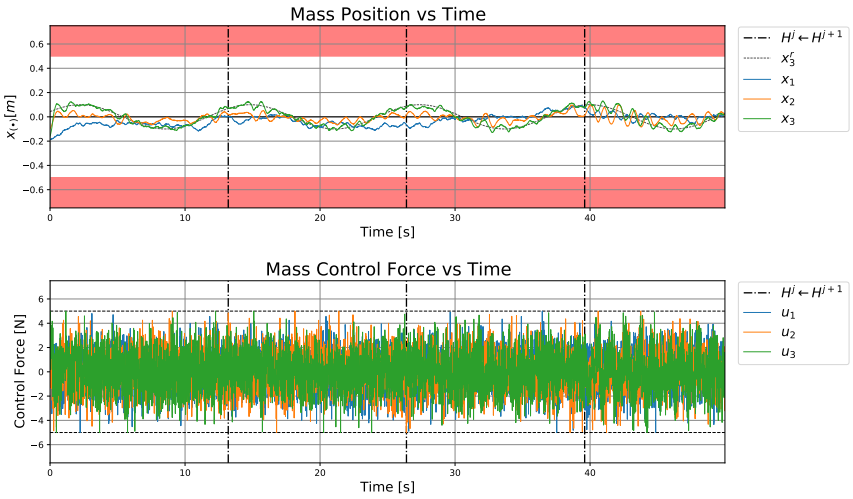
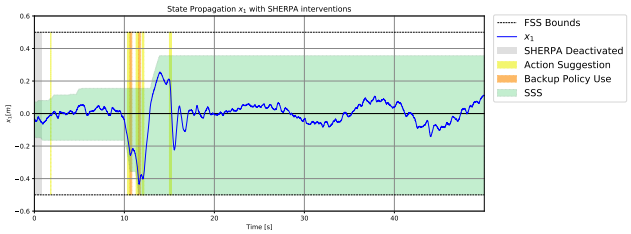
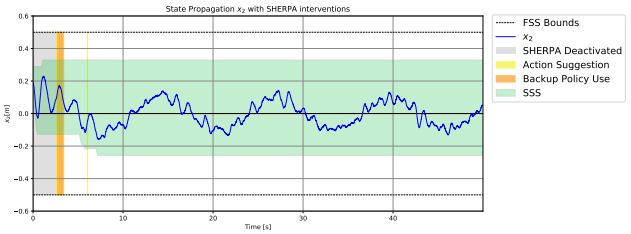


Figure 8.15: Evolution of mass positions over time for a 3-MSD system; Online VI; Safe Curriculum Learning (step 3); SHERPA activated; Gaussian sensor noise, $\sigma = 0.0012$; $x_i \in [-0.2, 0.2]$; $\dot{x}_i \in [-0.25, 0.25]$; Red is FSS for x_3.

In the above figures, one of the seven successful curricula is shown. The position states of the respective masses of a 1-MSD, 2-MSD and 3-MSD are shown in Figures 8.14a, 8.14b and 8.15, respectively. The first two figures are repeated in Appendix D.3, as Figures D.8 and D.10 for better readability. Additionally, the interventions of SHERPA during curricular steps 1, 2 and 3 are shown in Figures 8.16a and 8.16b and 8.17, respectively. For sake of overview the first two figures are shown in a smaller form factor in this chapter. However they have been repeated in Figures D.9 and D.11 for better readability.
The impact of SHERPA on unstable learning behaviour can be seen in Figure 8.16a, where action suggestions and backup policies are used to change the trajectory of the system. At around 10 seconds into the simulation the agent performs a policy update, which is unstable. The consequences on the system's state propagation can be seen in Figure 8.14a, where the mass position suddenly drops towards the lower bound of the FSS. Once SHERPA has brought the system back to a safer state, the agent is able to learn a stabilising policy. The subsequent curricular step sees interventions of SHERPA at the beginning of the learning process. This is linked to two aspects, one tied to the agent and one tied to SHERPA. The first reason for interventions is that even though mapping was performed between the curricular steps, this mapping is not perfect. As such, it is expected that during the first few seconds of a curricular step, the agent's policy is not controlling the newly added state elements correctly. A second aspect is the fact that initially, the SSS is relatively small. Leading to the policy search within SHERPA not finding policies as easily. As for the third curricular step, it sees similar behaviour to the second curricular step. In the sense that SHERPA intervenes majoritarily in the first few seconds of the simulation. Furthermore, the safety filter is activated near the end of the simulation, where it correctly projects that the system would leave the SSS if kept on the current course. Giving a closer look enables to see that when backup policies are activated, the slope of the position evolution is steep.



(a) Evolution of the SSS over time along with SHERPA interventions for the position of the first mass, x_1; Online VI; Safe Curriculum Learning (step 1); SHERPA activated; Gaussian sensor noise, $\sigma = 0.0012$.



(b) Evolution of the SSS over time along with SHERPA interventions for the position of the second mass, x_2; Online VI; Safe Curriculum Learning (step 2); SHERPA activated; Gaussian sensor noise, $\sigma = 0.0012$.

Figure 8.16: Evolution of the SSS over time along with SHERPA interventions for the position of the first and second mass, x_1, x_2, on the left and the right, respectively; Online VI; Safe Curriculum Learning (steps 1 and 2); SHERPA activated; Gaussian sensor noise, $\sigma = 0.0012$; Enlarged images of both sub-figures are repeated in Figures D.9 and D.11 for better readability.

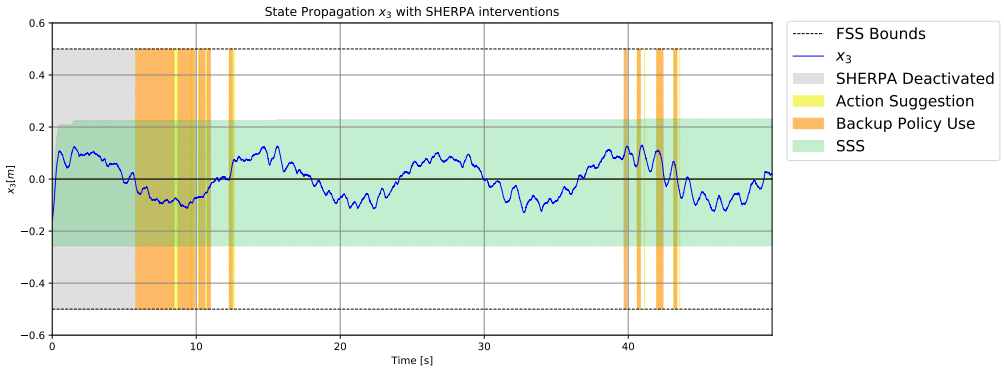
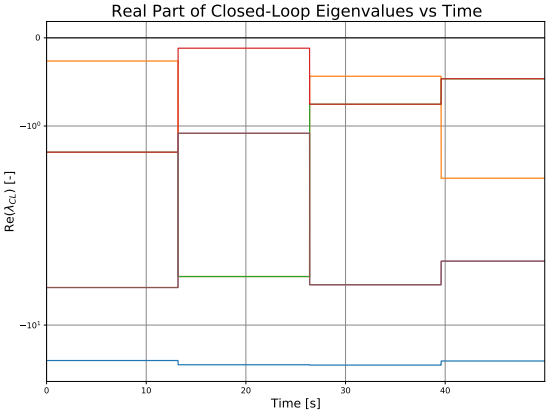
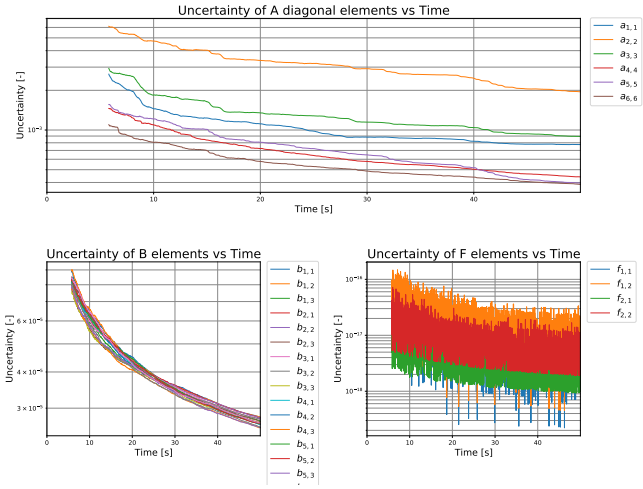


Figure 8.17: Evolution of the SSS over time along with SHERPA interventions for the position of the third mass, x_3; Online VI; Safe Curriculum Learning (step 3); SHERPA activated; Gaussian sensor noise, $\sigma = 0.0012$.

Ultimately, the agent finds a stabilising policy, which is seen in the closed-loop eigenvalues of the 3-MSD system presented in Figure 8.18a. All the real parts are negative which means a stable closed-loop control is reached. The eigenvalues of other curricular steps are not shown in this report, since their evolution is not of importance to the results outlined here. Finally, the evolution of the uncertain model's absolute parametric uncertainty in the third curricular step, is given in Figure 8.18b, repeated in Appendix D.3 for better readability. Due to the more complex nature of the tasks, the system identification module updates its estimate at each time step. It can be seen that all the uncertainties diminish over time, which provides SHERPA with a better model to find backup policies.



(a) Evolution of the real part of the the 3-MSD system's closed-loop eigenvalues; Online VI; Safe Curriculum Learning (step 3); SHERPA activated; Gaussian sensor noise, $\sigma = 0.0012$.



(b) Evolution of the absolute parametric uncertainty of all matrices forming the uncertain model estimate used in SHERPA, $\tilde{A}$, $\tilde{B}$ and $\tilde{F}$; Online VI; Safe Curriculum Learning (step 3); SHERPA activated; Gaussian sensor noise, $\sigma = 0.0012$; Figure repeated in Figure D.12, for better readability.

Figure 8.18: Evolution of closed-loop eigenvalues and absolute parametric uncertainty over time; Online VI; Safe Curriculum Learning (step 3); SHERPA activated; Gaussian noise, $\sigma = 0.0012$.

Following the 9 simulations that were done for the statistics of the impact SHERPA has on a curriculum, one of the results exhibited a behaviour true to SHERPA's RSS. In Figure 8.19 it can be seen that the final policy of the agent for a 3-MSD system is unstable. What is noticeable, however, is that the last mass is close to the reference signal. From the perspective of the safety filter, this example is performing well. This is due to the definition of the RSS used in this implementation. The safety filter only aims at keeping the last mass outside of the FSS, without caring about the behaviour of other masses.

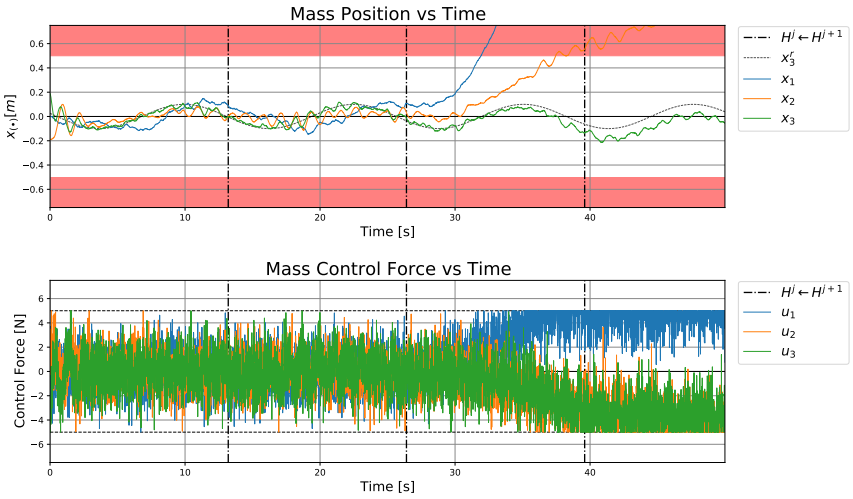


Figure 8.19: Evolution of mass positions over time for a 3-MSD system; Online VI; Safe Curriculum Learning (step 3); SHERPA activated; Gaussian sensor noise, $\sigma = 0.0012$; $x_i \in [-0.2, 0.2]$; $\dot{x}_i \in [-0.25, 0.25]$; Red is FSS for x_3.

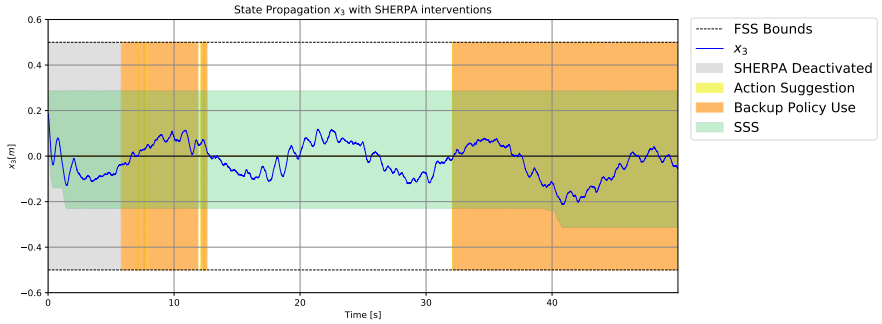


Figure 8.20: Evolution of the SSS over time along with SHERPA interventions for the position of the third mass, x_3; Online VI; Safe Curriculum Learning (step 3); SHERPA activated; Gaussian sensor noise, $\sigma = 0.0012$.

Figure 8.20 provides a better insight into the activity within SHERPA. It can be seen that the last mass is entirely controlled by the safety filter after the second policy update. In the case of this implementation, this proves to be a flaw and it is recommended to provide SHERPA with an RSS that is more complete. It is expected that the policy search will take longer, since the policy is intrinsically faced with more constraints.

It can be concluded, that although including a safety filter such as SHERPA results in longer learning times, the benefits are clearly visible. Indeed, SHERPA has proven time and again that when the agent provides an unstable policy, it will take over control of the system for as long as it takes the agent to find a better policy. Coming with a big asterisk, SHERPA's control is entirely dictated by the definition of the RSS. Only the states part of this space are controlled sensibly in by the safety filter. The others are still controlled, regardless of their safety aspects.

8.5. Hyperparameter Sensitivity

Having determined that online VI in combination with curriculum learning and a safety filter results in finding stabilising policies in most of cases, it is of interest to understand the sensitivity of the learning convergence on hyperparameters. This section is split into the different modules used in this analysis. First the hyperparameters of the core LQT task are analysed. In a second instance, the focus is laid on the safety algorithm used in this analysis, namely SHERPA. The latter includes the hyperparameters for system identification, since the model that is estimated is used in SHERPA solely.

8.5.1. Linear Quadratic Tracking Hyperparameters

First, insights are provided with regard to the amplitude of the persistence of excitation signal. In previous cases, this signal was defined as white noise with an amplitude of 1.5N. If the amplitude is lowered, the agent is not able to update the kernel matrix using OLS, due to the precision matrix ($P^T P$, from Section 6.3.1) not being invertible. The reason being that when the persistence of excitation amplitude is not high enough, the rows of the regression matrix P become linearly dependent, making $|P|$ tend to zero. Due to the agent failing to update, no plots could be generated as the implementation crashed every time. On the other hand, when increasing PE amplitude, the agent learns slower, but the policy is more consistently updated. In essence, the gain parameters within K_1 are not shifting as much. Instead, for a PE amplitude of 4.5N, the first value in K_1 starts at 26.5 and converges to 25.1 over 13 kernel updates. It should be noted, however, that having amplitudes that are too high, will overwhelm the agent, and result in no learning at all. The use of the OLS method will still generate kernel estimates. However, these estimates result in policies that are diverging from the optimal policy.

In a second instance, the sensitivity with respect to the number of collected samples for a policy evaluation is analysed. In the example shown in figures 8.8 and 8.9, the sample factor is set to 5. Setting lower sample factors results in the precision matrix's determinant, $|P|$, to tend to zero. On the other hand increasing the aforementioned factor, leads to an increase of $|P|$. However, it comes at the price of not having an update for a longer time. Should the system be highly unstable, it could have catastrophic consequences to have a non-optimal policy for a longer time. Indeed, if the safety aspect is considered, the state might get out of the safe state space by the time a new policy evaluation and policy improvement are performed.

The agent's learning process is highly dependent on the cost function of the said process. In the case of LQT, the cost function given in Equation (3.12) is inherently defined by the diagonal values of weighting matrices Q and R. Both matrices define the energy in the error from the reference state and the energy in the input, respectively. Changing their values has an effect on the learning objective as a whole. Additionally, their ratio defines the focus of the optimisation. Indeed, if the focus is laid on tracking the reference state at all costs, then the values on the diagonal of Q are set high and the values of R are set low. On the other hand if the limiting factor in the applied case is input force, then focus has to be set on keeping the energy in the input low. For the latter case the values of R have to be increased. The importance of these values will determine the feasibility of the learning process. For instance, setting the values of Q too high, will produce kernel matrix updates that are too aggressive. It can be compared to the phenomenon known in the Machine Learning (ML) community as overfitting. In contrast, setting the values of Q too low causes the opposite behaviour, resembling underfitting. The same is true for the values of R. Where setting them too low, results in the agent taking abnormally large actions to attach its state propagation to the reference. Setting them too high has as consequence that low actions are taken. Which makes the agent follow the reference, indeed. However, the error between the reference and the agent's state reduces relatively slowly. Resulting in longer learning times. As such, choosing these values and their comparative ratio defines the focus and the aggressiveness of the learning process.

In order to make the task more complicated, the agent is conditioned to saturated inputs. Not only is this closer to reality, where infinite force is impossible, it also forces the agent to manage its resources during the learning process. The action saturation bounds have an impact on the learning behaviour, as was discovered during the flat learning case presented in Section 8.2. During the sampling time frame (between two kernel updates), a relationship is sought between the cost and the state-action pair. However, when actions are continuously saturated, this relationship is skewed, as no persistence of excitation is present. Defining an action saturation interval that is too small makes it more probable that the action will be saturated during sampling. Therefore, increasing the chances of a bad kernel update due to a skewed state-action-cost relationship. Larger action saturation intervals solve this problem. However, they allow for more aggressive inputs to be taken, which in case of highly non-optimal policies can lead to overshooting the reference. If safety is considered as well, larger action saturation intervals allow for action to be taken that lead to the FSS quicker. Leaving the safety filter less opportunity to correct the course of the system to stay within the SSS.

Finally, the impact of the reference signal is inspected. In the preliminary analysis, solely a sinusoidal signal and its according parameters are considered. Sine waves have three defining parameters, namely amplitude, frequency and phase. Empirically, it was discovered that the amplitude and frequency are highly related. Based on the system's configuration and the action saturation, there is a limiting factor in the signal the system can physically track. Taking the extreme example of extremely high amplitude and extremely high frequency, the system would require close to infinite input force to allow such tracking. As such, either amplitude or frequency has to be high and the other low, for the tracking task to be feasible. The third parameter, phase, is used to shift the reference signal. Which can be seen as changing the initial reference state.

8.5.2. Safety Hyperparameters

The safety filter is defined by a set of hyperparameters defining its performance. This section includes the hyperparameters of the system identification module as well, since the integration of this module is entirely used for defining the uncertain model used in SHERPA. First the hyperparameters of SHERPA and their impact on learning performance are discussed, after which the hyperparameters of the system identification module are discussed.

First, a set of policy search's iteration parameters are discussed. Starting with the number of attempts SHERPA is given to successfully find a backup policy at each time step. In the implementation presented in Section 8.4, this parameter is set to 10. Meaning that at each time step, ten attempts are made to find a backup policy using the policy search processes given in Section 8.4.1. Increasing this iteration parameter raises the likelihood of finding a backup policy. However, with the main drawback that it increases the run time of the simulation.
Going deeper into the SHERPA processes, the policy search iteration loops are dictated by two variables. One is the number of policies that are checked, and the second is the number of time steps for which the uncertain state propagation is projected with each policy. In alignment with the previous hyper parameter, increasing the number of policies to be checked augments the probability of finding a backup policy, whilst demanding more computational power. Decreasing this hyperparameter has the opposite consequences than increasing it. Lower probability of finding a backup policy and lower computational power requirement, that is.
The number of time steps for which uncertain state propagation is performed in each policy is defined as 20 in this implementation. Increasing it, means that the uncertainty interval of the state at the last time step in the projection will be wider. The width of this interval is constrained by the condition that it has to be lower than twice the closeness condition $\delta_{(\bullet)}$, for each state element in the RSS, respectively. As such increasing the number of steps too much results in an uncertainty interval that is too large to satisfy the closeness condition. Since the uncertainty interval width is also dependent on the model's parametric uncertainties, the number of time steps used in the projection is indirectly related to this property as well. More on the model's parametric uncertainties in a subsequent paragraph. In the opposite case, where the number of time steps were to be decreased, the policy is less likely to get close to previously visited states. As such, rendering the search of a backup policy more difficult. Furthermore, when resorting to a backup policy, this parameter defines the number of time steps during which the backup policy is followed. Consequently, lowering it comes at the cost of SHERPA not being granted enough time to adjust the system's trajectory enough to lead it away from the FSS. Like the previous iteration parameters, when this hyperparameter is increased, it requires more computational power, and the opposite is true when decreasing the parameter.

Regarding the risk function used in SHERPA, the sensor reach plays an important role. Its sole impact is on the evolution of the SSS. Since the SSS bounds are increased with the sum of the state and the sensor reach, the magnitude of the reach defines how fast the SSS can grow at each time step. Increasing the sensor reach provides a faster exploration of the SSS. However, when having a larger reach, the state is considered risky more quickly. Indeed, adding the reach to the state makes it easier to end up in the FSS. In the opposite case, where a lower sensor reach is taken, the exploration of the SSS is more gradual, and slower.

Next the FSS bounds and the RSS are discussed. Since the SSS reaches its maximal size at the boundary with the FSS, the larger the FSS, the slimmer the SSS can be. When a large FSS is chosen, the constraints are much more stringent on the learning process of the agent. It is more difficult to find backup policies that do not lead the agent into the FSS. Closely, related to the previous hyperparameter, is the definition of the RSS. The RSS defines which state elements are to be tracked by the safety filter. When more state elements are included in the RSS, it increases the difficulty of finding a backup policy. It means that more state elements have to stay within safety bounds. However, limiting the state elements comprised within the RSS, leads to results shown in Figure 8.19, where only the state element tracked by SHERPA stays within bounds. The other elements shoot out and provoke an unstable policy update from the agent.

Finalising the hyperparameters of SHERPA is the closeness bound, δ. This hyperparameter's influence is most prominent in the closeness condition. It's element wise value defines the size of an area around the states previously visited by the agent. The closeness bound is defined element wise to provide a more tractable task conditioning. Furthermore, the projected uncertain state has to lie entirely within the aforementioned area to satisfy the closeness condition. As such, increasing δ increases the probability that the closeness condition will be satisfied. Although this facilitates the search of a backup policy, its quality is diminished. When δ is high, the ergodicity condition is more softly enforced, due to the propagated uncertain state having a larger interval to satisfy. Guarantees of SHERPA, which rely strongly on the ergodicity condition, become less certain. Decreasing δ results in a more difficult policy search. However, the backup policies that are found, are relatively better and provide a more consistent guarantee of staying outside of the FSS.

To conclude this subsection, the hyperparameter dedicated to the system identification module is the number of samples collected for an update of the state space model estimate. In the simulations presented in Section 8.4, the sample factor is selected to be 9. The hyperparameter's magnitude is defined based on the measurement noise and the persistence of excitation signal magnitudes, which are discussed in more detail in the previous subsection. When measurement noise and PE magnitudes are relatively high, the sample factor of the system identification module needs to be high as well. An additional concept is the sliding window. In essence, the number of samples is set to a given value for the first update and after that, at every time step a new estimate is computed whilst keeping older samples. The impact of using the sliding window concept, is the faster availability to more accurate (i.e. less uncertain) model estimates for SHERPA. This facilitates the policy search.

In conclusion, this section has analysed the impact all hyperparameters of this implementation have on the learning process aspects such as learning stability, learning convergence, safety and run time. Providing a better insight for new implementation when hyperparameters are yet to be determined. The search for such parameters is generally found to be tricky, due to the lack of theoretical or mathematical backing for the optimal values.

8.6. Preliminary Analysis Conclusion

The role of the preliminary analysis, is to verify and validate the implementation before targeting more complex systems. In this chapter a thorough analysis is performed on a Mass-Spring-Damper (MSD) system with unstable dynamics. It is proven that a policy can be found to force the system into a tracking task in a safe manner.

In a first instance, three types of learning have been applied to the system, each with different results. First online flat learning was simulated with relatively bad statistics. Indeed, online flat learning has 0% of convergence to an optimal policy. Clearly, the system is too complex for the agent to find a stabilising policy quickly enough, before the system reaches a point where the agent is no longer able to control it due to imposed input force saturation.
As a consequence, an offline implementation of flat learning was trialed with an equal cumulative training time as the online implementation. The former gave much better results. In this case, about 96% of cases converged to an optimal policy. The main issue with the offline process, is the rather unstable learning behaviour of the system throughout the offline episodes. The behaviour is attributed to the fact that the unstable system is too complex for the agent to form a stabilising policy quickly enough to avoid unstable system behaviour. However, this time, the found policy is good enough to keep the system within controlling bounds of the agent.
Finally, the agent is given a curriculum, based on increasing task complexity. Each curricular step adds a mass to the system. The results are much better, not only with regard to the amount of cases that converged to a stabilising policy (100% of cases did), but also in the stability of the learning process. For transferring knowledge between curricular steps, a simple semantic mapping was used. After each curricular step the system seems to be fulfilling the Linear Quadratic Tracking (LQT) task in a stable manner. Control dynamics for which a policy was found in a previous curricular step were correctly used by the agent in the new curricular step. Allowing the agent to focus on controlling unseen components of the system.

In order to complete the scope of this research, which includes the consideration of the learning process' safety, a safety filter is added to the implementation along with measurement noise. Due to the already safe learning behaviour when pacing the agent through a curriculum, measurement noise was added to force the agent outside to this safe comfort zone, and as such provide a better insight on the use cases for a safety filter. Although the learning process appears safe with curriculum learning, no guaranties are provided with respect to safety. With the safety filter module included in the implementation, the simulation run time is found to take a hit, more specifically by a factor of 21. Additionally, the learning convergence towards a stabilising policy is reduced. On the one hand due to the addition of measurement noise, which increases the chance of unstable policy updates, and in a second instance due to the RSS being defined such that SHERPA only tracks the position of the last mass to stay out of the FSS. Resulting in some simulations complying with a safe learning process, whilst having an unstable policy.

The preliminary analysis serves as an empirical proof of concept for the algorithm proposed in this report. Once having demonstrated that combining the theoretical principles uncovered in the literature study (Part II of this report), the algorithm is to be used on a more advanced test setup constituted of a quadcopter. The empirical results formulated in this chapter measure the impact of a curricular approach on a flat reinforcement learning algorithm, which advances this research towards its objective outlined in the introduction of this technical report.

IV

Final Analysis: Additional Results

9

Quadcopter Modelling

This chapter contains a short derivation of the linearised quadrotor model used in the final analysis (in the article provided in Part I) as well as the derivation of the ξ mapping matrix provided in the paper. The latter is configuration dependent (either ”x” or ”+”). Here the derivation is made for the *x*-configuration.

9.1. Linear Quadrotor Model

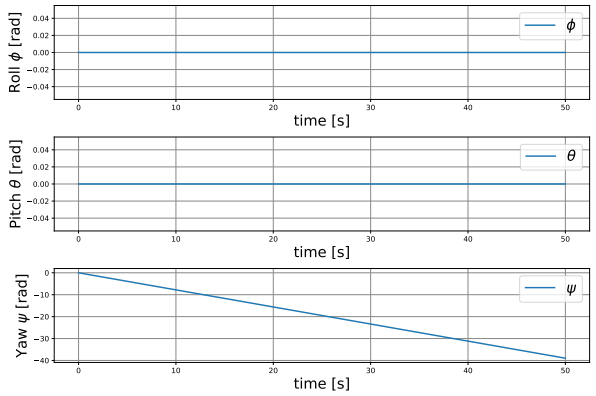
The linearised quadrotor model used in the final analysis of this thesis was based on the model derived in [45]. In [45], the resultant linear continuous-time model is defined as shown in Equation (9.2). The according state and action vectors are given in Equation (9.1).

$$\begin{aligned} x &= \begin{bmatrix} \phi & \theta & \psi & p & q & r & u & v & w & x & y & z \end{bmatrix}^T \\ u &= \begin{bmatrix} f_t & \tau_x & \tau_y & \tau_z \end{bmatrix}^T \end{aligned} \tag{9.1}$$

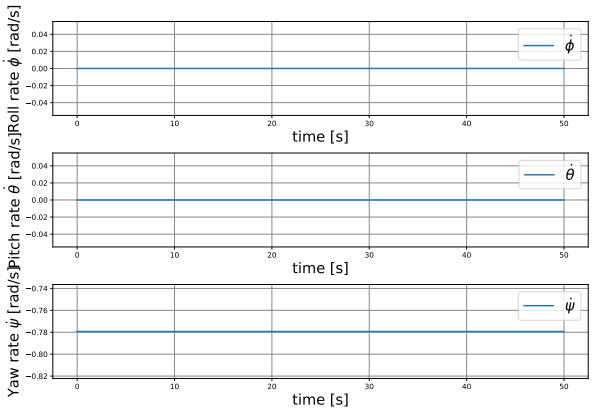
$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{1}{I_{xx}} & 0 & 0 \\ 0 & 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & 0 & \frac{1}{I_{zz}} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{-1}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{9.2}$$

Notably, the derivation in [45] makes assumptions to simplify and linearise the model. The model is linearised around the hovering trimming point. Thus, all inputs are relative to hover conditions. Furthermore, the small angle approximation is made. The latter allows the equality $[p, q, r] = [\dot{\phi}, \dot{\theta}, \dot{\psi}]$ to hold. To convert the model from continuous-time to discrete time, the `scipy.signal.cont2discr()` function can be used, which defaults to the zero-order hold method for state space discretisation.

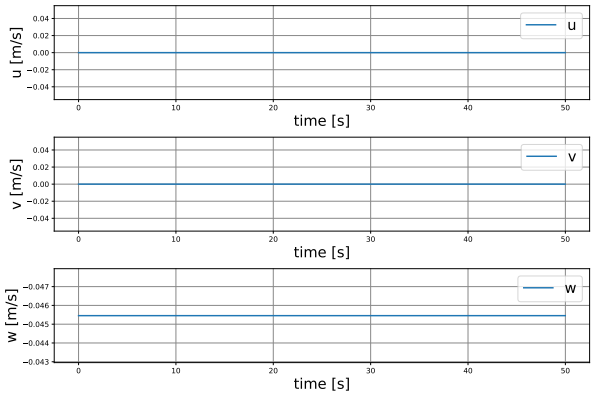
The model was verified by giving impulse inputs in all dimensions and ensuring that only the states tied to that specific input were impacted. The magnitude of the response was, however, not validated with real quadrotor data. This verification process has been performed on all four input individually as well as a combination of input pairs. However, of the sake of clarity, only the responses for the thrust force and yaw torque pair ($u_{init} = [-10, 0, 0, -10]^T$) is given in Figure 9.1.



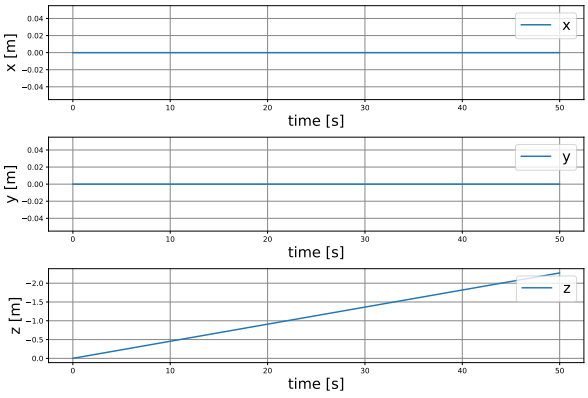
(a) Model responses of attitude angles ϕ, θ and ψ with impulse input on thrust force and yaw torque (f_t and τ_z, respectively).



(b) Model responses of attitude angles $\dot{\phi}, \dot{\theta}$ and $\dot{\psi}$ with impulse input on thrust force and yaw torque (f_t and τ_z, respectively).



(c) Model responses of attitude angles u, v and w with impulse input on thrust force and yaw torque (f_t and τ_z, respectively).



(d) Model responses of attitude angles x, y and z with impulse input on thrust force and yaw torque (f_t and τ_z, respectively).

Figure 9.1: Linearised discrete-time quadrotor time response whit initial impulse input on thrust force and yaw torque (f_t and τ_z, respectively).

From the figures above, it can be seen that when giving an impulse on the vertical DOF with the thrust force, the vertical velocity is shifted and the the quadrotor climbs in altitude. Additionally, an impulse input is given in around the z-axis, with τ_z. Consequently, the yaw rate is shifted and the yaw angle monotonically increases. Notably, both the pitch and the roll dimensions are not affected by these inputs.

9.2. Quadrotor Configuration Implementation

The state space model defined by A and B (Equation (9.2)) is valid for any quadrotor, whatever the configuration ("+" or "x"). The configuration will determine the impact of each rotor on the torques τ_x, τ_y and τ_z. In this document the "x" configuration is considered, as is shown in Figure 9.2. Using the free body diagrams in this figure, the input elements can be translated as given in ...

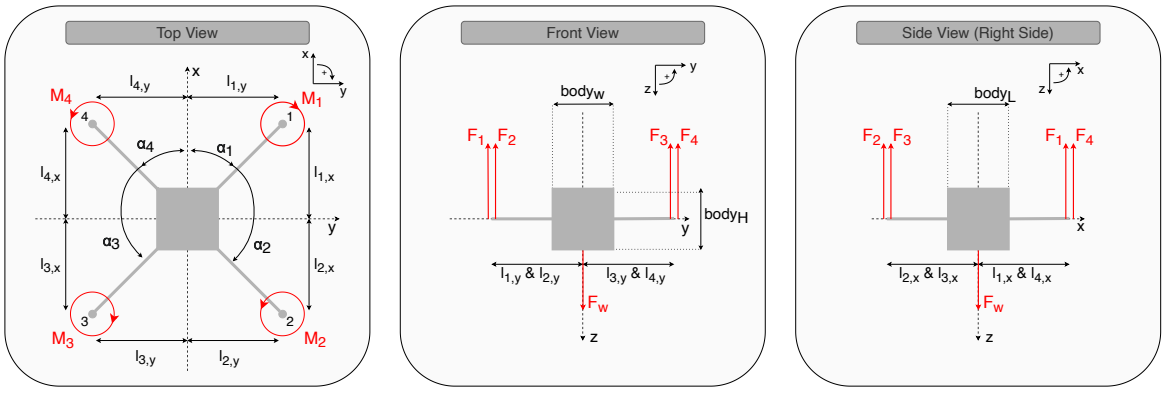


Figure 9.2: Views of the quadcopter forces and other important dimensions.

$$\begin{cases} f_t &= F_1 + F_2 + F_3 + F_4 \\ \tau_x &= -F_1 l_{1,y} - F_2 l_{2,y} + F_3 l_{3,y} + F_4 l_{4,y} \\ \tau_y &= F_1 l_{1,x} - F_2 l_{2,x} - F_3 l_{3,x} + F_4 l_{4,x} \\ \tau_z &= M_1 - M_2 + M_3 - M_4 \end{cases} \tag{9.3}$$

Instead of having f_t, τ_x, τ_y and τ_z as input elements, it is desired to have everything in terms of rotor RPM. As such, a mapping is required: $\begin{bmatrix} f_t & \tau_x & \tau_y & \tau_z \end{bmatrix}^T \rightarrow \begin{bmatrix} \Omega_1 & \Omega_2 & \Omega_3 & \Omega_4 \end{bmatrix}^T$. For such mapping, a linear relation is assumed between the thrust force of a rotor and its RPM, as well as a linear relationship between the torque moment produced by a rotor and the rotor's RPM. These relationships are quadratic in nature, but will be assumed linear in order to fit them within a linear state space structure. In order to be realistic, the linear factors, (b, d), mapping the force and torque moment to the RPM, respectively, are derived from performance data of the `PER3_11x4` propellor[1]. The thrust and torque values at each available RPM were taken for no vertical wind speed, $V = 0$. This gives the relations shown in Figure 9.3. In orange, the linear approximation is shown. With zero thrust and zero torque at zero RPM. From the respective legend boxes of these plots, the mapping factors can be found to be $b = 4.08732 \cdot 10^{-3}$ and $d = 9.29643 \cdot 10^{-5}$, for force and torque mapping respectively

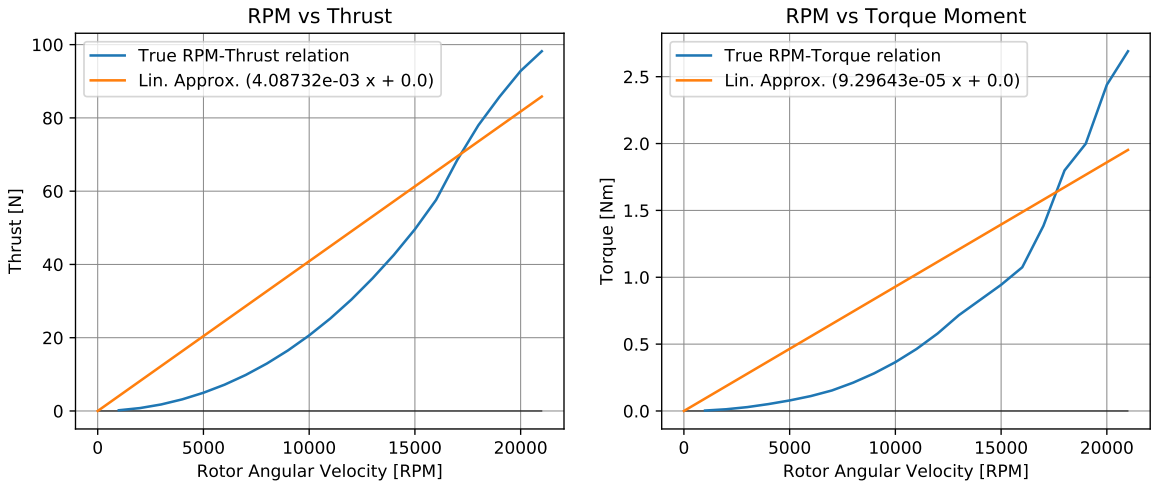


Figure 9.3: Relationship between Thrust and RPM (left) and Relationship between Torque moment and RPM (right) [rotor=`PER3_11x4`].

Substituting these relations in Equation (9.3), provides functions only dependent on the rotor angular velocities. These substitution is shown in Equation (9.4). If a the vector containing all the rotor RPMS, is defined as $\Omega = \begin{bmatrix} \Omega_1 & \Omega_2 & \Omega_3 & \Omega_4 \end{bmatrix}^T$, Equation (9.4) can be written in matrix form as shown in Equation (9.5).

$$\begin{cases} f_t &= b\Omega_1 + b\Omega_2 + b\Omega_3 + b\Omega_4 \\ \tau_x &= -b\Omega_1 l_{1,y} - b\Omega_2 l_{2,y} + b\Omega_3 l_{3,y} + b\Omega_4 l_{4,y} \\ \tau_y &= b\Omega_1 l_{1,x} - b\Omega_2 l_{2,x} - b\Omega_3 l_{3,x} + b\Omega_4 l_{4,x} \\ \tau_z &= d\Omega_1 - d\Omega_2 + d\Omega_3 - d\Omega_4 \end{cases} \tag{9.4}$$

$$\begin{cases} f_t &= \begin{bmatrix} b & b & b & b \end{bmatrix} \Omega \\ \tau_x &= \begin{bmatrix} -bl_{1,y} & -bl_{2,y} & bl_{3,y} & bl_{4,y} \end{bmatrix} \Omega \\ \tau_y &= \begin{bmatrix} bl_{1,x} & -bl_{2,x} & -bl_{3,x} & bl_{4,x} \end{bmatrix} \Omega \\ \tau_z &= \begin{bmatrix} d & -d & d & -d \end{bmatrix} \Omega \end{cases} \tag{9.5}$$

Finally, the B matrix has to be mapped to be as a function of Ω instead of $u_{init} = \begin{bmatrix} f_t & \tau_x & \tau_y & \tau_z \end{bmatrix}^T$. For this mapping, the resulting Bu multiplication should have the equality condition: $B_{u_{init}} u_{init} = B_\Omega \Omega$. The left hand side is given below:

[1]The performance data file was taken from a database developed by NASA, which can be found at `https://www.apcprop.com/v/PERFILES_WEB/listDatafiles.asp`

$$B_{u_{init}} u_{init} = \begin{bmatrix} 0 & 0 & 0 & \frac{\tau_x}{I_{xx}} & \frac{\tau_y}{I_{yy}} & \frac{\tau_z}{I_{zz}} & 0 & 0 & \frac{f_t}{m} & 0 & 0 & 0 \end{bmatrix}^T \tag{9.6}$$

In order to achieve the same result, but in terms of Ω, the new B matrix has to be as given in Equation (9.7).
Note: The signs in this matrix assume absolute values of $l_{\bullet,x}$ and $l_{\bullet,y}$.

$$B_\Omega = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ (-bl_{1,y})/I_{xx} & (-bl_{2,y})/I_{xx} & (bl_{3,y})/I_{xx} & (bl_{4,y})/I_{xx} \\ (bl_{1,x})/I_{yy} & (-bl_{2,x})/I_{yy} & (-bl_{3,x})/I_{yy} & (bl_{4,x})/I_{yy} \\ d/I_{zz} & -d/I_{zz} & d/I_{zz} & -d/I_{zz} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -b/m & -b/m & -b/m & -b/m \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{9.7}$$

9.3. Model Validation

The model presented in this chapter is limited to linearised dynamics. On a system that is inherently non-linear, such as a quadrotor, the discrepancies between the model's behaviour and that of the actual craft can be quiet large in some parts of the state space. Consequently, when using a linearised model, the experiments have to be carefully crafted as to stay within the regions of the state space where the aforementioned discrepancies are kept relatively small.

V

Discussion and Recommendations

10

Conclusion

On the highest level, the purpose of the research presented in this report is to design a system controller that can optimally track a reference. On a more specific note, however, the aim is to provide a controller that is safe and independent of knowledge about system dynamics. The direction of ML was chosen, more specifically RL, for its advantageous model-free approach. Although advanced RL algorithms such as policy gradients including methods of the lines of Trust Region Policy Optimisation (TRPO) [48] exist, the scope of the literature study was reduced to Q-learning and SARSA. The argument is made that the focus of this research lies in the combination of curricula and safe learning methods completed with system identification. Consequently, using more advanced RL techniques would lead to unnecessary implementation delays.
In order to benchmark performance of the learning process, it was opted to use a tracking task due to the numerous applications resembling such task. In cases of flight path tracking in modern aircraft for example. Mathematically, this is translated to a LQT task, which uses a Linear Quadratic (LQ) cost function in the agent-environment interaction of RL. The research discovered in the literature study on Q-learning for tracking tasks [22], founds the basis of the interaction between the agent and the environment used in both the preliminary and the final analysis outlined in Part III and Part I, respectively.
To the aforementioned RL interaction, a curricular structure is added. It involves creating a series of different entities of the interaction, where transfer learning is used to map gathered knowledge between curricular steps. Lastly, to each of the agent-environment interactions in their respective curricular steps, the goal is to add a safety filter. More specifically, SHERPA[32]. This safe learning method guarantees safety by ensuring that a backup policy is available at all times and states. The policy is validated based on the ergodic condition as well as the condition that the trajectory generated by such policy is safe. As was discovered in the literature review, SHERPA comes with a major drawback. It is reliant on external knowledge in the form of a bounding model representing the system dynamics. In order to remove this dependence, a system identification module is added to the interaction as well. It will serve the purpose of constructing an uncertain model which can be seen as a bounding model. Literature suggests three model constructs are advantageous, namely linear state space system (A,B,C,D), Artificial Neural Network (ANN)'s and multivariate splines.

The efficacy of the paradigm proposed in this thesis, has been tested through two experiments. The first experiment used a cascaded mass-spring-damper system as environment. The system was chosen for its modularity in complexity, observability, controllability and stability. The preliminary analysis in Part III, provides an insight into the learning performance and convergence benefits gained by using curriculum learning. Additionally, SHERPA manages to keep the system safe throughout the curriculum. This performance is proven to be maintained even when measurement noise is present and the model estimates provided by the system identification module are relatively uncertain. Finally, Part III is concluded with a sensitivity analysis of the hyperparameters in this architecture. The paradigm is particularly sensitive to the persistence of excitation and reference signals for the learning efficiency. Additionally, depending on the dynamic stability of the system, the severity of SHERPA's hyperparameters regarding the ergodicity condition has to be adapted to promote the quality of the backup policies.

Finally, the paradigm was tested on a quadrotor to check its applicability in primary flight control. The model was limited to four degrees of freedom, where the positions in x and y were discarded. The architecture proposed by this thesis allowed for a safe learning process of an optimal stabilising and tracking policy on the aforementioned model. The resulting policy was able to track independent reference signals on all four degrees of freedom. Furthermore it was found that the complexity of the system has a significant impact on the curricular strategies that have to be used. The quadrotor required a LQR supervisor to keep the learning process stable and convergent.

10.1. Answers to Research Questions

As part of the conclusion, this part will serve the purpose of answering the research question presented in Section 1.2.2 along with its sub-questions. Not all sub-questions can be answered based on the work found in this literature study. These will form the basis of the recommendations provided in the subsequent section for future work. First the sub-questions are addressed before tackling the main research question, since the knowledge found in the former's responses will be used to construct the latter's answer.

Sub-Research Question 1

1. How is Reinforcement Learning advantageous for complex systems?

Reinforcement Learning (RL), as presented in Chapter 2, makes use of an interaction between an environment and an agent to learn the control of a system. As was discovered during the literature study, the RL algorithms do not require a model of the system they aim to control. However, in order to tune the algorithms' hyperparameters correctly, they depend on the dimensions of the system dynamics. As such, RL algorithms depend on the complexity of a system for their hyperparameters, but do not require further information to find a controller for the system. Finding the controller in itself is independent of any system model constructs, which take up the largest amount of time when designing controllers using classical methods.

Sub-Research Question 2

2. How can a tracking task be implemented with Reinforcement Learning?

In Reinforcement Learning (RL), an agent learns a controlling policy by means of a reward function. In case of a tracking task, use is made of a Linear Quadratic (LQ) reward/cost function. Such functions represent the energy contained in the combination of a state and an input vector. Initially, the theory was founded for regulation problems or controllers. However, as recently as 2014 an adaptation has been provided to regulating controllers, in order to accommodate for tracking tasks using data-driven Q-learning. Instead of regulating the state and inputs, the error with respect to a reference signal is regulated. Indeed, when discussing tracking tasks, the purpose is to have a controller capable of following a reference signal. Using a LQ function that measures the energy of the combination of the error state and an input vector as the reward function in the RL algorithm does, consequently provide enough directions for the agent to learn appropriately. This was proven empirically in the preliminary analysis, where the convergence to an optimal policy was achieved in offline flat learning and curriculum learning.

Sub-Research Question 3

3. How is an effective curriculum for a Reinforcement Learning agent designed?

On the basis of the theory found in the literature study performed in Chapter 4, a curriculum is deemed effective when complying to two rules. Namely, a monotonically increasing statistical entropy as well as a monotonically increasing diversity of state-action pairs that can occur throughout the curriculum. Three core curricular strategies that do so, have been identified in literature: (a) reward shaping, (b) variable task complexity and (c) reversed problem space. These strategies are implemented using scrutinised task design and principles found in transfer learning.

3.1 What are the key aspects defining a curriculum?

A curriculum in Reinforcement Learning (RL) has to comply with two rules according to the literature study on the subject of curriculum learning presented in Section 4.1. A curriculum is comprised of a set of curricular steps in which the statistical entropy must monotonically increase. This means that the predictability of the state-action space needs to decrease throughout the curricular steps. Which leads to the second rule, which states that the probabilistic weight function associated with the state-action space probability of occurrences, is to increase monotonically as well. The aforementioned weight function can be interpreted as the diversity of the state-action pairs from the state-action space that can occur in the curricular step.

3.2 What are the current strategies for Reinforcement Learning curricula?

Curricular strategies are divided into three categories, each leveraging a specific part of the RL process presented in Chapter 2. Namely, (a) reward shaping, (b) variable task complexity and (c) reversed problem space. As the name suggests, when reward shaping is used, the curriculum adapts the reward function in order to ease or guide the learning process of the agent in the correct direction. Second, by changing the task complexity, the agent can be initiated on tasks of reduced complexity. Over the course of the curriculum, simple tasks are combined and/or sequenced in order to construct an intellect capable of tackling more complex tasks for the agent. Finally, there is the strategy of reversed problem space, which is based on the fundamental principle of removing the agent gradually from its goal throughout the curriculum. Indeed, in the first curricular steps, the agent is placed in the state space close to the goal state. When paced through the curriculum, the initial state in which the agent is placed is set further and further from the goal state.

3.3 How are curricula imposed on RL agents?

An agent is paced through a curriculum by two principal aspects, (a) task design and (b) transfer learning. In order to use one of the three strategies presented in the answer of the previous sub-question, the tasks forming the curricular steps have to be designed accordingly. A task is defined as a specific agent-environment set up, where depending on the strategy, an aspect of this interaction is varied between curricular steps. If the example of reward shaping is taken, each curricular step will contain an agent-environment interaction where the reward function of the environment is different from the previous curricular step. In order to comply with the rules stated in the answer of sub-question 3.1, the reward function bias, should be decreased along the curriculum. Secondly, there is the transfer of knowledge that has to occur between curricular steps. In curriculum RL, the tasks learned by the agent are related to a given degree. When combining tasks or sequencing these, the knowledge acquired should be conserved between tasks and curricular steps. Use is made of methods contained in transfer learning. Such methods consider the similarities in state and action spaces between tasks and develop ways of mapping knowledge across agent representations accordingly.

Sub-Research Question 4

4. How does safe learning impact the efficiency of curriculum learning?

In the preliminary analysis on the mass-spring-damper system, as well as the final analysis on the quadrotor in the article, it has been shown that safe learning algorithms based on risk sensitive exploration as safety aspect, such as SHERPA, impede on the learning efficiency gains made by the curriculum learning paradigms. However they do not nullify these gains. SHERPA, specifically, removes the persistence of excitation condition when imposing a new and safe action. Thus reducing the likelihood of stable policy updates by the agent over the course of a curricular step.

4.1 What does safe learning entail?

The purpose of safe learning is to tackle the most important drawback of Reinforcement Learning (RL), namely the safety aspect of the learning process. Indeed, RL is notorious for its initially erratic exploration of state and action spaces. Safety is defined in literature as an amalgam of concepts that segregate the state space in different sub-spaces. All of which are required to understand the final separation between the Safe State Space (SSS) and the Fatal State Space (FSS). As the respective names suggest, the states contained in the SSS are ones in which the agent is not harmed or destroyed, whereas the ones part of the FSS have opposite characteristics.

4.2 What are the current trends in safe learning?

A survey has been performed previously on the trends in literature. However, that survey contains a considerable amount of more classical approaches for safe learning, such as variable optimisation criterion or the teacher advice methods. The latter of which are part of the greater part of methods in safe learning which rely on providing external knowledge to the safe learning algorithm. Others in literature show that risk-oriented exploration has benefits towards increasing the safety of the learning process.

Furthermore, a deeper look has been given in the literature study to the Safety Handling with Risk Perception Algorithm (SHERPA), which acts as a safety filter on the RL agent's action. Inherently, SHERPA aims at having a backup policy at all times. This policy is used to remove the agent from fatal states when such states appear in the agent's reach.

4.3 Which, if any, safe learning methods rely (partially) on knowledge of system dynamics?

As was briefly mentioned in the answer to the previous sub-question, one of the categories defining the safe learning is one that relies on external knowledge. Anything that is not available to the agent throughout the learning process is considered to be external knowledge.
System dynamics are considered to be external knowledge. As such, it can be said that methods contained in this category are conducive to rely on knowledge of system dynamics. These are numerous in literature. Methods such as SHERPA mentioned in the answer of the previous sub-question and its related optiSHERPA method rely on a bounding model of the system dynamics. Generally, this knowledge is used to propagate the state to verify that the agent will not end up in the FSS.

Sub-Research Question 5

5. How are system dynamics modeled when they are unknown?

Different model representations serve the purpose of formulating a model estimate of the system, input and reference dynamics at hand in an specific application. In this research the scope was limited to using the linear state space representation (A, B, C, D). Since only linear systems are used in the experiments provided by this research, this structure is totally valid. The parameters within this representation are estimated using an Ordinary Least Squares (OLS) estimator, which relies on the persistence of excitation signal, used for the agent's policy update, to compute a model estimate. Besides OLS, there are other methods to estimate model parameters. For ANN's for example, backpropagation is used to correct the weights of the links between the neurons of neural network layers.

5.1 What are state-of-the-art model constructs?

In order to model complex, sometimes (highly) non-linear state spaces, advanced model representations are required. In the literature study, two such constructs were found and scrutinised. Namely Artificial Neural Networks (ANN)'s and Multivariate Splines. Both are adept at modelling highly non-linear complex system dynamics by virtue of their adaptable model resolution. On the one hand, the ANN's propagate states through the model using inherent logic in the form of interconnected neurons. On the other hand, a multivariate spline constructs an approximation of the model using simplices of which the vertices are adapted over time to fit the real system as closely as possible. Finally, if applied on linearised systems, linear state space representations can be used.

5.2 What system identification methods can be combined with Reinforcement Learning

It was found in literature that system identification methods can be added as a module onto the agent-environment interaction. Impact of linear system identification methods on the reinforcement learning process, stability and convergence have not been observed during the implementation of the two experiments in this research. However, solely linear system identification in the form of a state space representation estimated using OLS has been used in these experiments. Consequently, a conclusive statement can not be made regarding the broader spectrum of system identification methods.

Main Research Question

How can a curriculum be implemented such that it incorporates efficient learning of the policy and the system dynamics through safe exploration?

The curricular strategy that is to be implemented is case dependent and can not be generally defined. However, it has been found that by gradually increasing task complexity throughout the curriculum, a RL agent is able to find a control policy for complex tasks. In both experiments performed on the MSD system and the quadrotor in the preliminary analysis and final analysis, respectively, a tracking task was used to prove this statement. The sub-tasks need to be sufficiently easy to allow for safe learning proceedings. In the analyses provided in this research (Part I, Part III), the impact of the safety filter is negative on the learning stability. If the initial sub-task proves to be complex already, it means that the learning process is not necessarily stable. Thus, the combination of a safety filter on a relatively unstable learning process results in the agent not learning a stabilising control policy. The safety filter, by the means of SHERPA, impedes on the stability and convergence of learning due to the filter imposing new actions. These do not specifically satisfy the persistence of excitation condition required for both the agent policy update and the system model estimate. Additionally, SHERPA uses the model estimate internally to find backup policies. By removing the persistence of excitation that model becomes more uncertain and reduces the likelihood of SHERPA finding backup policies. Resulting in an unstable snowball effect. Thus the conclusion is that the curriculum has to provide a sturdy and stable learning process in each sub-task to ensure a positive learning outcome.

10.2. Recommendations & Future Work

The research presented in this paper is focused on the analysis and experimentation on time-invariant linear systems. Further research should detail the applicability of the architecture proposed in this paper to non-linear systems and time-variant linear systems. For such implementations, more advanced reinforcement learning approaches are required such as an actor-critic architecture [52]. A system identification module could be added to Pollack and Van Kampen [39] to assess the feasibility on an LQR controller, for example.

Regarding the system identification module on non-linear systems, the scope has been limited in this thesis to linear states space model representation. However, as briefly covered in the literature study, other model representations can be used. Non-linear systems, require more advanced model representations such as ANN's or multi-variate splines. Additionally, based on the observations made during the preliminary analysis with regard to sensor noise, the use of a Kalman filter may improve the learning stability in such cases.

Additionally, another limitation that became apparent in the experiments is the dependence on the persistence of excitation signals for stable and convergent learning. In the paradigm presented in this paper the placement of a safety filter between the PE signal and the state propagation in the environment poses a risk to the inclusion of such a signal. One way this could be solved, would be to define an uncertainty that is representative of the PE signal and include it into the model estimate uncertainties coming from the system identification module. Adding PE uncertainty would allow SHERPA to perform its policy projections with the inclusion of PE and return an action with excitation.

Furthermore, the safety filter used in the experiments proposed in this paper, namely SHERPA is relatively computationally expensive. Consequently, the learning process can not happen in real-time on conventional computers, and certainly not onboard. However, the Mannucci et al [32] proposes another method, optiSHERPA, which has lower computational complexities.

Finally, it should be investigated if this paradigm shows equivalent potential when applied on systems with coupled dynamics. Coupled dynamics require a different approach in curricular and safe learning strategies when a supervisor is present. This is due to the actions computed through the policy gain being partially defined by the agent and partially by the supervisor. Consequently, the supervisor has an impact on the cost function in the agent's learning process.

Bibliography

[1] M. Asadi and M. Huber. Effective control knowledge transfer through learning skill and representation hierarchies. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 2054–2059, 2007.

[2] R. Bellman. Dynamic programming. *Science*, 1966. ISSN 00368075. doi: 10.1126/science.153.3731.34.

[3] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th International Conference On Machine Learning, ICML 2009*, pages 41–48, 2009. ISBN 9781605585161.

[4] S. Chen and Y. Wei. Least-squares sarsa(lambda) algorithms for reinforcement learning. In *2008 Fourth International Conference on Natural Computation*, volume 2, pages 632–636. IEEE, 2008.

[5] P. Cichosz. Truncating temporal differences: On the efficient implementation of td(λ) for reinforcement learning. *Journal of Artificial Intelligence Research*, 2:287–318, 1995. doi: https://doi.org/10.1613/jair.135.

[6] C. de Visser. System identification of aerospace vehicles, parameter estimation. University Lecture 6, Delft University of Technology, 2019.

[7] C. de Visser. System identification of aerospace vehicles, introduction to multivariate splines. University Lecture 6, Delft University of Technology, 2019.

[8] C. de Visser and D. Pool. System identification of aerospace vehicles, introduction. University Lecture 1, Delft University of Technology, 2019.

[9] P. Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012. doi: 10.1145/2347736.2347755.

[10] F. Fernández and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the International Conference on Autonomous Agents*, volume 2006, pages 720–727, 2006. ISBN 9781595933034. doi: 10.1145/1160633.1160762.

[11] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel. Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300*, 2017.

[12] J. Garcıa and F. Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[13] J. García and F. Fernández. Probabilistic policy reuse for safe reinforcement learning. *ACM Transactions on Autonomous and Adaptive Systems*, 2019. ISSN 15564703. doi: 10.1145/3310090.

[14] P. Geibel and F. Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence Research*, 2005. ISSN 10769757. doi: 10.1613/jair.1666.

[15] A. Geramifard, J. Redding, and J.P. How. Intelligent cooperative control architecture: A framework for performance improvement using safe learning. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 72(1):83–103, 2013. doi: 10.1007/s10846-013-9826-6.

[16] C. Gnanasambandham. *State Space Model of Multiple DOF Spring-Mass-Damper System*, 2015-08-03 (accessed April 30, 2020). URL https://de.mathworks.com/matlabcentral/fileexchange/52373-state-space-model-of-multiple-dof-spring-mass-damper-system?focused=3891039&tab=function.

[17] I. Grondman, L. Busoniu, G.A.D. Lopes, and R. Babuška. 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, 2012. doi: 10.1109/TSMCC.2012.2218595.

[18] S. Haykin. *Kalman filtering and neural networks*, volume 47. John Wiley & Sons, 2004.

[19] F. S. Hillier and G. J. Lieberman. *Introduction to operations research*. McGraw-Hill Science, Engineering & Mathematics, 9th edition, 2010.

[20] C.K. Jaya, R. Sunitha, and A.T. Mathew. State estimation using Weighted Least Square method with minimum measurement data. In *2015 International Conference on Control, Communication and Computing India, ICCC 2015*, pages 275–279, 2016. ISBN 9781467373494. doi: 10.1109/ICCC.2015.7432905.

[21] John G. Kemeny and J. Laurie Snell. *Markov chains*. Springer-Verlag, New York, 1976.

[22] B. Kiumarsi, F. L. Lewis, H. Modares, A. Karimpour, and M. B. Naghibi-Sistani. Reinforcement Q-learning for optimal tracking control of linear discrete-time systems with unknown dynamics. *Automatica*, 2014. ISSN 00051098. doi: 10.1016/j.automatica.2014.02.015.

[23] B. Kiumarsi, F.L. Lewis, M.-B. 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, 2015. doi: 10.1109/TCYB.2014.2384016.

[24] H. Kour and N. Gondhi. *Machine learning techniques: A survey*, volume 46. Springer, 2020. doi: 10.1007/978-3-030-38040-3_31.

[25] B. Langmead. Markov chains. University Lecture 22, John Hopkins Whiting School of Engineering, 2013.

[26] T.-H. Lee and K.S. Narendra. Robust adaptive control of discrete-time systems using persistent excitation. *Automatica*, 24(6):781–788, 1988. doi: 10.1016/0005-1098(88)90054-4.

[27] F. L. Lewis, D. Vrabie, and V. L. Syrmos. *Optimal Control*. John Wiley & Sons, 3 edition, 2012.

[28] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[29] T.J. Ludeña Cervantes, S.H. Choi, and B.S. Kim. Flight Control Design using Incremental Nonlinear Dynamic Inversion with Fixed-lag Smoothing Estimation. *International Journal of Aeronautical and Space Sciences*, 21(4):1047–1058, 2020. doi: 10.1007/s42405-020-00273-8.

[30] T. Mannucci. *Safe Online Robust Exploration for Reinforcement Learning Control of Unmanned Aerial Vehicles*. Delft University of Technology, 2017. ISBN 9789055841745. doi: 10.4233/uuid:dbaf67cc-598c-4b26-b07f-5d781722ebfdPublication.

[31] T. Mannucci, E. Van Kampen, C.C. de Visser, and Q.P. Chux. SHERPA: A safe exploration algorithm for reinforcement learning controllers. In *AIAA Guidance, Navigation, and Control Conference, 2013*, 2015. ISBN 9781624103391.

[32] T. Mannucci, E.-J. 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, 2018. doi: 10.1109/TNNLS.2017.2654539.

[33] T.M. Moldovan and P. Abbeel. Safe exploration in Markov decision processes. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, volume 2, pages 1711–1718, 2012. ISBN 9781450312851.

[34] T.M. Moldovan and P. Abbeel. Risk aversion in Markov decision processes via near-optimal Chernoff bounds. In *Advances in Neural Information Processing Systems*, volume 4, pages 3131–3139, 2012. ISBN 9781627480031.

[35] K.S. Narendra and A.M. Annaswamy. Persistent excitation in adaptive systems. *International Journal of Control*, 45(1):127–160, 1987. doi: 10.1080/00207178708933715.

[36] Andrew Y. Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In Marcelo H. Ang and Oussama Khatib, editors, *Experimental Robotics IX*, pages 363–372, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-33014-1. doi: 10.1007/11552246_35.

[37] N. Passalis and A. Tefas. Deep reinforcement learning for controlling frontal person close-up shooting. *Neurocomputing*, 335:37–47, 2019. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2019.01.046. URL http://www.sciencedirect.com/science/article/pii/S0925231219300724.

[38] T.S.C. Pollack. Safe curriculum learning for primary flight control. Technical report, Delft University of Technology, 2019. URL "http://resolver.tudelft.nl/uuid:1b2becfd-c2db-43fc-a273-a3ff6a9ba50a".

[39] T.S.C. Pollack and E. van Kampen. Safe curriculum learning for optimal flight control of unmanned aerial vehicles with uncertain system dynamics. In *AIAA Scitech 2020 Forum*, volume 1 PartF, 2020. ISBN 9781624105951. doi: 10.2514/6.2020-2100.

[40] D. Pool. System identification of aerospace vehicles, state estimation. University Lecture 3, Delft University of Technology, 2019.

[41] P. Quinťia, R. Iglesias, M.A. Rodŕiguez, and C.V. Regueiro. Learning on real robots from experience and simple user feedback. *Journal of Physical Agents*, 7(1):56–64, 2013.

[42] B. Ravindran and A.G. Barto. Relativized Options: Choosing the Right Transformation. In *Proceedings, Twentieth International Conference on Machine Learning*, volume 2, pages 608–615, 2003. ISBN 1577351894.

[43] S. Rizvi and Z. Lin. Output Feedback Q-Learning Control for the Discrete-Time Linear Quadratic Regulator Problem. *IEEE Transactions on Neural Networks and Learning Systems*, 30(5):1523–1536, 2019. doi: 10.1109/TNNLS.2018.2870075.

[44] C. Robert and G. Casella. *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

[45] Francesco Sabatino. Quadrotor control: modeling, non-linear control design, and simulation, 2015.

[46] M. Sato, H. Kimura, and S. Kobayashi. Td algorithm for the variance of return and mean-variance reinforcement learning. *Transactions of the Japanese Society for Artificial Intelligence*, 16(3):353–362, dec 2001. ISSN 1346-0714. doi: 10.1527/tjsai.16.353.

[47] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel. Trust region policy optimization. In *32nd International Conference on Machine Learning, ICML 2015*, volume 3, pages 1889–1897, 2015. ISBN 9781510810587.

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

[49] H. Seijen and R. Sutton. True online td (lambda). In *International Conference on Machine Learning*, pages 692–700, 2014.

[50] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. doi: 10.1038/nature16961.

[51] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, T. Graepel, and D. Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017. doi: 10.1038/nature24270.

[52] R. S. Sutton and A. G. Barto. *Reinforcement learning: An Introduction (2nd edition draft; 2012)*. MIT Press, 2012. ISBN 0262193981. doi: 10.1109/MED.2013.6608833.

[53] M.E. Taylor and P. Stone. Representation transfer for reinforcement learning. In *AAAI Fall Symposium - Technical Report*, volume FS-07-03, pages 78–85, 2007. ISBN 9781577353485.

[54] M.E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

[55] M.E. Taylor, P. Stone, and Y. Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8:2125–2167, 2007.

[56] L. Torrey and M.E. Taylor. Help an agent out: Student/teacher learning in sequential decision tasks. In *Proceedings of the Adaptive and Learning Agents Workshop 2012, ALA 2012 - Held in Conjunction with the 11th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012*, pages 41–48, 2012.

[57] L. Torrey, J. Shavlik, T. Walker, and R. MacLin. *Relational macros for transfer in reinforcement learning*, volume 4894 LNAI. Springer, 2008. ISBN 9783540784685. doi: 10.1007/978-3-540-78469-2_25.

[58] E. Van Kampen. Global optimization using interval analysis: interval optimization for aerospace applications. Technical report, TU Delft University of Technology, Kluyverweg 1, 2629 HS Delft, Netherlands, 2010.

[59] R. van Paasen, M. Mulder, C. Borst, and D. Pool. Aerospace human-machine systems, introduction. University Lecture 1, Delft University of Technology, 2018.

[60] J. West, F. Maire, C. Browne, and S. Denman. Improved reinforcement learning with curriculum. *Expert Systems with Applications*, 158, 2020. doi: 10.1016/j.eswa.2020.113515.

A

Uncertain State Space Model

In this appendix the theory of uncertain state space models is described. Starting with interval arithmetic, Appendix A.1, which is the foundation for matrix arithmetic with uncertain parameters (Appendix A.2). Finally, in Appendix A.3 the theory is projected on an uncertain state space system, along with a verification of the theory proposed in this appendix. Indeed for uncertain state space systems, the parameters within the A and B matrices are assumed to be within an uncertainty interval, for which arithmetic rules exist.

A.1. Interval Arithmetic and Uncertain Values

Forming the basis of this appendix, is the interval arithmetic on which uncertain vector and matrix algebra is constructed. It is assumed that performing any arithmetic operation on two intervals, leads to the resulting interval containing all the possible combinations of each interval's values for that operation. According to [58], four principal rules exist and are quoted in this for summation (Equation (A.1)), subtraction(Equation (A.2)), multiplication (Equation (A.3)) and division. However, since the ultimate goal is to implement this for matrices and vectors, the division rule is not quoted in this report.

$$[a,b]+[c,d]=[a+c,b+d] \tag{A.1}$$

$$[a,b]-[c,d]=[a-d,b-c] \tag{A.2}$$

$$[a,b]\cdot[c,d]=\left[min(ac,ad,bc,bd),max(ac,ad,bc,bd)\right] \tag{A.3}$$

Uncertain values are generally expressed in a different form, namely $a\pm\epsilon$. Where a is an arbitrary variable defining the center of the interval and ϵ defines the maximal error above or below the central value a, also known as the *absolute uncertainty*. By assuming a positive absolute uncertainty, $\epsilon\geq 0$, an uncertain value can be translated to interval formulation as given below:

$$a\pm\epsilon_a=[a-\epsilon_a,a+\epsilon_a] \tag{A.4}$$

Using Equation (A.3) and considering the interval formulation of uncertainty given above, the multiplication of two uncertain values $p\pm\epsilon_p$ and $q\pm\epsilon q$ is can be derived. The derivation is divided into the 4 parts defined by the four elements used in the the $min()$ and $max()$ arguments of Equation (A.3). Namely, denominated by ac,ad,bc and bd.

First, the multiplication of uncertain values is rewritten in the interval multiplication format, given by Equation (A.3).

$$\begin{aligned}(p\pm\epsilon_p)\cdot(q\pm\epsilon_q)&=[p-\epsilon_p,p+\epsilon_p]\cdot[q-\epsilon_q,q+\epsilon_q] \qquad for\ \ \epsilon_p,\epsilon_q\geq 0\\ &=\Big[min\big((p-\epsilon_p)(q-\epsilon_q),(p-\epsilon_p)(q+\epsilon_q),(p+\epsilon_p)(q-\epsilon_q),(p+\epsilon_p)(q+\epsilon_q)\big),\\ &\qquad max\big((p-\epsilon_p)(q-\epsilon_q),(p-\epsilon_p)(q+\epsilon_q),(p+\epsilon_p)(q-\epsilon_q),(p+\epsilon_p)(q+\epsilon_q)\big)\Big]\end{aligned} \tag{A.5}$$

In a second instance, the components from Equation (A.3), (ac,ad,bc,bd), are derived separately, for the sake of overview in equations A.6, A.7, A.8 and A.9, respectively.

ac:

$$\begin{aligned}(p-\epsilon_p)(q-\epsilon_q) &= pq - p\epsilon_q - q\epsilon_p + \epsilon_p\epsilon_q \\ &= pq - pq\frac{\epsilon_q}{q} - pq\frac{\epsilon_p}{p} + pq\frac{\epsilon_p\epsilon_q}{pq} \\ &= pq\left(1 - \frac{\epsilon_q}{q} - \frac{\epsilon_p}{p} + \frac{\epsilon_p\epsilon_q}{pq}\right) = pq\left(1 - \frac{\epsilon_p}{p} - \frac{\epsilon_q}{q} + \frac{\epsilon_p\epsilon_q}{pq}\right)\end{aligned} \tag{A.6}$$

ad:

$$\begin{aligned}(p-\epsilon_p)(q+\epsilon_q) &= pq + p\epsilon_q - q\epsilon_p - \epsilon_p\epsilon_q \\ &= pq + pq\frac{\epsilon_q}{q} - pq\frac{\epsilon_p}{p} - pq\frac{\epsilon_p\epsilon_q}{pq} \\ &= pq\left(1 + \frac{\epsilon_q}{q} - \frac{\epsilon_p}{p} - \frac{\epsilon_p\epsilon_q}{pq}\right) = pq\left(1 - \frac{\epsilon_p}{p} + \frac{\epsilon_q}{q} - \frac{\epsilon_p\epsilon_q}{pq}\right)\end{aligned} \tag{A.7}$$

bc:

$$\begin{aligned}(p+\epsilon_p)(q-\epsilon_q) &= pq - p\epsilon_q + q\epsilon_p - \epsilon_p\epsilon_q \\ &= pq - pq\frac{\epsilon_q}{q} + pq\frac{\epsilon_p}{p} - pq\frac{\epsilon_p\epsilon_q}{pq} \\ &= pq\left(1 - \frac{\epsilon_q}{q} + \frac{\epsilon_p}{p} - \frac{\epsilon_p\epsilon_q}{pq}\right) = pq\left(1 + \frac{\epsilon_p}{p} - \frac{\epsilon_q}{q} - \frac{\epsilon_p\epsilon_q}{pq}\right)\end{aligned} \tag{A.8}$$

bd:

$$\begin{aligned}(p+\epsilon_p)(q+\epsilon_q) &= pq + p\epsilon_q + q\epsilon_p + \epsilon_p\epsilon_q \\ &= pq + pq\frac{\epsilon_q}{q} + pq\frac{\epsilon_p}{p} + pq\frac{\epsilon_p\epsilon_q}{pq} \\ &= pq\left(1 + \frac{\epsilon_q}{q} + \frac{\epsilon_p}{p} + \frac{\epsilon_p\epsilon_q}{pq}\right) = pq\left(1 + \frac{\epsilon_p}{p} + \frac{\epsilon_q}{q} + \frac{\epsilon_p\epsilon_q}{pq}\right)\end{aligned} \tag{A.9}$$

In the above equations, the terms $\frac{\epsilon_p}{p}$ and $\frac{\epsilon_q}{q}$ are the *relative uncertainties* of p and q, respectively. The above equations can be combined to a matrix formulation to find ac, ad, bc and bd, as shown in Equation (A.10).

$$\begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix} = pq \begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ \epsilon_q/q \\ \epsilon_p/p \\ (\epsilon_p\epsilon_q)/(pq) \end{bmatrix} \tag{A.10}$$

The above formulation provides a computationally more optimised operation. Once all values for ac, ad, bc and bd are found, the minimum and maximum of this vector can be easily computed.

A.2. Uncertain Matrix Algebra

In this section, the arithmetic discovered in previous section is elaborated further to understand how uncertainty propagates through vector and matrix multiplications. Uncertain vectors and matrices are understood to have uncertain elements. Clarification is given, by providing a definition of uncertain vector notation in Equation (A.11). In this equation a 2×1 vector is considered.

$$\begin{aligned}\tilde{v} &= \begin{bmatrix} v_1 \pm \epsilon_{v_1} & v_2 \pm \epsilon_{v_2} \end{bmatrix} \\ &= \begin{bmatrix} [v_1 - \epsilon_{v_1}, v_1 + \epsilon_{v_1}] & [v_2 - \epsilon_{v_2}, v_2 + \epsilon_{v_2}] \end{bmatrix} \\ &= \begin{bmatrix} [v_{1,a}, v_{1,b}] & [v_{2,a}, v_{2,b}] \end{bmatrix}\end{aligned} \tag{A.11}$$

Since uncertain matrices can be seen as a stacking of uncertain vectors, vector algebra is first derived before making statements regarding uncertain matrix algebra.

Consider a second vector $\tilde{x}$, which is a column vector. The uncertain vector multiplication, $\tilde{v}\tilde{x}$ is then derived in Equation (A.12). Which gives insight to the interval arithmetic required. First each term needs to be multiplied using Equation (A.3) and then all terms are added together using Equation (A.1).

$$\begin{aligned}\tilde{v} \cdot \tilde{x} &= \begin{bmatrix} [v_{1,a}, v_{1,b}] & [v_{2,a}, v_{2,b}] \end{bmatrix} \cdot \begin{bmatrix} [x_{1,a}, x_{1,b}] \\ [x_{2,a}, x_{2,b}] \end{bmatrix} \\ &= \begin{bmatrix} [v_{1,a}, v_{1,b}] \cdot [x_{1,a}, x_{1,b}] + [v_{2,a}, v_{2,b}] \cdot [x_{2,a}, x_{2,b}] \end{bmatrix}\end{aligned} \tag{A.12}$$

The derivation of the resulting interval of is given in Equation (A.13). Depending on what type of uncertainty is provided, *relative* or *absolute*, the products in the equation above can be replaced by the relative error formulation given in Equations A.6, A.7, A.8 and A.9. Additionally, it can be seen that if dimensions of the vectors are increased, additional intervals would be completing the sum of the above equation.

$$\begin{aligned}&[v_{1,a}, v_{1,b}] \cdot [x_{1,a}, x_{1,b}] + [v_{2,a}, v_{2,b}] \cdot [x_{2,a}, x_{2,b}] \\ &= \left[\min\left(v_{1,a}x_{1,a}, v_{1,a}x_{1,b}, v_{1,b}x_{1,a}, v_{1,b}x_{1,b}\right), \max\left(v_{1,a}x_{1,a}, v_{1,a}x_{1,b}, v_{1,b}x_{1,a}, v_{1,b}x_{1,b}\right)\right] + \\ &\quad \left[\min\left(v_{2,a}x_{2,a}, v_{2,a}x_{2,b}, v_{2,b}x_{2,a}, v_{2,b}x_{2,b}\right), \max\left(v_{2,a}x_{2,a}, v_{2,a}x_{2,b}, v_{2,b}x_{2,a}, v_{2,b}x_{2,b}\right)\right]\end{aligned} \tag{A.13}$$

If a matrix is seen as a stacking of vectors, the algebra found above, can be used row and column wise for the first and second matrix, respectively.

A.3. Uncertain State Space Systems

The ultimate goal of this appendix is to understand how uncertainty in a state space system can be sourced for predictive state propagation. In this section, the theory on uncertainty propagation is linked to state space systems that have inherent parametric uncertainty. In essence, state space systems follow the algebraic rules of matrix multiplication. Likewise, an uncertain state space system follows the uncertain matrix multiplication rules described in Appendix A.2.

An uncertain state space system is defined by uncertain system dynamics in $\tilde{A}$ and uncertain input dynamics $\tilde{B}$. Additionally, the case for uncertain state and input vectors, $\tilde{x}$ and $\tilde{u}$, is considered. Should one of these elements be certain, it can be replaced by an uncertain formulation with $\epsilon = 0$. For clarification, the notation of an uncertain discrete-time state space system is given below.

$$\tilde{x}_{k+1} = \tilde{A}\tilde{x}_k + \tilde{B}\tilde{u}_k \tag{A.14}$$

In order to verify the theory laid out in previous sections, the uncertain state propagation of a sinusoidal signal was implemented. With a relative uncertainty of 1% and 0.01% for all elements of A and B, the propagation is given in Figures A.1 and A.2, respectively.

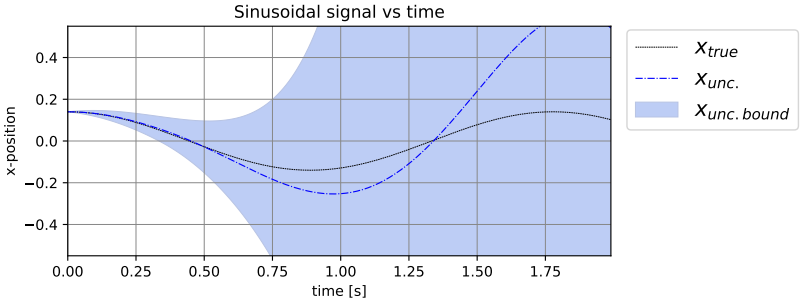


Figure A.1: Uncertain state propagation through an uncertain state space system of a sinusoidal signal; Relative parametric uncertainties of $\tilde{A}$ and $\tilde{B}$ are 1%.

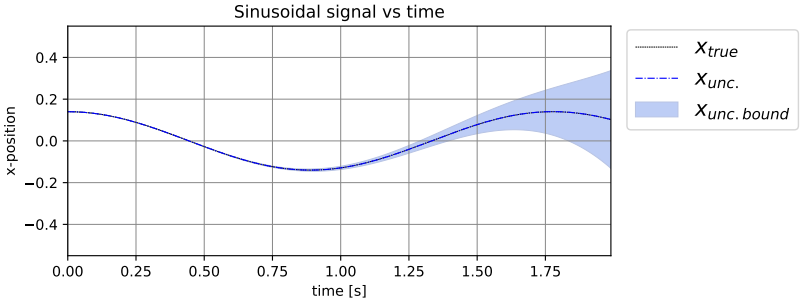


Figure A.2: Uncertain state propagation through an uncertain state space system of a sinusoidal signal; Relative parametric uncertainties of $\bar{A}$ and $\bar{B}$ are 0.01%.

Note, the input is set to zero during these simulations, to allow for better understanding of the uncertainty region propagation. Step responses are not of interest here, since the initial state space system used for this simulation is derived from Appendix B and has been proven to work. From Figures A.1 and A.2, it can be seen that providing a lower uncertainty results in a more accurate propagation of the state over time. This behaviour is expected. The line representing $x_{unc.}$, is computed by taking the middle of the uncertainty interval at that point in time. Interestingly, this line detaches after about 0.5 seconds for when having a larger relative uncertainty. Moreover, the sate propagation deviates drastically in Figure A.1, which means that having a large uncertainty can lead to an objectively bad approximation of a system's behaviour. This is to be expected.

In this appendix, the intrinsic properties of uncertain state propagation through state space systems is derived. First presenting the foundation on which these properties are based, namely interval arithmetic. This theory is then expanded to cover uncertain vector and matrix multiplications. Finally, these findings are applied to uncertain state space systems. In order to ensure the theory in this appendix is correct, verification was performed on uncertain state propagation of a sinusoidal signal over time. Conclusive results show that the theory derived in these sections behaves as expected. Namely, provided a lower uncertainty (relative or absolute), the uncertain state propagates more closely to the true state.

B

Cascaded N-Mass-Spring-Damper System

This appendix elaborates on the equations of motions governing a linear N-MSD system. The system is of the form shown in Figure B.1. Note the last mass, m_n, is free on the right hand side, hence only being influenced by the spring and damper with constants k_n and c_n, respectively on the left hand side.

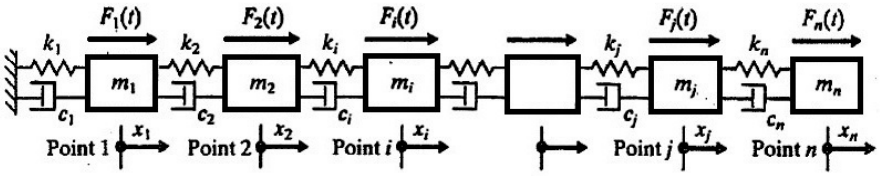


Figure B.1: Overview of a cascaded N-MSD system (taken from [16])

B.1. Governing Equations of Motion for an N-MSD System

On closer inspection, a single mass is always influenced by five forces, the spring forces of both sides, the damper forces of both sides and an input control force. With the exception of the last mass as mentioned before. The masses are assumed to be friction-less and the spring and dampers are assumed to be lossless. The free body diagram for one of those masses is given in Figure B.2 with the assumption that the system is stretched to the right.

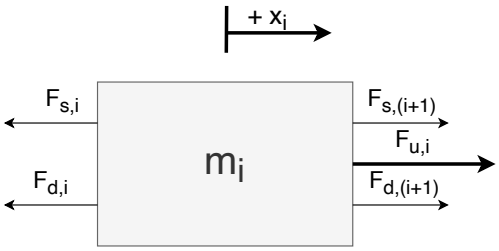


Figure B.2: Free body diagram of the i^{th} mass in the N-MSD system

Furthermore, the spring force and damper force are defined in terms of relative mass positions. The governing equations for these forces are given in Equation (B.1). Clearly, the forces are dependent not only on the position and velocity of the i^{th} mass, $\{x_i, \dot{x}_i\}$, but also on the equivalent parameters of the preceding mass, m_{i-1}, and the subsequent mass, m_{i+1}.

$$\begin{aligned} F_{s,i} &= k_i(x_i - x_{i-1}) \\ F_{d,i} &= c_i(\dot{x}_i - \dot{x}_{i-1}) \end{aligned} \tag{B.1}$$

One exceptions to Equation (B.1) exist. Namely the first mass, which is connected to a static boundary. Consequently, in that specific case, x_{i-1} and $\dot{x}_{i-1}$ are said to be constantly 0.

According to Newton's second law of motion, $F = m \cdot a$, the acceleration of the i^{th} mass is governed by the equation of motion derived in Equation (B.2). Which is defined in matrix form in Equation (B.3).

$$
\begin{aligned}
m_i \cdot \ddot{x}_i = \sum F &= -F_{s,i} - F_{d,i} + F_{s,(i+1)} + F_{d,(i+1)} + F_{u,i} \\
&= -\left[k_i \cdot (x_i - x_{i-1})\right] - \left[c_i \cdot (\dot{x}_i - \dot{x}_{i-1})\right] \\
&\quad + \left[k_{i+1} \cdot (x_{i+1} - x_i)\right] + \left[c_{i+1} \cdot (\dot{x}_{i+1} - \dot{x}_i)\right] + F_{u,i} \\
&= -\left[k_i x_i - k_i x_{i-1}\right] - \left[c_i \dot{x}_i - c_i \dot{x}_{i-1}\right] \\
&\quad + \left[k_{i+1} x_{i+1} - k_{i+1} x_i\right] + \left[c_{i+1} \dot{x}_{i+1} - c_{i+1} \dot{x}_i\right] + F_{u,i} \\
&= k_i x_{i-1} - (k_i + k_{i+1}) x_i + k_{i+1} x_{i+1} \\
&\quad + c_i \dot{x}_{i-1} - (c_i + c_{i+1}) \dot{x}_i + c_{i+1} \dot{x}_{i+1} + F_{u,i}
\end{aligned}
\tag{B.2}
$$

$$
\Rightarrow \ddot{x}_i = \frac{k_i}{m_i} x_{i-1} + \frac{-(k_i + k_{i+1})}{m_i} x_i + \frac{k_{i+1}}{m_i} x_{i+1} + \frac{c_i}{m_i} \dot{x}_{i-1} + \frac{-(c_i + c_{i+1})}{m_i} \dot{x}_i + \frac{c_{i+1}}{m_i} \dot{x}_{i+1} + \frac{1}{m_i} F_{u,i}
$$

$$
\begin{bmatrix} \ddot{x}_i \end{bmatrix} = \begin{bmatrix} \frac{k_i}{m_i} & -\frac{(k_i+k_{i+1})}{m_i} & \frac{k_{i+1}}{m_i} & \frac{c_i}{m_i} & \frac{-(c_i+c_{i+1})}{m_i} & \frac{c_{i+1}}{m_i} \end{bmatrix} \cdot \begin{bmatrix} x_{i-1} \\ x_i \\ x_{i+1} \\ \dot{x}_{i-1} \\ \dot{x}_i \\ \dot{x}_{i+1} \end{bmatrix} + \begin{bmatrix} 1 \end{bmatrix} \cdot \begin{bmatrix} F_{u,i} \end{bmatrix} \tag{B.3}
$$

Note, this formulation does not hold for the last mass in the cascade, m_n, for which k_{i+1} and c_{i+1} are zero. Thus the equation of motion can be relieved from its dependence on x_{i+1}, which is non-existent for m_n.

When combining masses, mass positions used in the respective equations will overlap. The example is given for a 1-MSD and a 3-MSD system in Equations B.4 and B.5 respectively.

$$
\begin{bmatrix} \ddot{x}_1 \end{bmatrix} = \begin{bmatrix} \frac{-k_1}{m_1} & \frac{-c_1}{m_1} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \dot{x}_1 \end{bmatrix} + \begin{bmatrix} \frac{1}{m_1} \end{bmatrix} \cdot \begin{bmatrix} F_{u_1} \end{bmatrix} \tag{B.4}
$$

$$
\begin{bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \\ \ddot{x}_3 \end{bmatrix} = \begin{bmatrix} \frac{-(k_1+k_2)}{m_1} & \frac{k_2}{m_1} & 0 & \frac{-(c_1+c_2)}{m_1} & \frac{c_2}{m_1} & 0 \\ \frac{k_2}{m_2} & \frac{-(k_2+k_3)}{m_2} & \frac{k_3}{m_2} & \frac{c_2}{m_2} & \frac{-(c_2+c_3)}{m_2} & \frac{c_3}{m_2} \\ 0 & \frac{k_3}{m_3} & \frac{-k_3}{m_3} & 0 & \frac{c_3}{m_3} & \frac{-c_3}{m_3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} + \begin{bmatrix} \frac{1}{m_1} & 0 & 0 \\ 0 & \frac{1}{m_2} & 0 \\ 0 & 0 & \frac{1}{m_3} \end{bmatrix} \cdot \begin{bmatrix} F_{u_1} \\ F_{u_2} \\ F_{u_3} \end{bmatrix} \tag{B.5}
$$

Clearly a pattern starts to emerge inside both, the spring and damper parts the large matrix. In which, except for the first an last mass, a block as given in equations B.6 and B.7 for the spring part and damper part, respectively recursively appears along the matrix diagonal. This pattern follows a *tridiagonal Toeplitz matrix*.

$$
\begin{bmatrix} -(k_i + k_{i+1}) & k_{i+1} \\ k_{i+1} & -(k_{i+1} + k_{i+2}) \end{bmatrix} \tag{B.6}
$$

$$
\begin{bmatrix} -(c_i + c_{i+1}) & c_{i+1} \\ c_{i+1} & -(c_{i+1} + c_{i+2}) \end{bmatrix} \tag{B.7}
$$

B.2. State Space System of a 3-MSD System

In order to implement such system it is primordial that the equations of motion are translated into state space form (i.e. in matrices A, B, C, D). The general format of a state space system will be used:

$$\begin{cases} \dot{\bar{x}} &= A\bar{x} + B\bar{u} \\ \bar{y} &= C\bar{x} + D\bar{u} \end{cases} \tag{B.8}$$

First let's assume the state $\bar{x}$, and input, $\bar{u}$, for a 3-MSD system to be defined as:

$$\bar{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} \Rightarrow \dot{\bar{x}} = \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \ddot{x_1} \\ \ddot{x_2} \\ \ddot{x_3} \end{bmatrix} \quad and \quad \bar{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{B.9}$$

Which leads to the general state-space formulation of a 3-MSD system is given by Equation (B.10), where C is designed to give the positions of all three masses, $\{m_1, m_2, m_3\}$, as output of the system. However, the former can be adapted to generate the desired output.

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -\frac{k_1+k_2}{m_1} & \frac{k_2}{m_1} & 0 & -\frac{c_1+c_2}{m_1} & \frac{c_2}{m_1} & 0 \\ \frac{k_2}{m_2} & -\frac{k_2+k_3}{m_2} & \frac{k_3}{m_2} & \frac{c_2}{m_2} & -\frac{c_2+c_3}{m_2} & \frac{c_3}{m_2} \\ 0 & \frac{k_3}{m_3} & -\frac{k_3}{m_3} & 0 & \frac{c_3}{m_3} & -\frac{c_3}{m_3} \end{bmatrix} \; ; \; B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{m_1} & 0 & 0 \\ 0 & \frac{1}{m_2} & 0 \\ 0 & 0 & \frac{1}{m_3} \end{bmatrix}$$
$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \; ; \; D = \begin{bmatrix} 0 \end{bmatrix} \tag{B.10}$$

When more masses are desired, the patterns described in Equations B.6 and B.7, can be repeated in their respective parts of A for the necessary amount of masses. Additionally, the top right quarter of A is an identity matrix, I_N, of size N, being the number of masses in the cascade.

C

Linear Quadratic Tracking Task for N-MSD System

This appendix contains the necessary derivations for the the augmented state space system used in the tracking task in the preliminary analysis (Chapter 8). As mentioned in Chapter 3, the LQT task is basically a Linear Quadratic Regulation (LQR) problem, where the error between the state and the desired state is regulated instead of the state itself. Focus will be set on the tracking task for the last mass of a MSD system consisting of three masses.

First the state space system of the reference signal will be discussed in Appendix C.1, after which the augmented state space system for LQT will be derived in Appendix C.2.

C.1. Reference Signal

For the task presented in Chapter 8, the reference signal was chosen to be sinusoidal. There are two possibilities of defining a state space system for such signal. First, is linearising the function $sin(x)$ using a Taylor expansion. Another would be to understand that an undamped friction-less Mass-Spring (MS) system, with $m = 1$, will create a perfect sinusoidal signal. Since, a model for a 1-MSD is already defined in Equation (B.4), simply setting the damper constant to zero results in a 1-MS system, given in Equation (C.1).

$$\begin{aligned} \begin{bmatrix} \dot{x}^r \\ \ddot{x}^r \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ k_{ref} & 0 \end{bmatrix} \begin{bmatrix} x^r \\ \dot{x}^r \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \end{bmatrix} \\ y^r &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x^r \\ \dot{x}^r \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} \begin{bmatrix} 0 \end{bmatrix} \end{aligned} \tag{C.1}$$

The constant, k_{ref}, determines the frequency of the system. Hence, if it is assumed that the reference signal takes a frequency, f_{ref}, as input, the correct k_{ref} is computed according to Equation (C.2). The desired phase, Φ_{ref}, and amplitude, A_{ref}, are taken into account when the initial conditions ($t = 0$) of the reference signal are computed.

$$k_{ref} = \left(2 \cdot \pi \cdot f_{ref}\right)^2 \tag{C.2}$$

$$\begin{cases} x_0^r &= A_{ref} \cdot sin(\Phi_{ref}) \\ \dot{x}_0^r &= A_{ref} \cdot (2\pi f_{ref}) \cdot cos(\Phi_{ref}) \end{cases} \tag{C.3}$$

C.2. Augmented State Space System for LQT

As was explained in Chapter 3, an augmented state space system is created for solving the LQT task, required to regulate the error. In this section the augmented state space system for a 3-MSD system is derived. The general formulation for the continuous-time state pace is presented in Equation (C.4).

$$\dot{x} = T_c x + B_{1,c} u \begin{bmatrix} A_c & 0 \\ 0 & F_c \end{bmatrix} x + \begin{bmatrix} B_c \\ 0 \end{bmatrix} u \tag{C.4}$$

In Equation (C.4), the matrix T_c is build by creating a block diagonal of the continuous-time matrix A_c from the MSD state space given in Equation (B.10), and, F_c, the continuous-time internal dynamics matrix of the reference signal. From Equation (C.1), it can be deduced that $F_c = \begin{bmatrix} 0 & 1 \\ k_{ref} & 0 \end{bmatrix}$.

$$A_c = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -\frac{k_1+k_2}{m_1} & \frac{k_2}{m_1} & 0 & -\frac{c_1+c_2}{m_1} & \frac{c_2}{m_1} & 0 \\ \frac{k_2}{m_2} & -\frac{k_2+k_3}{m_2} & \frac{k_3}{m_2} & \frac{c_2}{m_2} & -\frac{c_2+c_3}{m_2} & \frac{c_3}{m_2} \\ 0 & \frac{k_3}{m_3} & -\frac{k_3}{m_3} & 0 & \frac{c_3}{m_3} & -\frac{c_3}{m_3} \end{bmatrix} \; ; \; B_c = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{m_1} & 0 & 0 \\ 0 & \frac{1}{m_2} & 0 \\ 0 & 0 & \frac{1}{m_3} \end{bmatrix} \tag{B.10}$$

$$C_c = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \; ; \; D_c = \begin{bmatrix} 0 \end{bmatrix}$$

As a result, T_c and $B_{1,c}$, are defined as formulated in Equation (C.5). These matrices can be converted in their respective discrete-time formulations easily. The latter is required for using the theory presented in Chapter 3.

$$T_c = \left[\begin{array}{cccccc|cc} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ -\frac{k_1+k_2}{m_1} & \frac{k_2}{m_1} & 0 & -\frac{c_1+c_2}{m_1} & \frac{c_2}{m_1} & 0 & 0 & 0 \\ \frac{k_2}{m_2} & -\frac{k_2+k_3}{m_2} & \frac{k_3}{m_2} & \frac{c_2}{m_2} & -\frac{c_2+c_3}{m_2} & \frac{c_3}{m_2} & 0 & 0 \\ 0 & \frac{k_3}{m_3} & -\frac{k_3}{m_3} & 0 & \frac{c_3}{m_3} & -\frac{c_3}{m_3} & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -k_{ref} & 0 \end{array}\right] \; ; \; B_{1,c} = \left[\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{m_1} & 0 & 0 \\ 0 & \frac{1}{m_2} & 0 \\ 0 & 0 & \frac{1}{m_3} \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}\right] \tag{C.5}$$

With regard to the LQ objective for the tracking task, the error a time step k is known to be, $e_k = X_k^T Q_1 X_k + u_k^T R u_k$, in which the matrix Q_1 translates the augmented state, X_k, into a weighted error. In order to compute $Q_1 = C_1^T Q C_1$, the matrix C_1 is required. The latter, when multiplied with the augmented state, X_k, results in the error of each state compared to its reference signal.

$$C_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \end{bmatrix} \tag{C.6}$$

The task at hand, and defined in Chapter 8, is to optimally enforce the position of the last mass, x_3, to track the sinusoidal reference signal presented in Appendix C.1, whilst regulating the other mass positions to zero. Consequently, C_1 is defined as Equation (C.6). Clearly, when multiplied with the augmented state, the first two rows will give the position error from the zero line and the last row will result in the error between the reference signal position, x^r, and the position of the last mass, x_3.

D

Supporting Plots for Preliminary Analysis

This chapter contains plots mentioned in the preliminary analysis chapter, Chapter 8. These were not shown there for sake of clarity, but will be given here. The same section structure as in Chapter 8 will be reflected here for consistency.

D.1. Flat Learning Benchmark

The figures in this section are presenting the learning episodes of the offline learning process presented in Section 8.2. The reader is invited to find more detailed explanation in that section.

Firstly, the mass positions of a 3-MSD are shown in Figure D.1, accompanied with their respective control forces. These are taken from the first episode in a series of three, of offline VI, with now curriculum learning applied.

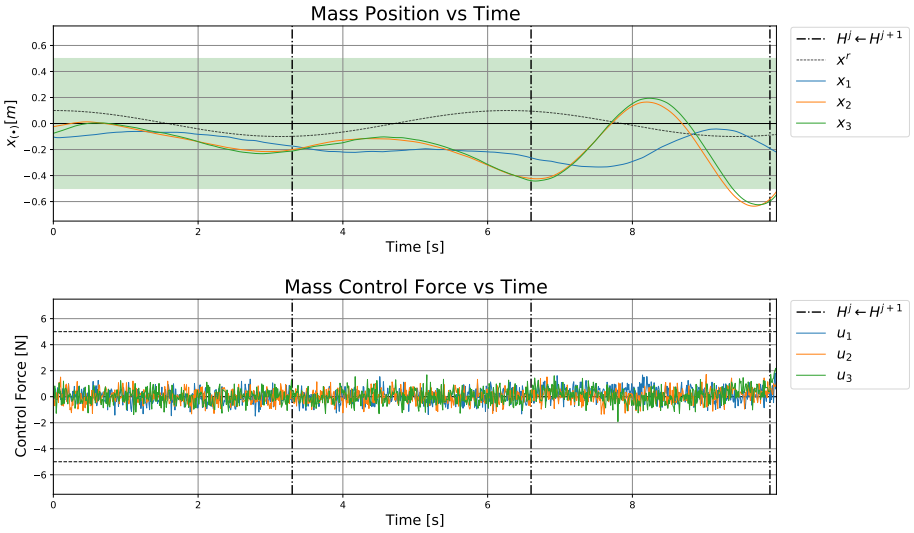
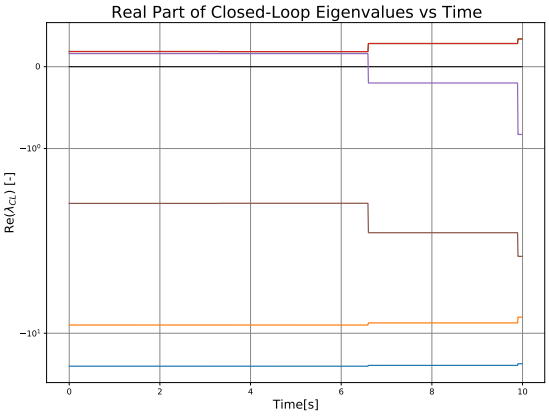


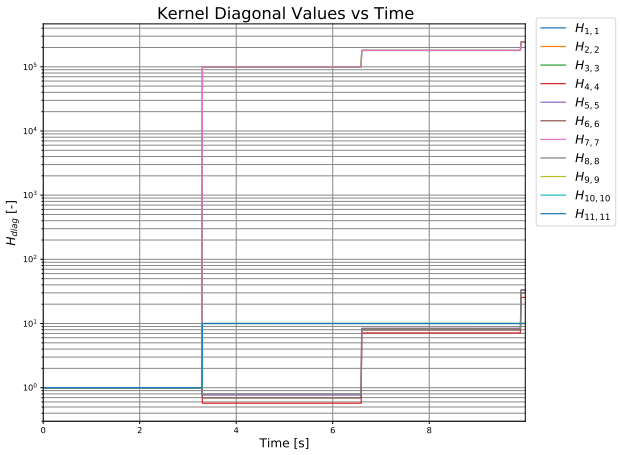
Figure D.1: Evolution of mass positions over time; Offline VI episode 1; Flat Learning; No SHERPA; No sensor noise; $x_i \in [-0.2, 0.2]$; $\dot{x}_i \in [-0.25, 0.25]$; Green is the SSS for x_3.

Figures D.2a and D.2b are also taken from the first episode in this series. Figure D.2a shows the evolution of the closed loop eigenvalues' real part throughout the episode. On the other hand, Figure D.2b shows the evolution of the diagonal parameters of H in the first episode.

Equivalently, the second episode of the offline VI learning process is shown in this section. The positions and control forces associated with each mass during this episode are shown in Figure D.3. Whereas, the evolution of the closed loop eigenvalues are shown in Figure D.4a. The attentive reader can see the continuation between figures D.2a and D.4a. Likewise the continuation appears in between figures D.2b and D.4b. The latter shows the evolution of the kernel values situated along the diagonal of H during the second episode.



(a) Evolution of the real part of the 3-MSD system's closed-loop eigenvalues; Offline VI episode 1; Flat Learning; No SHERPA; No sensor noise.



(b) Evolution over time of the diagonal values of H for the 3-MSD system; Offline VI episode 1; Flat Learning; No SHERPA; No sensor noise.

Figure D.2: Evolution of closed-loop eigenvalues and kernel diagonal values over time; Offline VI episode 1; Flat Learning; No SHERPA; No sensor noise.

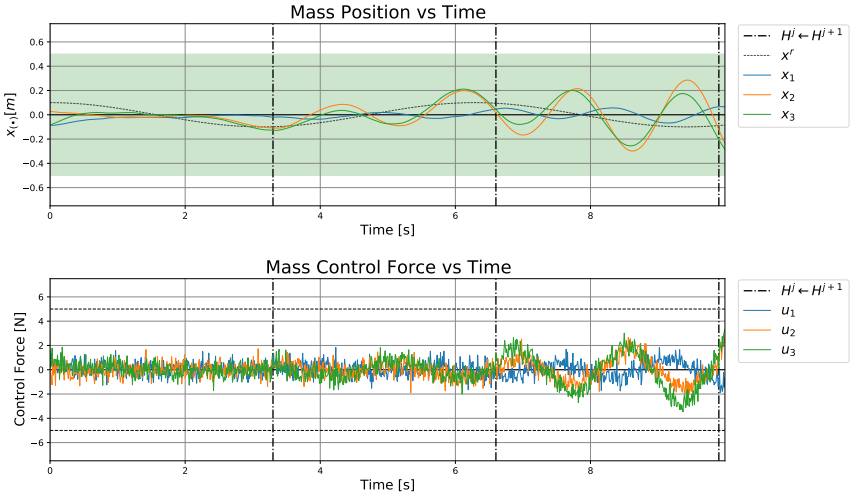
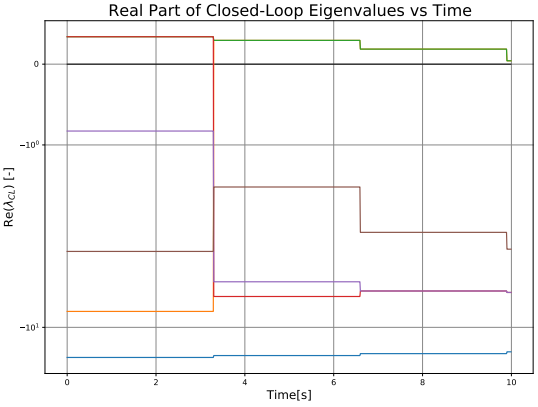
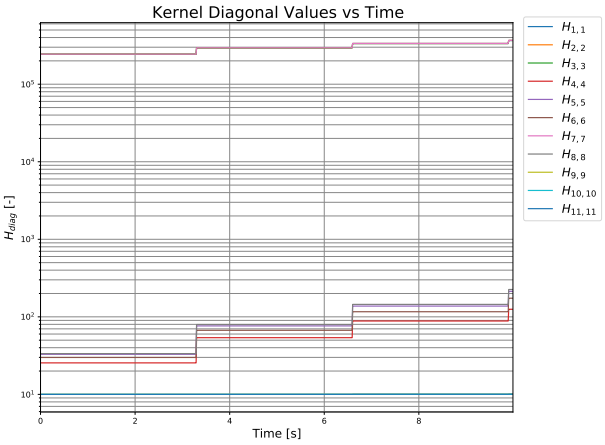


Figure D.3: Evolution of mass positions over time; Offline VI episode 2; Flat Learning; No SHERPA; No sensor noise; $x_i \in [-0.2, 0.2]$; $\dot{x}_i \in [-0.25, 0.25]$; Green is the SSS for x_3.



(a) Evolution of the real part of the 3-MSD system's closed-loop eigenvalues; Offline VI episode 2; Flat Learning; No SHERPA; No sensor noise.



(b) Evolution over time of the diagonal values of H for the 3-MSD system; Offline VI episode 2; Flat Learning; No SHERPA; No sensor noise.

Figure D.4: Evolution of closed-loop eigenvalues and kernel diagonal values over time; Offline VI episode 2; Flat Learning; No SHERPA; No sensor noise.

D.2. Curriculum Learning

This section presents a larger version of figures 8.8a and 8.8b, for better readability. Respectively, figures D.5 and D.6, show the position of all the masses present in the system during the first and second curricular step.

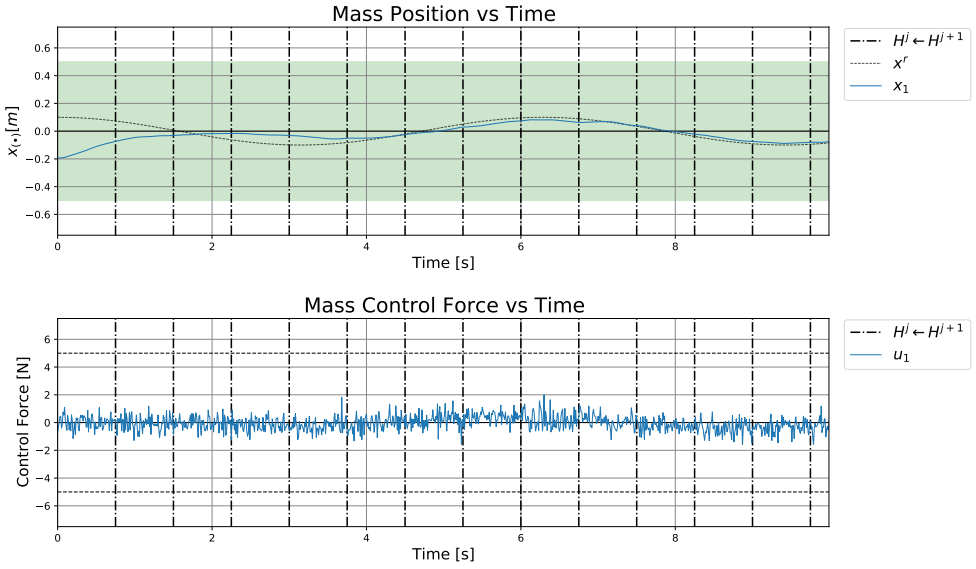


Figure D.5: Evolution of mass positions over time; Online VI; Curriculum Learning (step 1); No SHERPA; No sensor noise; $x_i \in [-0.2, 0.2]$; $\dot{x}_i \in [-0.25, 0.25]$; Green is the SSS for x_1.

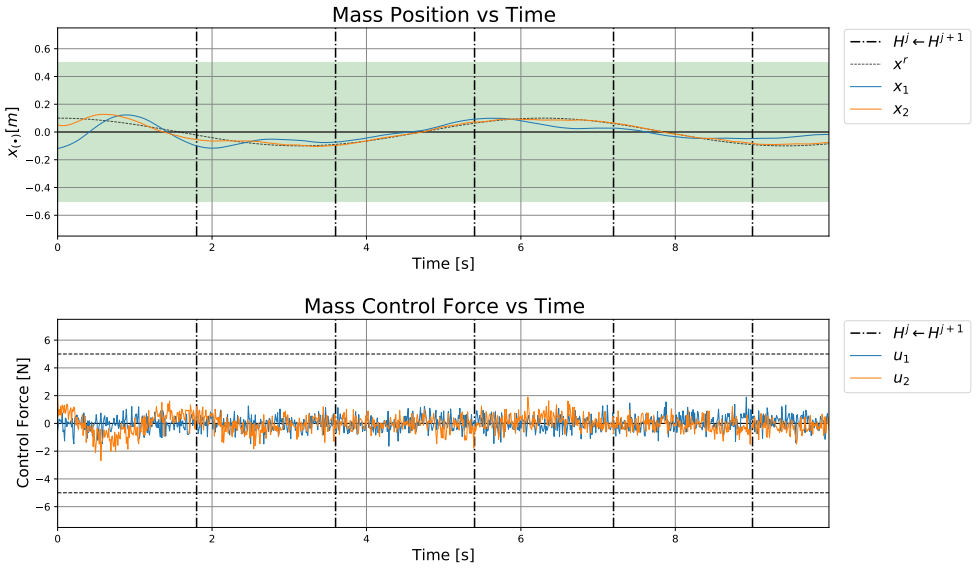


Figure D.6: Evolution of mass positions over time; Online VI; Curriculum Learning (step 2); No SHERPA; No sensor noise; $x_i \in [-0.2, 0.2]$; $\dot{x}_i \in [-0.25, 0.25]$; Green is the SSS for x_2.

D.3. Safe Curriculum Learning

In this section supporting plots for the safe learning implementation of the preliminary analysis are provided. Images are placed here for two reasons. The first is for better readability, when the plots had to be scaled down for sake of overview in the main chapter. These images are used in Chapter 8 in a smaller form factor to provide the reader with an idea of the trends in the graphs. A second reason, is that the focus lied on a particular curricular step. It is still important to see the other curricular steps that lead to the plot shown in Chapter 8. Therefore they are given in this section of the appendix.

In Figure D.7, the evolution of the absolute parametric uncertainties of the state space model used in SHERPA is shown. The top graphs shows the diagonal values of the the uncertain state space matrix $\tilde{A}$. The other two show all the element values of both $\tilde{B}$ and $\tilde{F}$. During the simulation from which these graphs are derived, measurement noise is added onto the state.

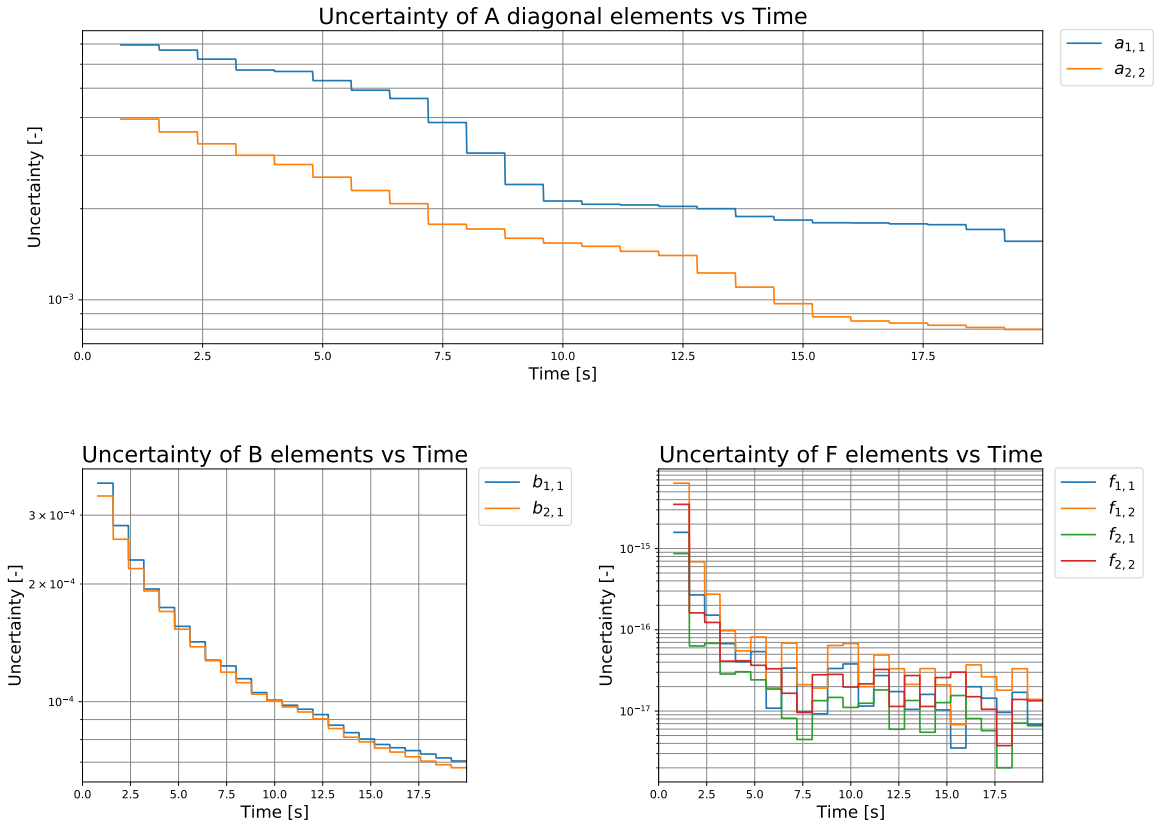


Figure D.7: Evolution of the absolute parametric uncertainty of all matrices forming the uncertain model estimate used in SHERPA, $\tilde{A}$, $\tilde{B}$ and $\tilde{F}$; Online VI; Safe Curriculum Learning (step 1); SHERPA activated; Gaussian sensor noise, $\sigma = 0.002$.

The figure above is linked to the analysis of the impact SHERPA has on a single noisy curricular step of a 1-MSD system.

In contrast, the figures below relate to the inclusion of a SHERPA in a full curriculum. Below, the first two figures (Figures D.8 and D.9) provide better readability of figures used in Section 8.4.3. They show the state evolution of the first mass and its input (in Figure D.8) during the first curricular step. Safe curriculum learning simulation condition apply with measurement noise. Figure D.9, shows the interventions of SHERPA on the trajectory of the first mass, x_1, during the first curricular step.

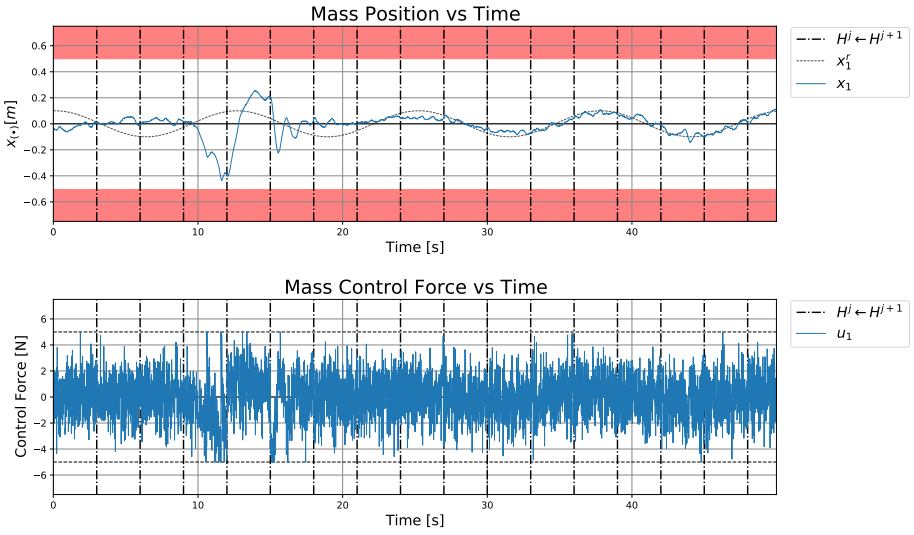


Figure D.8: Evolution of mass position over time for a 1-MSD system; Online VI; Safe Curriculum Learning (step 1); SHERPA activated; Gaussian sensor noise, $\sigma = 0.0012$; $x_i \in [-0.2, 0.2]$; $\dot{x}_i \in [-0.25, 0.25]$; Red is FSS for x_1.

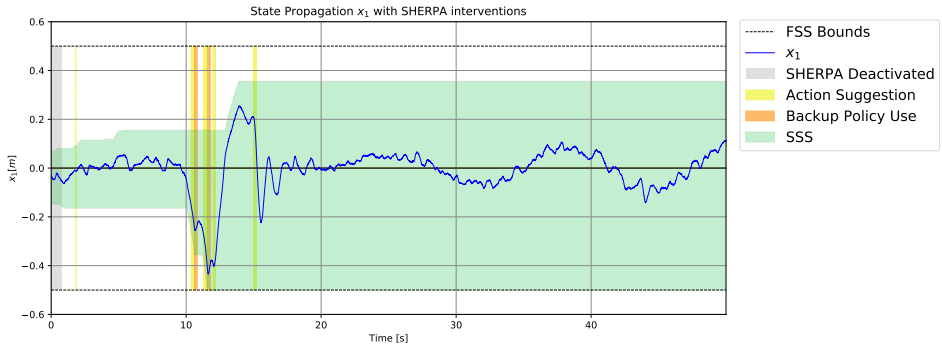


Figure D.9: Evolution of the SSS over time along with SHERPA interventions for the position of the first mass, x_1; Online VI; Safe Curriculum Learning (step 1); SHERPA activated; Gaussian sensor noise, $\sigma = 0.0012$.

In a second instance the same readability improvement is granted to the second curricular step in Figures D.10 and D.11, showing the position state evolution over time and the SHERPA interventions for a 2-MSD system, respectively.

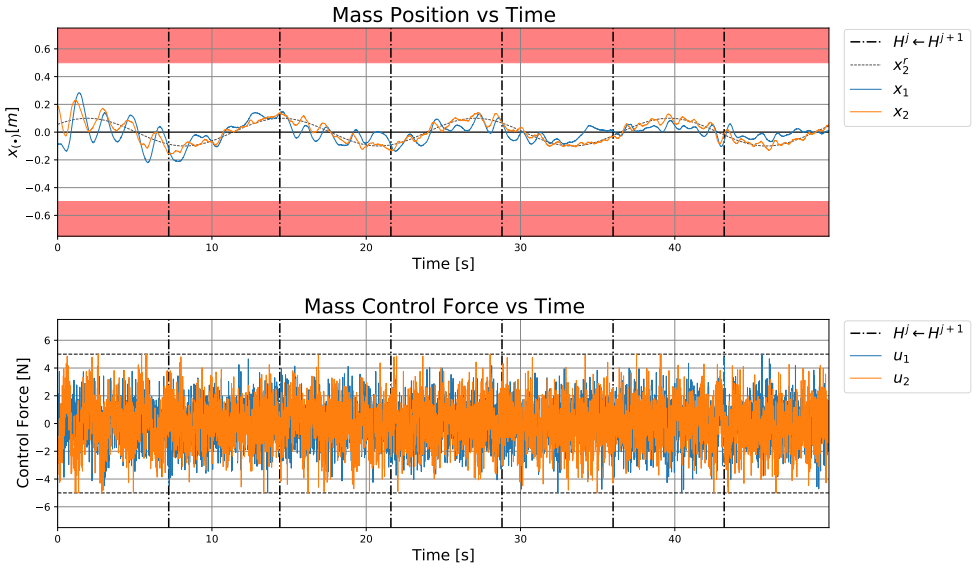


Figure D.10: Evolution of mass positions over time for a 2-MSD system; Online VI; Safe Curriculum Learning (step 2); SHERPA activated; Gaussian sensor noise, $\sigma = 0.0012$; $x_i \in [-0.2, 0.2]$; $\dot{x}_i \in [-0.25, 0.25]$; Red is FSS for x_2.

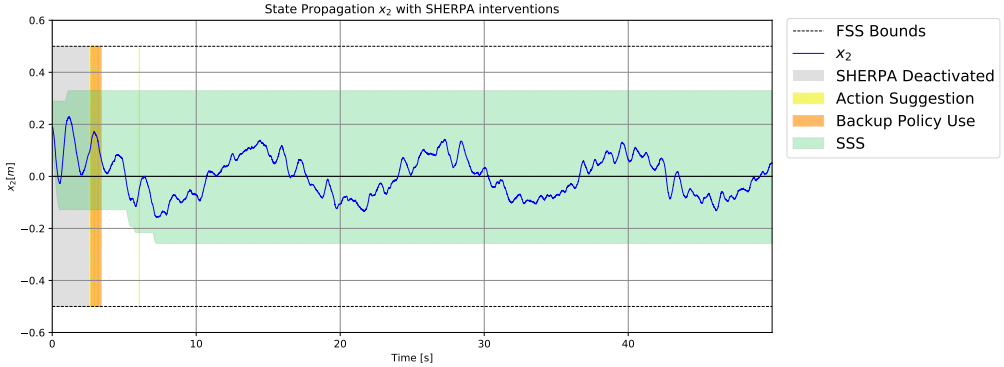


Figure D.11: Evolution of the SSS over time along with SHERPA interventions for the position of the second mass, x_2; Online VI; Safe Curriculum Learning (step 2); SHERPA activated; Gaussian sensor noise, $\sigma = 0.0012$.

Finally, the evolution of absolute parametric uncertainties in the third curricular step are shown in Figure D.12. The concept of sliding window is applied on this estimate evolution, which means that it is update at every time step after the first estimate has been computed. The oscillating uncertainties of $\tilde{F}$ are related to their relatively low values.

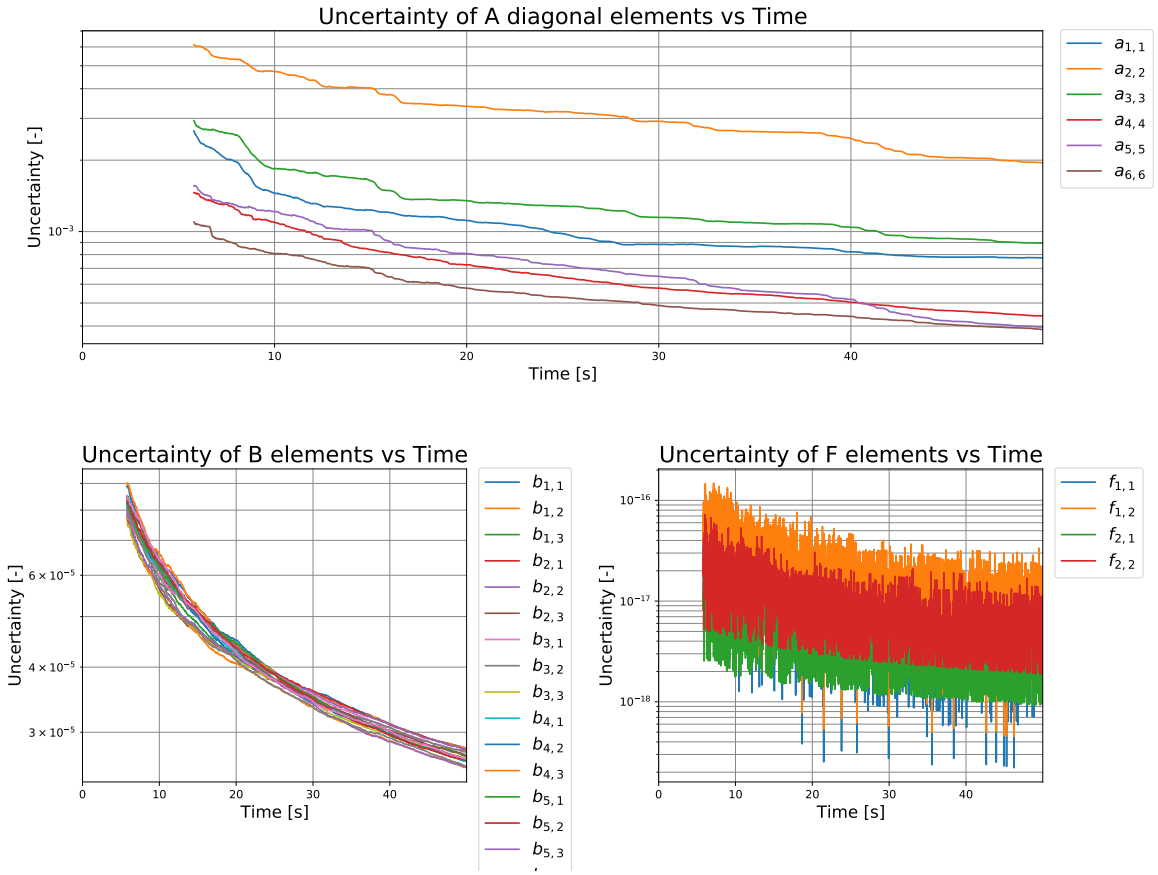


Figure D.12: Evolution of the absolute parametric uncertainty of all matrices forming the uncertain model estimate used in SHERPA, $\tilde{A}$, $\tilde{B}$ and $\tilde{F}$; Online VI; Safe Curriculum Learning (step 3); SHERPA activated; Gaussian sensor noise, $\sigma = 0.0012$.