Sondre Olsen

Safe reinforcement learning for control-affine systems with probabilistic safety constraints

Master's thesis in Cybernetics and Robotics
Supervisor: Professor Jan Tommy Gravdahl
Co-supervisor: Dr. Esten Ingar Grøtli
June 2021

Sondre Olsen

Safe reinforcement learning for control-affine systems with probabilistic safety constraints

Master's thesis in Cybernetics and Robotics
Supervisor: Professor Jan Tommy Gravdahl
Co-supervisor: Dr. Esten Ingar Grøtli
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of Master of Science in Cybernetics and Robotics, and it is the culmination of the work that will conclude my time at the Norwegian University of Science and Technology (NTNU). The work presented in this thesis was carried out during the spring semester of 2021 under the supervision of Jan Tommy Gravdahl and Esten Ingar Grøtli, who have offered valuable advice. Akhil S. Anand has provided feedback and has helped read through a draft of the thesis.

The thesis is inspired by recent advances in learning-based control for safety-critical systems. It is assumed the reader has prior knowledge of linear algebra and control theory, as well as some knowledge of optimization theory. Familiarity with machine learning, and in particular reinforcement learning, will be beneficial, but not necessary as this topic will be introduced.

This thesis is the continuation of a specialization project conducted during the fall semester of 2020. Some important background theory from the specialization project report will be paraphrased or restated in updated form throughout the thesis. Following is a complete list of the material included from the specialization project report:

- Section 2.1.1
- Section 2.4 - Section 2.4.1
- Section 2.5
- Section 2.7 - Section 2.7.3
- Section 2.8

One of the contributions in this thesis, a method for safe exploration, makes use of a matrix variate Gaussian process model based on the work by Dhiman *et al.* [1]. This model was implemented in the Python programming language and model regression was performed using the GPyTorch library [2]. The neural networks used in the deep reinforcement learning algorithm were implemented and trained using the PyTorch library [3].

Acknowledgements

I would like to thank my supervisor from the Department of Engineering Cybernetics, Professor Jan Tommy Gravdahl, as well as my co-supervisor from SINTEF Digital, Dr. Esten Ingar Grøtli, for their advice and guidance. I would also like to thank Akhil S. Anand for his feedback, our motivating discussions throughout the semester, and for proofreading the thesis.

Finally, thanks are also owed to my friends and family, who have offered valuable support.

Sondre Olsen
June 2021

Abstract

Reinforcement learning holds promise to enable autonomous systems to acquire novel skills without human intervention, and recent years have seen significant advances in ways of learning optimal control policies in unknown environments. While many of these algorithms achieve impressive performance, they are typically not concerned with guaranteeing safe operation during learning, which may cause unsafe or harmful behavior in real-world scenarios. Motivated by the importance of safety-critical control in learning-based systems, this thesis introduces a framework for safe learning based on control barrier functions to ensure system safety with high probability.

Many learning-based algorithms for safety-critical control rely on prior knowledge from the environment they are deployed in, or introduce restrictive assumptions on potential control policies. Currently, there is an initiative to unify the flexibility offered by reinforcement learning with the rigorousness of classical control methods, in order to ensure system safety and stability. Methods have been developed that learn subject to safety constraints, and which obtain stochastic model estimates of unknown system dynamics. However, few methods for obtaining less restrictive guarantees of safety for reinforcement learning frameworks exist.

In this thesis, a framework for safety-constrained, model-based reinforcement learning is proposed and evaluated. An exploration scheme for safely learning a Gaussian process model from actively sampled data is introduced. Control barrier functions are utilized to provide probabilistic guarantees of safety while exploring. Further, a method for safety-constrained policy optimization is developed. The stochastic dynamics model found by safe exploration is utilized to produce an episodic framework for learning. From the theoretical framework, a practical version is implemented and its performance is evaluated in simulation.

Sammendrag

Forsterkende læring holder løfte om å gjøre det mulig for autonome systemer å tilegne seg nye ferdigheter uten menneskelig innblanding, og i senere år er det gjort betydelige fremskritt innen måter å lære optimale reguleringsregler i ukjente omgivelser. Selv om mange av disse algoritmene oppnår imponerende ytelse, er de typisk ikke opptatt av å garantere sikker drift under læring, noe som kan forårsake skadelig oppførsel i scenarier i den virkelige verden. Denne oppgaven er motivert av viktigheten av sikkerhetskritisk regulering hos læringsbaserte systemer, og introduserer et rammeverk for sikker læring basert på kontrollbarrierefunksjoner for å sørge for systemsikkerhet med høy sannsynlighet.

Mange læringsbaserte algoritmer for sikkerhetskritisk regulering er avhengige av forhåndskunnskap om omgivelsene de er utplassert i, eller introduserer restriktive antagelser om potensielle reguleringsregler. For tiden finnes det initiativ for å forene fleksibiliteten gitt av forsterkende læring med strengheten til tradisjonell reguleringsteknikk, med den hensikt å sørge for sikkerheten og stabiliteten til systemer. Det har blitt utviklet metoder for læring som er underlagt sikkerhetsbegrensninger, og som innhenter stokastiske modellestimater av en ukjent dynamisk systemmodell. Det finnes derimot få metoder for å oppnå mindre restriktive sikkerhetsgarantier for forsterkningslæringsrammeverk.

I denne oppgaven utformes og evalueres et rammeverk for sikkerhetsbegrenset modellbasert forsterkende læring. En metode for utforskning for å lære en Gaussisk prosess fra aktivt innhentet data blir introdusert. Kontrolbarrierefunksjoner blir benyttet for å gi probabilisitiske sikkerhetsgarantier under utforskning. Videre utvikles en metode for sikkerhetsbegrenset optimalisering av en reguleringsregel. Den stokastiske dynamiske modellen funnet gjennom sikker utforskning utnyttes for å lage et episodisk læringsrammeverk. En praktisk versjon av det teoretiske rammeverket implementeres og evalueres i simulering.

Contents

List of Figures

List of Tables

Abbreviations

CBC Control Barrier Condition. 11

CBF Control Barrier Function. 3

CLC Control Lyapunov Condition. 10

CLF Control Lyapunov Function. 3

ECBF Exponential Control Barrier Function. 12

FFNN Feed Forward Neural Network. 24

GP Gaussian Process. 2

MDP Markov Decision Process. 2

QP Quadratic Program. 38

RL Reinforcement Learning. 1

UCB Upper Confidence Bound. 38

1 | Introduction

1.1 Background and motivation

Reinforcement Learning (RL) is an effective paradigm for learning to control dynamical systems [4]. When a model of the environment is not perfectly known *a priori*, learning-based control methods can provide successful policies for complex tasks and objectives in ways which classical control methods might not be able to. Intelligent agents are, for example, able to learn driving behavior from pure visual input [5, 6], and deep reinforcement learning techniques are utilized to outperform human players in complex, many-state games such as chess and go [7, 8]. Moreover, autonomous systems that learn to observe and interact with their environments have also demonstrated formidable abilities in recent years, especially within areas such as motion planning [9], obstacle avoidance [10], autonomous navigation [11, 12] and robotic locomotion and manipulation [13, 14].

Autonomous systems intended to operate in physical environments must ensure their own safety as well as that of their surroundings. Many strategies for control traditionally rely on fixed rules for behavior, provide pre-computed action sequences or utilize *known* dynamics for specifying online policy updates. While such strategies often work well in supervised scenarios or in cases where the operational conditions rarely change, systems that fail to consider uncertainty, unforeseen changes or *unknown* dynamics can cause inadvertent harm during real-world deployment. This expectation of autonomous systems to safely operate in unstructured environments has, in part, elicited an influx in research on safety-critical, learning-based control methods [15].

Consider, for example, a medical robot performing an ultrasound scan on a patient. This task requires compliant manipulation in an unknown environment, and an autonomous system must safely scan the patient. It is clearly unacceptable for the robot to harm the patient during the procedure or crash into other medical equipment. One safety constraint can, for instance, be defined as a maximum amount of contact force that should be applied to the skin. The system should avoid applying more force in order to remain safe. As the environment dynamics are *a priori* unknown and only an initial model of the soft-body physics is available, the contact forces will be highly uncertain. Therefore, it is not possible for a control algorithm to reason about which actions are safe prior to their execution. In order to guarantee safe control, the algorithm must learn a dynamics model *while* interacting with the environment through a process of safe exploration: data samples should be collected in such a way that the safety-critical constraints are satisfied. As the system adapts its behavior and improves its model of the environment episodically, under the assumption of a safe control scheme, a control policy that is both optimal and does not violate the imposed safety constraints could be synthesized.

Despite their impressive enterprises, most learning-based methods do not take safety guarantees into consideration during learning [15, 16]. They allow the control of dynamical systems in unknown environments, however the *exploration vs. exploitation* trade-off intrinsic to reinforcement learning necessitates the need to try out random actions in order to find optimal ones [4]. A reinforcement learning agent generally depends on randomly exploring its environment in order to improve model accuracy, while simultaneously taking advantage of what it already knows by exploiting actions that maximize cumulative reward over time. In contrast to the example of safe exploration laid out above, many algorithms therefore end up evaluating all possible actions in search of an optimal solution, and in doing so they fail to consider the potential harmful effects of intermediate policies. Random actions cannot necessarily be fed as control inputs to real-world systems on any account, since they may be considered dangerous or cause unsafe behavior. Consequently, most learning-based methods cannot reliably be deployed in safety-critical scenarios.

Several methods have been suggested to guarantee the safety of dynamical systems. Since the notion of safety was first introduced in the form of *program correctness* [17] and later formalized for safety-critical systems [18], many different approaches to safe reinforcement learning have been defined [19], including risk-averse reward specification, transformation of optimization criteria and robust- and constrained Markov Decision Processes (MDPs) [19–22]. Commonly, many of these methods require upfront knowledge of an accurate model of the system dynamics.

Gaussian Processs (GPs) are a type of stochastic process that can be used to learn an approximate model of a system in cases where the true system dynamics are unknown or an accurate model is unavailable [23]. GPs have been utilized by many for modeling uncertain environments [24, 25], including efforts to capture uncertainty in constraints related to system properties [26]. Polymenakos *et al.* [27] propose a method for safe policy search under uncertainty, while others use GPs for safe optimization in model-based reinforcement learning settings to provide better model estimates [28].

Recent years have seen an initiative in research on analyzing and verifying the stability and safety of learning-based methods. Perkins *et al.* [29] switch between *a priori* safe policies, while Uchibe *et al.* [30] utilize a policy gradient search algorithm that enforces active constraints. The method in [30] does not guarantee the safety of a policy at all times, so further efforts have been made to develop higher-dimensional policy search algorithms for constrained MDPs that provide guarantees about policy behavior throughout the entire training period [31, 32].

Lyapunov stability analysis can be used to ensure safe system behavior during learning, in terms of stability guarantees. Several methods apply Lyapunov function theory to verify the stability of *known* dynamics and to safely approximate a region of attraction for *unknown* dynamics [24, 33], while others utilize Control Lyapunov Functions (CLFs) to enforce asymptotic system stability [34]. Berkenkamp *et al.* [28] propose a framework for model-based reinforcement learning that iteratively verifies the safety of a second-order dynamical system using Lyapunov's method, which yields provable high-probability, closed-loop stability certificates.

Guarantees of safety in terms of constraint satisfaction have also received significant attention lately. Control Barrier Functions (CBFs) have been utilized to define a more permissive notion of safety based on barrier certificates, which enforce the safety of a closed-loop system through the invariance of a safe set of system states. Many have used CBFs to define safety constraints during learning [35–37], while Khojasteh *et al.* [38] propose a Bayesian framework for learning unknown dynamics and imposing probabilistic safety constraints when optimizing system behavior. A recent publication by Dhiman *et al.* [1] builds upon the safe learning framework from [38], and the authors utilize a Gaussian process distribution of the unknown system dynamics to formulate probabilistic constraints in order to enforce safety under uncertainty.

1.2 Problem formulation

The aim of this thesis is to investigate the use of control barrier functions as a tool for ensuring the safety of an online, model-based reinforcement learning framework. Dhiman *et al.* [1] learn a posterior estimate of unknown system dynamics by Gaussian process regression, but limit their set of potential control policies to those that are synthesized from solving a safety-constrained quadratic program. Berkenkamp *et al.* [28], on the other hand, perform online, model-based reinforcement learning and solely evaluate actions guaranteed to be safe with regards to an uncertain model of the system dynamics during the exploration phase. However, they utilize control Lyapunov functions and analyze the stability of their closed-loop system, which in turn means their method is based on a more restrictive notion of safety than the one afforded by control barrier functions.

Based on these considerations, a question related to the safety of learning-based control methods is posed, which the work presented in the following sections attempt to answer:

Can control barrier functions be used to provide high-probability safety guarantees for a model-based reinforcement learning algorithm in an unknown environment?

1.3 Contributions

The following contributions are made in an attempt to find a solution to the above problem:

1. A method for safe exploration is developed, which aims to only sample regions of the state space that are guaranteed to be safe with high probability in order to improve a model of *a priori* unknown system dynamics.

2. An algorithm for model-based reinforcement learning is proposed, which uses safe exploration to inform a model of unknown dynamics, and which utilizes control barrier functions in order to synthesize a control policy that is optimized subject to probabilistic safety constraints.

1.4 Thesis outline

The thesis is organized into six main parts, and is comprised of the following sections:

Chapter 2 presents relevant background on theoretical concepts, including central theory related to nonlinear systems, the notions of system stability and safety and control Lyapunov- and barrier functions, Gaussian process regression, reinforcement learning, and deep learning.

In Chapter 3, the method for safe learning with CBFs is explained, and the high-probability guarantees of the safety of the system during exploration is showed. The proposed algorithm is inspired by the model-based learning framework from Berkenkamp *et al.*, and utilizes probabilistic control barrier conditions derived in Dhiman *et al.*

Further, in Chapter 4, simulation experiments on two second-order system are conducted, to test a practical implementation of the safe learning algorithm.

Then, in Chapter 5, both the practical and theoretical results are discussed and evaluated, and the feasibility and limitations of the theoretical algorithm is discussed. Finally, Chapter 6 concludes the thesis.

2 | Background theory

In this chapter, necessary background theory is presented. First, an introduction to some relevant aspects of dynamical systems modeling and control are given. Next, control Lyapunov and control barrier functions are presented, and their relation to system safety is explained. Further, key aspects of reinforcement learning are covered, with a focus on model-based, policy gradient methods. Lastly, a few notable concepts from deep learning are briefly presented.

Material in Section 2.1.1, Section 2.4 - Section 2.4.1, Section 2.5, Section 2.7 - Section 2.7.3 and Section 2.8 was included in the specialization project report [39], and is restated here in updated form.

2.1 Dynamical systems

A dynamical system describes the behavior of a system over time. Consider a time-varying nonlinear system for which the system state and input are represented by a state variable $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^n$ and a control action variable $\boldsymbol{u} \in \mathcal{U} \subset \mathbb{R}^m$, respectively. A model of the system dynamics that describes the time evolution of the system state, $f : \mathbb{R}^n \to \mathbb{R}^n$, can then be expressed as

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}). \tag{2.1}$$

Control-affine systems are a special class of nonlinear dynamical systems, characterized by being affine with regards to the system control actions. Considering a *drift term* $f : \mathbb{R}^n \to \mathbb{R}^n$ and an *input gain* $g : \mathbb{R}^n \to \mathbb{R}^{n \times m}$, a model describing the time evolution of the system state of a control-affine system can then be expressed as

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u} = [f(\boldsymbol{x}) \;\; g(\boldsymbol{x})] \begin{bmatrix} 1 \\ \boldsymbol{u} \end{bmatrix} := F(\boldsymbol{x})\underline{\boldsymbol{u}}. \tag{2.2}$$

Throughout the thesis, the dynamical systems to be controlled are assumed to be control-affine. Furthermore, f and g are assumed to be locally Lipschitz, which is a standard assumption in reinforcement learning [34, 40]. Also, an initial state $\boldsymbol{x}_0$ is assumed to be known in advance of the initialization of the control scheme.

In general, systems are controlled by applying actions at every time step. The sequence of actions that are applied to the system can be denoted by a control policy,

$$\boldsymbol{u} = \pi(\boldsymbol{x}), \tag{2.3}$$

such that for a given state $\boldsymbol{x}_t$ at time step t, the policy π selects which action $\boldsymbol{u}_t$ to apply. The resulting closed-loop system can be written on shorthand form as $f_\pi(\boldsymbol{x}) = f(\boldsymbol{x}, \pi(\boldsymbol{x}))$. In cases where the policy is parameterized by a vector of parameters $\boldsymbol{\theta} \in \Theta \subset \mathbb{R}^d$, such as in Section 2.8.1, this can be explicitly expressed by denoting the policy as $\pi_{\boldsymbol{\theta}}(\boldsymbol{x})$.

2.1.1 Nonlinear control

The *control problem* can be defined as applying control actions that lead a dynamical system to exhibit desirable behavior or perform a desired task. This is done by finding a suitable control policy. Some well-known examples of control tasks are system stabilization, reference tracking and disturbance rejection. Nonlinear systems, which exhibit more complex behavior than linear systems due to their inherently richer dynamics, are thereby more relevant for non-trivial control tasks and more interesting for qualitative and quantitative analysis [41].

System stabilization is a control task that motivates safety for dynamical systems. By finding a policy that drives a system to some stable state, an equilibrium point of the system is stabilized. An equilibrium point $\boldsymbol{x}$ is stable if all solutions that start near $\boldsymbol{x}$ remain nearby forever. A formal definition of stability is provided in Definition 1, where $\boldsymbol{x}$ is considered an equilibrium point of (2.1), such that $f(\boldsymbol{x}) = \mathbf{0}$ and f is assumed to be locally Lipschitz.

Definition 1 (Definition 4.1 from Khalil [41]).
The equilibrium point $\boldsymbol{x} = \mathbf{0}$ of (2.1) is

- *stable if, for each $\epsilon > 0$, there is $\delta = \delta(\epsilon) > 0$ such that*

$$\|\boldsymbol{x}(0)\| < \delta \implies \|\boldsymbol{x}(t)\| < \epsilon, \ \forall\, t \geq 0 \tag{2.4}$$

- *unstable if it is not stable.*
- *asymptotically stable if it is stable and δ can be chosen such that*

$$\|\boldsymbol{x}(0)\| < \delta \implies \lim_{t\to\infty} \|\boldsymbol{x}(t)\| = \mathbf{0} \tag{2.5}$$

The equilibrium point is defined at the origin without loss of generality [41], as any equilibrium point can be shifted to the origin by a change in variables. The function f in Definition 1 is assumed to satisfy the Lipschitz condition, which for all points $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ states that,

$$\|f(\boldsymbol{x}) - f(\boldsymbol{y})\| \leq L\|\boldsymbol{x} - \boldsymbol{y}\| \tag{2.6}$$

for a positive constant L, called the Lipschitz constant [41].

Verifying the stability of a system can prove challenging to do in general. Directly using the condition (2.5) in Definition 1 to show the asymptotic stability of a nonlinear system requires reasoning about all trajectories around the equilibrium point infinitely long into the future. Under some additional conditions however, Lyapunov's stability theorem provides a convenient way to determine the stability of Lipschitz systems without needing to study all trajectories. Utilizing a Lyapunov function V, then by Theorem 1, the asymptotic stability of an equilibrium point of a continuous nonlinear system can be verified.

Theorem 1 (Theorem 4.2 from Khalil [41]).
Let $\boldsymbol{x} = \boldsymbol{0}$ be an equilibrium point for (2.1). *Let $V : \mathbb{R}^n \longrightarrow \mathbb{R}$ be a continuously differentiable function such that*

$$V(\boldsymbol{0}) = 0 \text{ and } V(\boldsymbol{x}) > 0 \;\forall\; \boldsymbol{x} \neq \boldsymbol{0} \tag{2.7}$$

$$\|\boldsymbol{x}\| \rightarrow \infty \implies V(\boldsymbol{x}) \rightarrow \infty \tag{2.8}$$

$$\dot{V}(\boldsymbol{x}) < 0, \;\forall\; \boldsymbol{x} \neq \boldsymbol{0} \tag{2.9}$$

Then, $\boldsymbol{x} = \boldsymbol{0}$ is globally asymptotically stable.

In a nonlinear context, Theorem 1 can be interpreted equivalently to finding a control policy, or more specifically a feedback control law, $\pi(\boldsymbol{x})$ that drives a positive definite function V to zero [34]. In the case of control-affine systems on the form of (2.2), V is a Lyapunov function candidate if there exists a control policy $\boldsymbol{u} = \pi(\boldsymbol{x})$ that provides system inputs in a way which makes the time derivative of the Lyapunov function negative, such that $\nabla V(\boldsymbol{x}) \cdot (f(\boldsymbol{x}) + g(\boldsymbol{x})\pi(\boldsymbol{x})) < 0$. That is, for some class $\mathcal{K}$ function α [34],

$$\dot{V}(\boldsymbol{x}, \pi(\boldsymbol{x})) \leq -\alpha V(\boldsymbol{x}). \tag{2.10}$$

In (2.10), the time derivative of the Lyapunov function can be expressed using the Lie derivatives of V along f and g,

$$\dot{V}(\boldsymbol{x}, \pi(\boldsymbol{x})) = \mathcal{L}_f V(\boldsymbol{x}) + \mathcal{L}_g V(\boldsymbol{x})\pi(\boldsymbol{x}). \tag{2.11}$$

As can be seen from Theorem 1 and (2.10), the process of finding a Lyapunov function serves as a convenient way to verify the stability of a closed-loop system for a given control policy. However, in the cases where such a policy has not been selected in advance, this method of Lyapunov function stability verification will not suffice. This has motivated the development of control Lyapunov functions [34], partly through an observation that only the *existence* of a control policy which results in the equality (2.10) is needed in order to verify system stability [41, 42].

2.2 Control Lyapunov functions

Control Lyapunov functions extend the theory behind Lyapunov's method for stability verification to systems where no explicit control policy has already been constructed. Similarly to the definition of the function in (2.10), the function V is a CLF for the system in (2.2) if it is a Lyapunov function candidate for which there exists a possible choice of control actions $\boldsymbol{u}$ that renders its time derivative negative. In other terms, there must exist a control policy providing inputs such that $\nabla V(\boldsymbol{x}) \cdot (f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u}) < 0$ [1, 34]. A formal definition of control Lyapunov functions is provided in Definition 2.

Definition 2 (Definition 1 from Dhiman *et al.* [1], paraphrased).
A function $V : \mathcal{X} \in \mathbb{R}^n \to \mathbb{R}$ is a CLF for the system in (2.2) *if*

$$V(\boldsymbol{x}) > 0 \;\forall\; \boldsymbol{x} \in \mathcal{X} \setminus \{\boldsymbol{0}\}, \;\; \textit{and}\; V(\boldsymbol{0}) = 0, \tag{2.12}$$

and if it satisfies

$$\inf_{u \in \mathcal{U}} [\mathcal{L}_f V(\boldsymbol{x}) + \mathcal{L}_g V(\boldsymbol{x})\boldsymbol{u} + \alpha V(\boldsymbol{x})] \leq 0 \;\forall\; \boldsymbol{x} \in \mathcal{X} \tag{2.13}$$

for some class $\mathcal{K}$ function α.

The left-hand side of (2.13), which by Definition 2 serves as a stability condition, can be defined as a Control Lyapunov Condition (CLC) [1], such that

$$CLC(\boldsymbol{x}, \boldsymbol{u}) := \mathcal{L}_f V(\boldsymbol{x}) + \mathcal{L}_g V(\boldsymbol{x})\boldsymbol{u} + \alpha V(\boldsymbol{x}). \tag{2.14}$$

The utilization of the control Lyapunov condition in (2.14) leads to a central result for a way which control Lyapunov functions provide a sufficient condition for stability [34].

Theorem 2 (Theorem 1 from Ames *et al.* [34], paraphrased).
If there exists a control Lyapunov function V for the system in (2.2)*, then any Lipschitz continuous control policy*

$$\pi(\boldsymbol{x}) \in \{\boldsymbol{u} \in \mathcal{U} \mid CLC(\boldsymbol{x}, \boldsymbol{u}) \leq 0\} \tag{2.15}$$

asymptotically stabilizes the system.

Control Lyapunov functions are utilized to find control policies that satisfy the condition in Theorem 2. Such controllers, which can be found through synthesis or otherwise, can enforce stability on the closed-loop system dynamics they are applied to.

2.3 Control barrier functions

Control barrier functions enforce safety. They build upon the foundation of the definitions pertaining control Lyapunov functions, and the safety enforced by CBFs can be considered a "dual" to the stability enforced by CLFs [34]. In contrast to Lyapunov stability theory, where closed-loop stability is ensured by driving a system to an asymptotically stable state, *barrier certificates* are utilized in the context of safety-critical control to encode safety through set invariance [43].

More specifically, safety can be enforced by ensuring invariance of a safe set. Consider the set

$$C = \{\boldsymbol{x} \in \mathcal{X} \mid h(\boldsymbol{x}) \geq 0\} \tag{2.16}$$

to be a superlevel set of the continuously differentiable function $h : \mathbb{R}^n \rightarrow \mathbb{R}$, containing all safe system states. The system in (2.2) is defined as being safe with respect to C, if C is rendered forward invariant by the existence of h [34].

Analogous to the condition for control Lyapunov functions in Definition 2, h is considered a control barrier function if there exists a possible choice of control actions which renders the time derivative of h positive. The formal definition is provided in Definition 3.

Definition 3 (Definition 2 from Dhiman *et al.* [1], paraphrased).
A function $h : \mathcal{X} \in \mathbb{R}^n \rightarrow \mathbb{R}$ is a CBF for the system in (2.2) *if*

$$\sup_{u \in \mathcal{U}} [\mathcal{L}_f h(\boldsymbol{x}) + \mathcal{L}_g h(\boldsymbol{x})\boldsymbol{u} + \alpha h(\boldsymbol{x})] \geq 0 \; \forall \; \boldsymbol{x} \in \mathcal{X} \tag{2.17}$$

for some class $\mathcal{K}_\infty$ function α.

The left-hand side of (2.17), which by Definition 3 serves as a safety condition, can then be defined as a Control Barrier Condition (CBC) [1], such that

$$\mathit{CBC}(\boldsymbol{x}, \boldsymbol{u}) := \mathcal{L}_f h(\boldsymbol{x}) + \mathcal{L}_g h(\boldsymbol{x})\boldsymbol{u} + \alpha h(\boldsymbol{x}). \tag{2.18}$$

The implication that the set C is forward invariant means that for an initial state inside the safe set, all states propagated through the system forward in time stay inside the safe set. Effectively, if $\boldsymbol{x}(t_0) = \boldsymbol{x}_0 \in C$ then forward invariance implies that $\boldsymbol{x}(t) = \boldsymbol{x} \in C \; \forall \; t \geq t_0$.

Equivalent to the safe states, any state in the *un*safe region complimentary to the safe region defined by the CBF is considered unsafe. In practical utilization of control barrier functions, h is sometimes defined as the complement function to the set of unsafe states.

As shown in Ames *et al.* [34] and reformulated in [1], a sufficient condition for safety can then be expressed based on Definition 3.

Theorem 3 (Proposition 2 from Dhiman *et al.* [1], paraphrased)**.**
If h is a control barrier function and $\nabla h(\boldsymbol{x}) \neq 0 \; \forall \; \boldsymbol{x}$ when $h(\boldsymbol{x}) = 0$, then any Lipschitz continuous control policy

$$\pi(\boldsymbol{x}) \in \{\boldsymbol{u} \in \mathcal{U} \mid CBC(\boldsymbol{x}, \boldsymbol{u}) \geq 0\} \tag{2.19}$$

renders the system in (2.2) *safe with respect to the set C in* (2.16).

2.3.1 Exponential control barrier functions

Exponential Control Barrier Functions (ECBFs) extend the definition of control barrier functions in Definition 3, and enables control barrier conditions for systems of an arbitrary relative degree to be considered. This opens up the possibility of using control barrier functions to enforce safety-critical constraints on a wider range of nonlinear control systems. Definition 3 relies on the assumption that the function h is of relative degree $r = 1$, which means that the first-degree time derivative of h depends on the system control input. If h as given in Definition 3 has $r > 1$, then $\mathcal{L}_g h(\boldsymbol{x}) = 0$ and the set of admissible control inputs trivially equals either $\mathcal{U}$ or $\emptyset$ [44]. This can sometimes be restrictive, for example in many robotic applications where the input does not necessarily appear directly in the first derivative of the system state but rather as a function of some configuration variable [34, 44].

The assumption that the relative degree of the system is known in advance can also be argued to be limiting in practice, but since the CBF is likely designed with a specific application in mind, it natural to assume knowledge of the order of the system. In many robotic applications, for instance, the functionality of the system may induce the order of the relative degree, while the parameters and any interactions or movements remain unknown.

For higher-order relative degree systems with $r \geq 2$, it is now assumed that h is an r-times differentiable function, such that

$$h^{(r)}(\boldsymbol{x}, \boldsymbol{u}) = \mathcal{L}_f^{(r)} h(\boldsymbol{x}) + \mathcal{L}_g \mathcal{L}_f^{(r-1)} h(\boldsymbol{x}) \boldsymbol{u} \tag{2.20}$$

is the r-th time derivative of h. Then h can be described as the output of a time-invariant linear system

$$\dot{\boldsymbol{\eta}}(\boldsymbol{x}, \boldsymbol{u}) = \mathfrak{F} \boldsymbol{\eta}(\boldsymbol{x}) + \mathfrak{G} \boldsymbol{u} \quad , \quad h(\boldsymbol{x}) = \boldsymbol{c}^T \boldsymbol{\eta}(\boldsymbol{x}) \tag{2.21}$$

where $\boldsymbol{c} := [1, 0, ..., 0]^T \in \mathbb{R}^r$ and by defining

$$\boldsymbol{\eta}(\boldsymbol{x}) := \begin{bmatrix} h(\boldsymbol{x}) \\ \mathcal{L}_f h(\boldsymbol{x}) \\ \vdots \\ \mathcal{L}_f^{(r-1)} h(\boldsymbol{x}) \end{bmatrix} \quad , \quad \mathfrak{F} := \begin{bmatrix} 0 & 1 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 1 \\ 0 & 0 & \dots & 0 \end{bmatrix} \quad , \quad \mathfrak{G} := \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}. \tag{2.22}$$

Based on (2.21), an exponential control barrier function is formally defined in Definition 4 [34, 45].

Definition 4 (Definition 4 from Dhiman *et al.* [1], paraphrased).
An r-times, continuously differentiable function $h : \mathcal{X} \in \mathbb{R}^n \longrightarrow \mathbb{R}$ *is an ECBF for the system in* (2.2) *if there exists a vector* $\boldsymbol{k}_\alpha \in \mathbb{R}^r$ *such that the r-th order control barrier condition* $CBC^{(r)}(\boldsymbol{x}, \boldsymbol{u}) := \mathcal{L}_f^{(r)} h(\boldsymbol{x}) + \mathcal{L}_g \mathcal{L}_f^{(r-1)} h(\boldsymbol{x}) \boldsymbol{u} + \boldsymbol{k}_\alpha \eta(\boldsymbol{x})$ *satisfies*

$$\sup_{u \in \mathcal{U}} [CBC^{(r)}(\boldsymbol{x}, \boldsymbol{u})] \geq 0 \; \forall \; \boldsymbol{x} \in \mathcal{X}, \tag{2.23}$$

and

$$h(\boldsymbol{x}(t)) \geq \boldsymbol{c}^T \eta(\boldsymbol{x}_0) e^{(\mathfrak{F} - \mathfrak{G} \boldsymbol{k}_\alpha)t} \geq 0 \;\; \textit{whenever} \;\; h(\boldsymbol{x}_0) \geq 0. \tag{2.24}$$

A similar condition for safety as in Theorem 3 exists, so that for a $\boldsymbol{k}_\alpha$ that satisfies the properties outlined in Definition 4, any control policy which yields control outputs that ensure $CBC^{(r)}(\boldsymbol{x}, \boldsymbol{u}) \geq 0$ will also ensure the safety of a system with regards to the safe set C in (2.16) [45].

2.4 Stochastic processes

A stochastic process describes the evolution of state values over time, in general much like the dynamical systems introduced in (2.1). The way each state depends on previous states and applied actions however, are through a conditional probability distribution [46].

Stochastic processes are considered collections of random variables, and are often interpreted as a time series of random events indexed by a discrete or continuous time variable [47]. A dynamical system can be interpreted as a stochastic process if the transition from one state to the next is not deterministic, but instead probabilistic and guided by statistical noise.

A version of the dynamical system in (2.1) can be made stochastic by representing the uncertainty in the transition from the current state to the next as independent, identically distributed zero-mean input noise to the system through the stochastic variable $\boldsymbol{\nu}$, so that

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\nu}). \tag{2.25}$$

2.4.1 Markov chains

Markov chains are simple stochastic processes made up of a transition probability distribution which is conditional only on the system states. For a current system state $\boldsymbol{x}_t$ at time t and a given set of previous state values, $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_{t-1}\}$, a Markov chain then defines the transition probability of moving to the next state by the equation

$$P(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_{t-1}, \boldsymbol{x}_t) = P(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t). \tag{2.26}$$

As can be interpreted from (2.26), the conditional distribution of the next state is dependent only on the current state and independent on previous states up until that point in time [47]. This property is called the Markov property, and all systems for which the property hold are labeled *Markovian*. In essence, the future states of any such Markovian system are characterized solely by the present state of the system, since all information about the past effectively is contained in it.

2.4.2 Chebyshev's inequality

Chebyshev's inequality is an inequality used in probability theory to impose a bound on the tail probability of a random variable [48]. It can be utilized as a bound of the *concentration of measure* to analyze the behavior of a probabilistic algorithm or stochastic system [49]. Chebyshev's inequality is derived from the more general Markov's inequality. Given a random variable X and a non-decreasing function $g : \mathbb{R}^+ \to \mathbb{R}^+$, then Markov's inequality states that

$$P(X \geq \zeta) \leq \frac{E\{g(X)\}}{g(\zeta)} \quad \forall \quad \zeta > 0. \tag{2.27}$$

The expression in (2.27) gives an upper bound on the probability of the random variable X, and imposes a limit on how far X can deviate from its expectation [49]. Using (2.27) and given the mean, $E\{X\}$, and variance, $Var\{X\}$, of X, it can be shown as in Alsmeyer [48], that

$$P(X - E\{X\} \geq \zeta) \leq \frac{Var\{X\}}{Var\{X\} + \zeta^2} \quad \forall \quad \zeta > 0. \tag{2.28}$$

The inequality in (2.28) is a one-sided Chebyshev's inequality, and also sometimes known as Cantelli's inequality.

2.5 Gaussian processes

Gaussian processes (GPs) are a type of stochastic process, where the collection of random variables which make up the process are jointly Gaussian distributed [23]. Similarly to other stochastic processes, Gaussian processes are often defined over time. It is a non-parametric method which can be used to statistically model a dynamical system. When using GPs to make predictions about system states or optimal values of some model, inferring a distribution is done over the entire function describing the system, since prior distributions are defined over the space of continuous functions [46]. Conversely, parametric methods like the maximum likelyhood estimator in Section 2.8.1 infers a distribution over the *parameters* of a function and not the function itself.

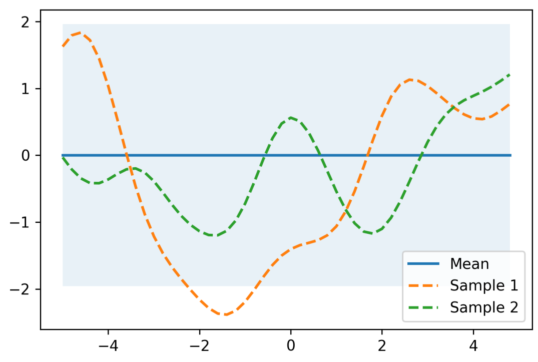
The dynamics of a Gaussian process are completely specified by its first- and second-order moments, and defining mean and covariance functions for a GP will thereby fully describe it [50]. The prior mean and covariance functions of a Gaussian process g can be expressed as the expected mean μ and covariance k functions,

$$\begin{aligned} \mu(\boldsymbol{x}) &= \mathbb{E}\{g(\boldsymbol{x})\} \\ k(\boldsymbol{x}, \boldsymbol{x}') &= \mathbb{E}\{(g(\boldsymbol{x}) - \mu(\boldsymbol{x}))(g(\boldsymbol{x}') - \mu(\boldsymbol{x}'))\}, \end{aligned} \tag{2.29}$$

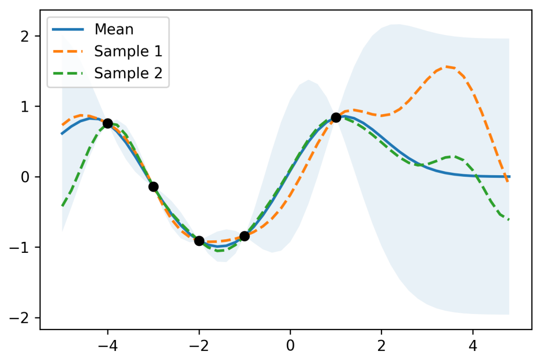
where k denotes a kernel function that defines the covariance between two function inputs $\boldsymbol{x}$ and $\boldsymbol{x}'$. The kernel function encodes information about the distribution over the unknown function, and can take on various forms. Common kernels are linear kernels, squared exponential function kernels and the stationary class of *Mátern* kernels [51]. After having established the parameterization in (2.29), the Gaussian process can be expressed as a joint normal distribution

$$g(\boldsymbol{x}) \sim \mathcal{N}(\mu(\boldsymbol{x}), K(\boldsymbol{x}, \boldsymbol{x})), \tag{2.30}$$

where K is the covariance matrix corresponding to the element-wise covariance functions k for a number of points, such that $[K(\boldsymbol{x}, \boldsymbol{x})]_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Given some input points $\boldsymbol{x}^*$, then a realization of the prior distribution over functions (2.30) can be found. An example of two such prior function samples can be seen in Figure 2.1a.



(a) Gaussian process prior.



(b) Gaussian process posterior.

Figure 2.1: Gaussian process regression. In both figures the shaded blue area denotes a 96 % confidence interval region around the mean.

In most cases, for simplicity, the prior mean function is set to be zero by letting $\mu(\boldsymbol{x}) = \boldsymbol{0}$, so that (2.30) turns into

$$g(\boldsymbol{x}) \sim \mathcal{N}(\boldsymbol{0}, K(\boldsymbol{x}, \boldsymbol{x})). \tag{2.31}$$

This is rarely limiting in practice, since the mean function of the posterior defined by (2.32) is not necessarily zero [23].

The Gaussian process prior can be conditioned on observations of data sampled from the process being modeled in order to generate a Gaussian process posterior. Assuming noisy observations with a zero-mean Gaussian distribution $\boldsymbol{y} \sim \mathcal{N}(0, \boldsymbol{I}\sigma^2)$, then, as shown in Rasmussen *et al.* [23], the posterior mean and covariance functions of the Gaussian process g in (2.31), can be expressed as

$$\begin{gathered} \mu_k(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x}, \boldsymbol{y})(K + \boldsymbol{I}\sigma^2)^{-1}\boldsymbol{y} \\ k_k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}') - \boldsymbol{k}(\boldsymbol{x}, \boldsymbol{y})(K + \boldsymbol{I}\sigma^2)^{-1}\boldsymbol{k}^T(\boldsymbol{x}', \boldsymbol{y}), \end{gathered} \tag{2.32}$$

where $\boldsymbol{k}(\boldsymbol{x}, \boldsymbol{y})$ is a vector containing the covariances between the input and all observed data points $\boldsymbol{y}$ [23]. Figure 2.1 shows an example of both a Gaussian process prior and posterior using a squared exponential function kernel. Figure 2.1a shows an uninformed prior mean. Figure 2.1b shows how the posterior mean is informed by four data points sampled from the underlying process, and illustrates how the confidence intervals are significantly reduced closer to the observations.

2.6 Matrix variate Gaussian processes

Matrix variate Gaussian process regression is a method for inferring the posterior distribution over the *drift* and *gain* terms from the control-affine dynamics in (2.2).

The matrix variate Gaussian distribution can be defined as a 3-parameter distribution describing a random *matrix* variable $\boldsymbol{X} \in \mathbb{R}^{n \times m}$,

$$p(\boldsymbol{X}; \boldsymbol{M}, \boldsymbol{A}, \boldsymbol{B}) := \frac{\exp(-\frac{1}{2}\mathbf{tr}[\boldsymbol{B}^{-1}(\boldsymbol{X} - \boldsymbol{M})^T \boldsymbol{A}(\boldsymbol{X} - \boldsymbol{M})])}{(2\pi)^{\frac{nm}{2}} \det(\boldsymbol{A})^{\frac{m}{2}} \det(\boldsymbol{B})^{\frac{n}{2}}}, \tag{2.33}$$

where $\boldsymbol{M}$ is the mean, and where $\boldsymbol{A}$ and $\boldsymbol{B}$ describe the covariance matrices of the the rows and columns of $\boldsymbol{X}$.

The random matrix variable can then, generally, be expressed as the matrix variate normal distribution

$$\boldsymbol{X} \sim \mathcal{MN}(\boldsymbol{M}, \boldsymbol{A}, \boldsymbol{B}). \tag{2.34}$$

A *state-control* dataset, $(\boldsymbol{X}_{1:k}, \boldsymbol{U}_{1:k}, \dot{\boldsymbol{X}}_{1:k})$, of k samples can be used as training data for the GP regression, where $\boldsymbol{X}_{1:k} := [\boldsymbol{x}_t, ..., \boldsymbol{x}_{t_k}]$, $\boldsymbol{U}_{1:k} := [\boldsymbol{u}_t, ..., \boldsymbol{u}_{t_k}]$ and $\dot{\boldsymbol{X}}_{1:k} := [\dot{\boldsymbol{x}}_t, ..., \dot{\boldsymbol{x}}_{t_k}]$. It is assumed only $\dot{\boldsymbol{X}}_{1:k}$ is corrupted by zero-mean Gaussian noise. Dhiman *et al.* [1] then propose a vectorized decomposition of the Gaussian process prior for the control-affine system,

$$\text{vec}(F(\boldsymbol{x})) \sim \mathcal{GP}(\text{vec}(\boldsymbol{M}_0(\boldsymbol{x}), \boldsymbol{B}_0(\boldsymbol{x}, \boldsymbol{x}') \otimes \boldsymbol{A}), \tag{2.35}$$

where $\text{vec}(F(\boldsymbol{x}))$ is obtained by stacking the columns of F in (2.2). Furthermore, based on (2.35) and for a given control action, it can be shown that the posterior distribution over F conditioned on k samples is expressed as

$$F_k(\boldsymbol{x})\underline{\boldsymbol{u}} \sim \mathcal{GP}(\boldsymbol{M}_k(\boldsymbol{x})\underline{\boldsymbol{u}}, \underline{\boldsymbol{u}}^T \boldsymbol{B}_k(\boldsymbol{x}, \boldsymbol{x}')\underline{\boldsymbol{u}} \otimes \boldsymbol{A}). \tag{2.36}$$

The posterior mean and covariance matrix functions are defined as

$$\begin{aligned} \boldsymbol{M}_k(\boldsymbol{x}) &:= \boldsymbol{M}_0(\boldsymbol{x}) + (\dot{\boldsymbol{X}}_{1:k} - \mathcal{M}_{1:k}\underline{\mathcal{U}}_{1:k})(\underline{\mathcal{U}}_{1:k}\mathcal{B}_{1:k}(\boldsymbol{x}))^\dagger \\ \boldsymbol{B}_k(\boldsymbol{x}, \boldsymbol{x}') &:= \boldsymbol{B}_0(\boldsymbol{x}, \boldsymbol{x}') + \mathcal{B}_{1:k}(\boldsymbol{x})\underline{\mathcal{U}}_{1:k}(\underline{\mathcal{U}}_{1:k}\mathcal{B}_{1:k}(\boldsymbol{x}'))^\dagger, \end{aligned} \tag{2.37}$$

where $\mathcal{M}_{1:k} = [\boldsymbol{M}_0(\boldsymbol{x}_1), .., \boldsymbol{M}_0(\boldsymbol{x}_k)]$, $\mathcal{B}_{1:k}(\boldsymbol{x}) = [\boldsymbol{B}_0(\boldsymbol{x}, \boldsymbol{x}_1), ..., \boldsymbol{B}_0(\boldsymbol{x}, \boldsymbol{x}_k)]$ and $\underline{\mathcal{U}}_{1:k} = Diag\{\underline{\boldsymbol{u}}_1, ..., \underline{\boldsymbol{u}}_k\}$. Lastly, using Schur's complement and defining $\left[\mathcal{B}_{1:k}^{1:k}\right]_{i,j} := \boldsymbol{B}_0(\boldsymbol{x}_i, \boldsymbol{x}_j)$, then [1]

$$(\underline{\mathcal{U}}_{1:k}\mathcal{B}_{1:k}(\boldsymbol{x}))^\dagger := (\underline{\mathcal{U}}_{1:k}^T \mathcal{B}_{1:k}^{1:k} \underline{\mathcal{U}}_{1:k} + \boldsymbol{I}_k \sigma^2)^{-1} \underline{\mathcal{U}}_{1:k}^T \mathcal{B}_{1:k}^T(\boldsymbol{x}) \quad , \quad \sigma > 0. \tag{2.38}$$

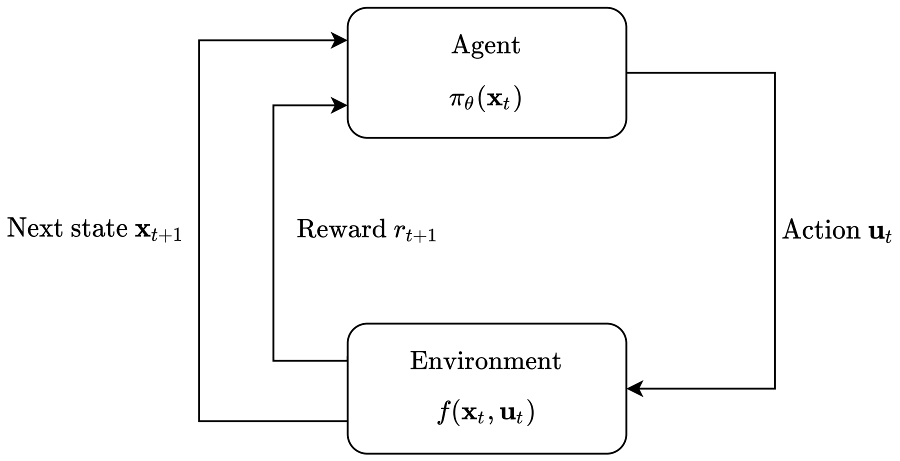

Agent
$\pi_\theta(\mathbf{x}_t)$
Next state $\mathbf{x}_{t+1}$
Reward r_{t+1}
Action $\mathbf{u}_t$
Environment
$f(\mathbf{x}_t, \mathbf{u}_t)$

Figure 2.2: Agent-environment interaction in reinforcement learning.

2.7 Reinforcement learning

Reinforcement learning is a machine learning framework concerned with how a decision-making agent interacts with its environment in order to learn an optimal control policy. The agent performs actions and learns, from experience, how to act in order to improve its performance on a certain task. Performance is defined based on *reward*, and the aim of the agent is to receive maximum amount of reward over time.

A reinforcement learning system consists of four main elements: a control policy, an external reward signal, a value function, which estimates the cumulative future reward an agent can expect following a certain policy, and optionally a model of the environment the agent operates in [4]. The agent-environment interaction loop describing an RL problem can be seen in Figure 2.2. At time step t an agent, represented by its policy π, selects an action $\boldsymbol{u}_t$ to interact with an environment, represented by a dynamical system model f, based on the current state $\boldsymbol{x}_t$. The environment, which to the agent itself in general is unknown, responds dynamically to the applied action $\boldsymbol{u}_t$ by feeding back to the agent the subsequent state $\boldsymbol{x}_{t+1}$ at the next time step $t + 1$, as well as a scalar reward r_{t+1}.

The reinforcement learning problem can be viewed as an iterative process, where the agent learns to act optimally by interacting with its environment over time. As the reward signal r effectively indicates how *good* an action is with respect to a reward function $R(\boldsymbol{x}, \boldsymbol{u})$, the agent therefore aims to adapt its behavior to increase performance with regards to R. Formally, the problem can be framed as a Markov decision process.

2.7.1 Markov decision processes

A Markov Decision Process (MDP) is a sequential, discrete time, fully observable stochastic process with a Markovian transition probability model [46]. It can be seen as an extension of the simple Markov chain in Section 2.4.1, which in addition to states and transition probabilities also contains actions and rewards. A MDP can formally be denoted by the four-tuple $\langle \mathcal{X}, \mathcal{U}, T, R \rangle$, where $\mathcal{X}$ is the state space, $\mathcal{U}$ is the action space, $T : \mathcal{X} \times \mathcal{U} \rightarrow \mathcal{X}$ is the transition function given by the transition probability $P(\boldsymbol{x}_{t+1}, r_{t+1} | \boldsymbol{x}_t, \boldsymbol{u}_t)$, and $R : \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R}$ is the reward function [19]. The transition to the next state of a MDP is given by the current state and the action performed in this state. The transition also leads to a reward for the action taken. The reward function, either dependent only on the system state or on both the state and action, returns a reward which can be processed by the RL agent and used to select the next action. A MDP can also be used to model deterministic systems with unknown model errors.

2.7.2 Bellman optimality

The goal of a decision-making agent is defined by an optimality criterion. For reinforcement learning agents, the goal is to maximize the expected future reward over time. This can be expressed through a value function v, which can be defined as the cumulative reward an agent receives by taking actions $\boldsymbol{u}_t$ over time. At a certain state $\boldsymbol{x}_0$ starting at time t_0, the value can be expressed as

$$v(\boldsymbol{x}_0) = \sum_{t=0}^{\infty} R(\boldsymbol{x}_t, \boldsymbol{u}_t). \tag{2.39}$$

This performance goal described by the value function can be linked to policy selection, by selecting the action which gives the largest reward and highest value. Given expected discounted future reward over an infinite horizon with discount factor γ, the value function v then represents how rewarding it is for an agent to be in a state [4]. Under a policy π, for which the agent starts in $\boldsymbol{x}$ and then selects actions that follow the policy, $\boldsymbol{u} = \pi(\boldsymbol{x})$, the value is

$$v^{\pi}(\boldsymbol{x}) = E_{\pi} \left\{ \sum_{t=0}^{\infty} \gamma^t R(\boldsymbol{x}_t, \boldsymbol{u}_t) \right\}. \tag{2.40}$$

It can be shown, as in Sutton *et al.* [4], that value functions satisfy a recursive relationship [4]. (2.40) can be rewritten as a sum of the immediate reward from state $\boldsymbol{x}$ and values of possible next states $\boldsymbol{x}'$, weighted by the transition probability, so that

$$v^{\pi}(\boldsymbol{x}) = \sum_{\boldsymbol{x}' \in \mathcal{X}} P(\boldsymbol{x} | \boldsymbol{x}', \pi(\boldsymbol{x}))(R(\boldsymbol{x}, \pi(\boldsymbol{x})) + \gamma^t v^{\pi}(\boldsymbol{x}')). \tag{2.41}$$

Bellman's principle of optimality states that if starting in a state and following a policy does not result in optimal values for subsequent states, then selecting other actions from an alternative policy would yield higher value [4]. This means that for an optimal value function v^* generated by selecting the actions that yield the highest value at every time step, an optimal policy can be found according to

$$\pi^*(\boldsymbol{x}) = \underset{\boldsymbol{u}\in\mathcal{U}}{\arg\max} \sum_{\boldsymbol{x}'\in\mathcal{X}} P(\boldsymbol{x}|\boldsymbol{x}', \pi(\boldsymbol{x}))(R(\boldsymbol{x}, \pi(\boldsymbol{x})) + \gamma^t v^*(\boldsymbol{x}')). \tag{2.42}$$

Exploration vs. exploitation

Selecting the best action at every single time step results in a greedy policy. Care must be taken when solving reinforcement learning problems not to exclusively exploit, as an agent could miss out on actions which may ultimately yield more reward and higher value further along in time. This is known as the *exploration vs. exploitation* trade-off in reinforcement learning. The greedy action maximizes the expected reward for one time step ahead, but by exploring other potential action selections, the agent might receive higher total reward [4].

2.7.3 Model-based reinforcement learning

The class of reinforcement learning problems where a model of the transition dynamics of the environment is known and used during learning, or can be estimated, is called model-based reinforcement learning. These types of problems can be solved with dynamical programming, which works by iteratively solving for an optimal policy, and utilizing the recursive Bellman equation in (2.41).

Generalized policy iteration is a set of model-based algorithms for finding optimal control policies for finite MDPs [4]. It is possible to obtain a monotonically improving sequence of control policies by alternating between policy evaluation, e.g. estimating the value function for current policy, and policy improvement, e.g. recalculating a locally optimal policy for a new estimate of the optimal value function [4]. If the policy is improved after this step, it means it is not stable and therefore not optimal as evaluation is being done in a greedy manner. Together, these two steps yield strict policy improvement and guarantee convergence in finite problem settings in discrete state- and action spaces. Policy iteration can be illustrated by the sequence of optimization steps

$$\pi_0(\boldsymbol{x}) \longrightarrow v^{\pi_0}(\boldsymbol{x}) \longrightarrow \pi_1(\boldsymbol{x}) \longrightarrow v^{\pi_0}(\boldsymbol{x}) \longrightarrow \dots \longrightarrow \pi_*(\boldsymbol{x}) \longrightarrow v^{\pi_*}(\boldsymbol{x}). \tag{2.43}$$

Approximate dynamic programming techniques can be used to estimate a value function for large scale problems, and bypasses the *curse of dimensionality* [52]. In problems with continuous state- and action spaces, then for a given parameterized policy, it is possible to perform approximate policy iteration and update the policy parameters by solving a nonlinear optimization problem [52]. One such class of techniques are policy gradient methods.

2.7.4 Policy gradient methods

Policy gradient methods are a class of reinforcement learning methods that aim to find an estimate of the optimal system behavior by optimizing a parameterized policy $\pi_{\boldsymbol{\theta}}$ with respect to a performance measure related to the expected return [4]. In contrast to reinforcement learning methods that learn an estimate of the values of different actions, and then select actions through a policy π based on the maximization of state values as in (2.42), policy gradient methods select actions based on the parameterized policy. The policy is in many cases considered stochastic, so that $\pi_{\boldsymbol{\theta}}(\boldsymbol{u} \mid \boldsymbol{x})$ represents a probability distribution. Even though a value function may be used to find an optimal set of policy parameters $\boldsymbol{\theta}$, it is not used directly in order to learn an optimal policy. Continuous state- and action spaces are common in many practical applications of reinforcement learning, for instance in robotics. This necessitates the need for policies and value functions to be expressed as function approximators, as the state value and policy entry for each state in the state space cannot be stored separately in a table format [53].

In order to find an optimal policy, some performance measure $J(\cdot)$ should be maximized with regards to the parameters of the policy. Similarly to the finite problem setting, a common objective function is the expected discounted cumulative reward under the stochastic policy $\boldsymbol{u} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{x})$, given by

$$J(\boldsymbol{\theta}) = E_{\pi_{\boldsymbol{\theta}}} \left\{ \sum_{t=0}^{\infty} \gamma^t R(\boldsymbol{x}_t, \boldsymbol{u}_t) \right\}. \tag{2.44}$$

Generally, the parameters of the policy can be augmented according to the gradient ascent update rule

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \eta \nabla J(\boldsymbol{\theta}). \tag{2.45}$$

where η denotes the learning rate. The gradient of the objective function can then be found either by direct calculation, finite difference methods, or sampling a batch of trajectories to approximate it [4].

Actor-critic methods

Actor-critic methods are a type of policy gradient methods where both the policy and the value function are approximated. An actor interacts with the environment, and attempts to

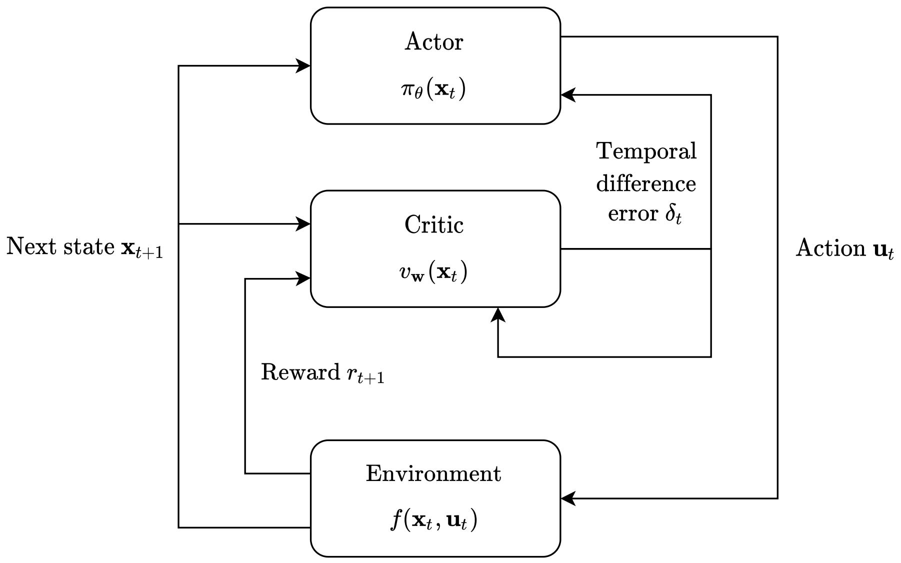

Actor
$\pi_\theta(\mathbf{x}_t)$
Temporal difference error δ_t
Critic
$v_\mathrm{w}(\mathbf{x}_t)$
Next state $\mathbf{x}_{t+1}$
Action $\mathbf{u}_t$
Reward r_{t+1}
Environment
$f(\mathbf{x}_t, \mathbf{u}_t)$

Figure 2.3: Actor-critic architecture in reinforcement learning.

learn an approximation of the optimal policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{x})$, which is often stochastic. A critic is a separate function approximator $v_{\boldsymbol{w}}(\boldsymbol{x})$ parameterized by $\boldsymbol{w}$, which learns an approximation of the value function. The critic uses the approximation of the value function to evaluate the current policy, and provides feedback to the actor through a temporal difference error which denotes the error between the true value function and the approximation [53]. The actor then improves its approximation of the optimal policy based on the feedback from the critic. The actor-critic architecture is illustrated in Figure 2.3.

The temporal difference error, which denotes the difference between the right and left side of the Bellman equation in (2.41), can be expressed as [4]

$$\delta = R(\boldsymbol{x}, \boldsymbol{u}) + \gamma v(\boldsymbol{x}') - v(\boldsymbol{x}). \tag{2.46}$$

The temporal difference error in (2.46) is then used to update the parameters of the critic. Using, for instance, a squared loss function on the form $J(\boldsymbol{w}) = \frac{1}{2}\delta^2$ and a learning rate $\eta_{\boldsymbol{w}}$, then the following standard update rule can be used [53]:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta_{\boldsymbol{w}} \delta \nabla_{\boldsymbol{w}} v_{\boldsymbol{w}}(\boldsymbol{x}). \tag{2.47}$$

Using the policy gradient theorem, derived in [4] as

$$\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\boldsymbol{x}) = \pi_{\boldsymbol{\theta}}(\boldsymbol{x}) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{x}), \tag{2.48}$$

then it can be shown using differential calculus, as in Sutton *et al.* [4], that the parameters of the actor can be updated according to the update rule

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta_{\boldsymbol{\theta}} \delta \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{x}). \tag{2.49}$$

Algorithm 1: One-step actor-critic method

```
Inputs: Randomly initialized policy $\pi_{\boldsymbol{\theta}}$,
        Randomly initialized state-value function $v_{\boldsymbol{w}}$,
        Learning rates $\eta_{\boldsymbol{\theta}}, \eta_{\boldsymbol{w}}$
for $n = 1, 2, \dots$ do
    Initialize state $\boldsymbol{x} = \boldsymbol{x}_0$
    while $\boldsymbol{x}$ is not terminal do
        $\boldsymbol{u} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{x})$
        Take action $\boldsymbol{u}$, observe next state $\boldsymbol{x}'$ and reward $r$
        $\delta \leftarrow r + \gamma v_{\boldsymbol{w}}(\boldsymbol{x}') - v_{\boldsymbol{w}}(\boldsymbol{x})$
        $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta_{\boldsymbol{w}} \delta \nabla_{\boldsymbol{w}} v_{\boldsymbol{w}}(\boldsymbol{x})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta_{\boldsymbol{\theta}} \delta \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{x})$
        $\boldsymbol{x} \leftarrow \boldsymbol{x}'$
    end
end
```

Combining the update rules in (2.47) and (2.49) for the critic approximating the value function and the actor approximating the policy, a method for finding an optimal policy using policy gradients can be found, by exploring the environment and running the optimization episodically. The resulting algorithm that describes a one-step episodic actor-critic method is shown in Algorithm 1.

2.8 Deep learning

Deep learning is a sub-field of machine learning, where models are used to find relationships and patterns by learning representations of data. The methods used in deep learning compose several layers of nonlinear functions, which transforms the data representations at each layer and enables learning at increasing levels of abstraction. In this way, deep learning models are able to learn complex function representations [54]. Neural networks are a class of deep learning models which have proven effective in many applications involving regression, classification and detection. They have been central in the development of a range of machine learning algorithms, such as deep reinforcement learning, and form the basis for many commercial and practical applications of statistical learning methods [55].

2.8.1 Feed-forward neural networks

Artificial Feed Forward Neural Networks (FFNNs) are a class of function approximators that learn to approximate some mathematical function f^*. A neural network defines a functional mapping $\boldsymbol{y} = f(\boldsymbol{x};\boldsymbol{\theta})$ between an input $\boldsymbol{x}$ and an output $\boldsymbol{y}$, parameterized by $\boldsymbol{\theta}$. The goal of training a neural network is to learn the value of the parameters $\boldsymbol{\theta}$ that correspond to the most accurate approximation of f^* based on training data $\boldsymbol{x}$.

Neural networks are represented as acyclical graphs, and can be seen as a composition of layers of nonlinear mappings, where each layer contains a linear mapping wrapped by a nonlinear activation function [55]. For example, a neural network with three layers is composed as

$$f(\boldsymbol{x}) = f^3(f^2(f^1(\boldsymbol{x}))), \tag{2.50}$$

where f^3 is the output layer, f^2 a hidden layer and f^1 the input layer. Figure 2.4 shows a graphical representation of a feed forward neural network with three layers and n nodes in each layer. The output of a each layer is given by a weight matrix $\boldsymbol{W}$, bias vector $\boldsymbol{b}$ and an activation function f. Given an input vector $\boldsymbol{x}$ to the layer, then

$$\boldsymbol{y} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \tag{2.51}$$

describes the output $\boldsymbol{y}$ of the layer. Typical activation functions are Sigmoid functions, Rectified Linear Units (ReLUs) or Tanh functions [55]. The activation of the output layer is often chosen to be a simple linear function, equivalent to an identity mapping, in order to let the output of the network take on any value. All weights and biases of a network can be collected in the parameter vector $\boldsymbol{\theta}$.

The output of a neural network can also be expressed on stochastic form. In this case, the output layer is made up of two separate nodes representing the mean and variance of a probability distribution, for instance a Gaussian distribution. The mean and variance which is returned from the network each forward pass is then used as the parameters for an instance of a probability distribution, which can then be sampled in order to produce stochastic outputs from the neural network. This method can for example be used to train stochastic control policies used in reinforcement learning. In order to find the set of parameters in a neural network which best fits f to the true target function f^*, the parameter vector must be adjusted. Commonly, this is done by minimizing a cost function $J(\boldsymbol{\theta})$ with respect to the parameters. The cost function captures an optimality criterion, and can be described generally as the cross-entropy between training data and the probability distribution of the true model [55]. As such, neural networks can be trained using the principle of maximum likelihood.

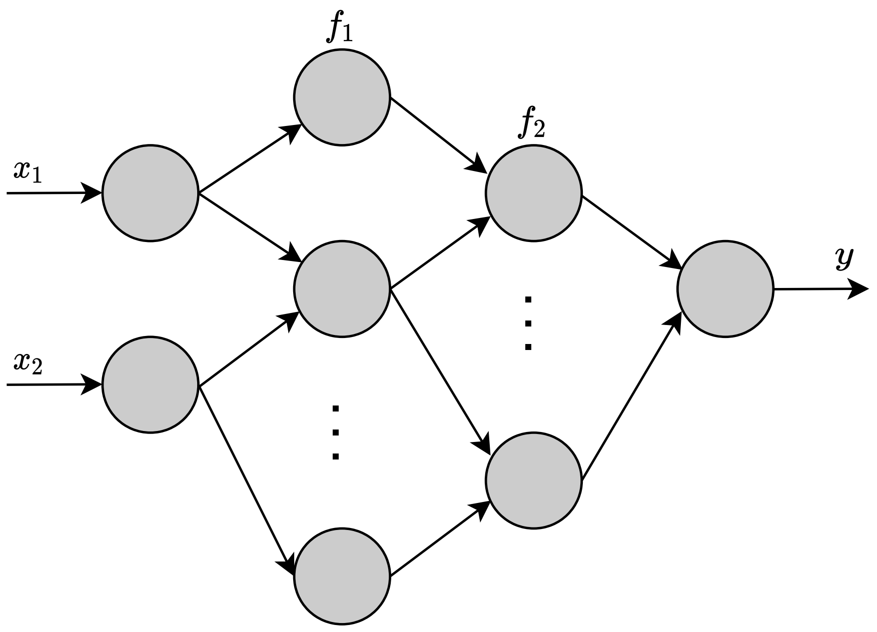


Figure 2.4: Network layer structure of feed forward neural network.

Gradient descent

The gradient descent algorithm can be used to train a neural network by minimizing the cost function through parameter optimization. Analogous to the gradient ascent update rule used for deep reinforcement learning, but with an opposite sign, the parameters are augmented in a direction opposite of the gradient $J(\boldsymbol{\theta})$ of the cost function. This update rule can be written as

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla J(\boldsymbol{\theta}) \tag{2.52}$$

where η denotes the learning rate. A version of gradient descent called Stochastic Gradient Descent (SGD) randomly samples one data points from the training data to perform the optimization each iteration. Essentially, only one data point is used to calculate an estimate of the gradient of J, so the variance in resulting update steps can become higher than when using all available training data to estimate the gradient.

Adaptive moment estimation, Adam, is a method for reducing the variance of the gradient calculations and thereby speed up training and improve the convergence rate [55]. The Adam method uses a moving average of the first- and second-order moments of the gradient of the objective function in order to update the weights of the neural network in the direction of the moving averages.

3 | Safe learning with control barrier functions

In this chapter, the main results of the thesis are presented. First, probabilistic safety constraints based on control barrier functions are expressed on a form which lets them be used for policy optimization and safety verification. Next, a method for performing safety-constrained policy optimization by employing probabilistic control barrier conditions is presented. Then, a method for safe exploration in a way which guarantees safety with high probability during data collection is developed. Finally, a theoretical framework combining safety-constrained policy optimization and safe exploration is presented, and two different versions of a `SafeLearningCBF` algorithm are shown.

3.1 Background and assumptions

The aim of the safe reinforcement learning framework can be divided into two parts. One part is optimizing a parameterized policy subject to safety constraints, in order to encourage safe closed-loop system behavior in addition to the optimality criteria typically used to measure performance in reinforcement learning. The second part is safe exploration, in order to gather data for learning about the unknown system dynamics without encountering unsafe regions of the state space.

When synthesizing a policy using a model-based reinforcement learning algorithm under the assumption of *a priori* unknown system dynamics, it is essential to actively gather information about the environment to improve the model used for policy optimization. During online learning of the unknown dynamics, meaning data is sampled online to update the model, only actions that belong to state-action pairs that are verified as safe and which do not violate the safety constraints should be applied to the real system. To this end, the most central part of a safe exploration scheme for data collection is attempting to ensure that the policy used during exploration does not drive the closed-loop system to any unsafe states.

The safety constraints in Section 2.3 defined by control barrier functions are dependent on a statistical model in the cases where the true dynamics are *a priori* unknown. This model can be formulated as a Gaussian process, which produces uncertain predictions of the dynamics. The expression for a control barrier condition, as defined in (2.18), will then contain the Lie derivatives of a control barrier function along components of the Gaussian process. The expression for the safety constraints therefore contains a model that produces uncertain predictions of the true dynamics. Thus, the control barrier conditions will produce posterior estimates for the safety constraints that are also uncertain. Probabilistic safety constraints must therefore be used to account for this uncertainty and, consequently, safe learning can be achieved in a way such that safety generally can be guaranteed *with high probability.*

For clarity, the control-affine model of the system dynamics is restated as

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u} = [f(\boldsymbol{x}) \;\; g(\boldsymbol{x})] \begin{bmatrix} 1 \\ \boldsymbol{u} \end{bmatrix} := F(\boldsymbol{x})\underline{\boldsymbol{u}}, \tag{3.1}$$

and the matrix variate Gaussian process posterior distribution over (3.1) conditioned on k samples is restated as

$$F_k(\boldsymbol{x})\underline{\boldsymbol{u}} \sim \mathcal{GP}(\boldsymbol{M}_k(\boldsymbol{x})\underline{\boldsymbol{u}}, \underline{\boldsymbol{u}}^T\boldsymbol{B}_k(\boldsymbol{x},\boldsymbol{x}')\underline{\boldsymbol{u}} \otimes \boldsymbol{A}). \tag{3.2}$$

Throughout the development of the safe learning framework, some typical assumptions on the continuity of the dynamics model and control policies are made.

Assumption 1 (Continuity of dynamics and policy)**.**
The drift- and gain terms of the dynamics in (3.1)*, f and g, are L_f- and L_g locally Lipschitz with respect to the 1-norm. The control policies being used are restricted to a set Π_L of functions that are L_π Lipschitz with respect to the 1-norm.*

The assumptions of locally Lipschitz dynamics and policies being made in Assumption 1 are considered standard in many control theory applications [34]. Furthermore, the functions defined by the set Π_L of potential control policies include a class of Lipschitz continuous neural networks, which enables the use of approximate dynamic programming techniques for policy optimization.

In addition to assumptions on the continuity of the system, the local Lipschitz continuity of the posterior distribution of the unknown system dynamics is assumed *with high probability*, in order to enable the construction of probabilistic safety constraints.

Assumption 2 (Assumption 1 from Dhiman *et al.* [1]).
Let P_k be the probability measure induced by the distribution of $F(\boldsymbol{x})\underline{\boldsymbol{u}}$ at time t_k. Assume that for any $L_k > 0$, $\boldsymbol{u}_{t_k}$, and τ_k, there exists a constant $b_k > 0$ such that:

$$P_k(\sup_{s\in[0,\tau_k)} \|F(\boldsymbol{x}(t_k+s))\underline{\boldsymbol{u}_{t_k}} - F(\boldsymbol{x}(t_k))\underline{\boldsymbol{u}_{t_k}}\| \leq \|\boldsymbol{x}(t_k+s) - \boldsymbol{x}(t_k)\|) \geq q_k := 1 - e^{-b_k L_k}.$$

Assumption 2 holds for many classes of Gaussian processes, including those with stationary kernels that are four times differentiable, such as squared exponential kernels and a class of some *Matérn* kernels [38, 56]. Essentially, the assumption contributes to ensuring sufficient conditions of smoothness on the statistical model of the unknown dynamics.

3.2 Probabilistic safety constraints

The safety conditions outlined in Theorem 3 effectively define constraints for which control actions generated from a policy, $\boldsymbol{u} = \pi(\boldsymbol{x})$, can be applied to the real system dynamics. Only those actions $\boldsymbol{u}$ that render the system safe with respect to the safe set C in (2.16), which implies that the conditions on the control barrier condition $CBC(\cdot)$ hold, are admissible. When the unknown dynamics are modeled using a statistical model however, the safety constraints given by the sufficient conditions for safety, $CBC(\boldsymbol{x}, \boldsymbol{u}) \geq 0$, are not necessarily satisfied. The posterior distribution of $F(\boldsymbol{x})\underline{\boldsymbol{u}}$, which is given on matrix variate form in (3.2), is a probabilistic estimate of the dynamics and not deterministic.

Dhiman *et al.* [1] define a probabilistic version of the safety constraints in Definition 3, induced by the posterior distribution of $F(\boldsymbol{x})\underline{\boldsymbol{u}}$. For a given state-action pair, $(\boldsymbol{x}, \boldsymbol{u})$, and given a probability threshold p, then a probabilistic control barrier condition can be expressed as

$$P(CBC(\boldsymbol{x}, \boldsymbol{u}) \geq 0) \geq p. \tag{3.3}$$

The probabilistic formulation in (3.3) can be extended to systems of higher-order relative degrees. It can be further specified by enforcing a tighter, positive constraint on $CBC^{(r)}(\cdot)$. When the probability measure P is induced at a sampling time t_k, a tighter bound can be imposed on the control barrier condition so that is holds between the sampling times as well, during inter-triggering times, through the use of a variable ζ [1, 38]. For a given state-action pair, $(\boldsymbol{x}, \boldsymbol{u})$, and given the threshold p, probabilistic safety constraints induced by the matrix variate posterior distribution of $F(\boldsymbol{x})\underline{\boldsymbol{u}}$ can then be expressed for r-th order relative degree systems as

$$P(CBC^{(r)}(\boldsymbol{x}, \boldsymbol{u}) > \zeta \mid \boldsymbol{x}, \boldsymbol{u}) \geq p. \tag{3.4}$$

Assuming that the true dynamics in (3.1) are Lipschitz continuous with high probability, as in Assumption 2, and that the Gaussian process of the dynamics has a Lipschitz continuous kernel, then by selecting a small enough sampling time, the safety constraints defined by (3.4) can be considered also at inter-sampling times [1].

Based on the sufficient conditions for safety outlined in Theorem 3, and following the assumptions on the matrix variate Gaussian process model, a probabilistic condition for safety can then be formulated.

Lemma 1 (Probabilistic condition for safety).
Consider h to be an r-th order exponential control barrier function for the control-affine system in (3.1), *and $\nabla h(\boldsymbol{x}) \neq 0 \ \forall \ \boldsymbol{x}$ when $h(\boldsymbol{x}) = 0$. Furthermore, let P be a probability measure induced by the posterior distribution of $F(\boldsymbol{x})\underline{\boldsymbol{u}}$ and let the posterior distribution of $F(\boldsymbol{x})\underline{\boldsymbol{u}}$ be modeled by the matrix variate Gaussian process in* (3.2). *Then, for $\zeta > 0$, any Lipschitz continuous control policy*

$$\pi(\boldsymbol{x}) = \{\boldsymbol{u} \in \mathcal{U} \mid P(CBC^{(r)}(\boldsymbol{x}, \boldsymbol{u}) > \zeta \mid \boldsymbol{x}, \boldsymbol{u}) \geq p\} \tag{3.5}$$

will render the system in (3.1) *safe with probability p with respect to the safe set*

$$\{\boldsymbol{x} \in \mathcal{X} \mid h(\boldsymbol{x}) \geq 0\}.$$

Proof. Using Theorem 3, and following Assumption 2 and the definition of probabilistic safety constraints in (3.4), then from Propostition 6 in [1], by choosing a small enough trigger interval τ_k, it holds that (3.5) defines a safety constraint with probability p. □

The probability distribution for an r-th order probabilistic exponential control barrier condition cannot necessarily be explicitly determined [1, 34]. While this means that it can be difficult to utilize the probabilistic safety constraints for verification of safe state-action pairs, as the posterior distribution in (3.4) will most likely have to be estimated through for instance Monte Carlo sampling [38], Dhiman *et al.* [1] utilizes Cantelli's inequality to produce a bound of the concentration of measure on the probabilistic safety constraint in (3.4). Effectively, the constraint is rewritten using its first- and second order moments. Following the proof of Proposition 8 in [1], the implication

$$\begin{gathered} E\{CBC^{(r)}(\boldsymbol{x}, \boldsymbol{u})\} - \zeta > \sqrt{\frac{p}{1-p}} Var^{\frac{1}{2}}\{CBC^{(r)}(\boldsymbol{x}, \boldsymbol{u})\} \\ \Longrightarrow \\ P(CBC^{(r)}(\boldsymbol{x}, \boldsymbol{u}) > \zeta \mid \boldsymbol{x}, \boldsymbol{u}) \geq p \end{gathered} \tag{3.6}$$

enables the safety constraints for systems of higher-order relative degrees to be calculated using the sample mean and variance functions of $CBC^{(r)}(\cdot)$. It can be shown, as in Dhiman *et al.* [1], that the sample mean and variance of $CBC^{(2)}(\cdot)$ for a system with relative degree $r = 2$ can be calculated analytically. The bound imposed on the safety constraint in (3.6) can be rewritten as a safety condition on the form of a zero-inequality, so that (3.4) effectively is expressed by the mean and variance of the posterior of $CBC^{(r)}(\cdot)$ on standard form. Hence, for a state-action pair, $(\boldsymbol{x}, \boldsymbol{u})$, the following condition must be satisfied in order for an action generated by a policy, $\boldsymbol{u} = \pi(\boldsymbol{x})$, to be safe with probability p:

$$c^{(r)}(\boldsymbol{x}, \boldsymbol{u}) := \sqrt{\frac{p}{1-p}} Var^{\frac{1}{2}}\{CBC^{(r)}(\boldsymbol{x}, \boldsymbol{u})\} + \zeta - E\{CBC^{(r)}(\boldsymbol{x}, \boldsymbol{u})\} < 0. \tag{3.7}$$

The probabilistic condition for safety, $c(\boldsymbol{x}, \boldsymbol{u}) < 0$, in (3.7) is expressed on standard form. Thereby, it can be used directly as a safety constraint in optimization problems. Whereas a regular optimization problem is expressed on Lagrangian form with state- or input inequality constraints, using (3.7) to constrain an optimization yields a *chance-constrained* optimization problem as $CBC^{(r)}(\cdot)$ is a probabilistic estimate.

The condition in (3.7) is utilized both in the policy optimization stage as well as for safety verification when safely exploring the true system to collect data for Gaussian process regression.

Safety constraints vs. stability constraints

The constraints enforced by control barrier functions and the conditions in (2.18) encode a more permissive notion of safety than what is defined by the asymptotic stability properties enforced by control Lyapunov functions and the conditions in (2.14). This is because the conditions must hold only on the boundary of the safe set C, and invariance is not required over an entire level set [34]. A policy constrained by Lyapunov methods based on CLFs would require all subsets of C to be invariant to guarantee asymptotic stability. Nevertheless, as observed in [34], the property of C under the sufficient conditions of safety yields an asymptotic effect, and drives the system not to the origin or another asymptotically stable point, but rather keeps it inside the safe set.

3.3 Safe policy optimization

A learned policy in a safety-critical setting should satisfy the conditions for safety in Lemma 1, and the policy should be optimized subject to the safety constraints. These conditions are encoded in the safety requirements defined by the control barrier function, $h(\boldsymbol{x})$, related to the control task at hand. The safety constraints in (3.4) for a certain task are defined based on the control barrier function, and evaluated probabilistically based on an instance of the control barrier condition, $CBC^{(r)}(\boldsymbol{x}, \boldsymbol{u})$. As observed from the sufficient conditions for safety in Theorem 3 and Lemma 1, the control barrier condition is evaluated for state-action pairs generated by a closed-loop system. Naturally, it follows that the policy that is used to produce the actions affects whether the state-action pairs are evaluated as safe or not. This means that for a system to be rendered safe with high probability, it should be driven by a control policy for which the conditions in (3.5) hold.

Following the conditions in (3.5) of Lemma 1, the set of all state-action pairs that fulfill the probabilistic condition for safety can be defined as

$$\mathcal{S} = \{(\boldsymbol{x}, \boldsymbol{u}) \in \mathcal{X} \times \mathcal{U} \mid P(CBC^{(r)}(\boldsymbol{x}, \boldsymbol{u}) > \zeta \mid \boldsymbol{x}, \boldsymbol{u}) \geq p\}. \tag{3.8}$$

As can be observed from (3.8), the set $\mathcal{S}$ is restricted to state-action pairs that uphold the probabilistic safety constraints in (3.4).

A requirement posed on a safe policy that follows from the definition of the safe state-action set $\mathcal{S}$, is that each action returned by the policy must result in $(\boldsymbol{x}, \pi(\boldsymbol{x}))$ being contained in $\mathcal{S}$. Depending on the set of potential policies in Π_L and the nature of the control task at hand, this requirement may hold for several policies that each yield a different type of safe closed-loop behavior. These policies, which are characterized as safe and which can be used to apply actions to a real safety-critical system, can be defined by a subset of *allowable* policies for which the safety constraints are satisfied, $\Gamma \subset \Pi_L$. The set Γ is illustrated in Figure 3.1, and the allowable policies are constrained by the safe state-action set

$$\pi \in \Pi_L, \quad \text{s.t.} \; \forall \, \boldsymbol{x} \in \mathcal{X} : (\boldsymbol{x}, \pi(\boldsymbol{x})) \in \mathcal{S}. \tag{3.9}$$

The policies contained in the allowable set of policies Γ only consider safety. In most learning-based control methods however, there is an inherent aim to solve the control task in the most optimal way if possible. Therefore, as a way to compare several safe, allowable policies, it is assumed that the policy is optimized according to some performance measure $J(\cdot)$, such as the standard measure of maximum cumulative discounted reward in reinforcement learning from Section 2.7. An optimal safe policy can then be found as the result of the constrained optimization problem

$$\pi^* = \underset{\pi \in \Pi_L}{\arg\max}\, J(\cdot), \quad \text{s.t.} \; \forall \, \boldsymbol{x} \in \mathcal{X} : (\boldsymbol{x}, \pi(\boldsymbol{x})) \in \mathcal{S}. \tag{3.10}$$

Evidently, the optimization problem in (3.10) will be intractable in practice. The set $\mathcal{S}$ cannot be evaluated for every state-action pair as long as the states and actions are defined based on the continuous spaces $\mathcal{X}$ and $\mathcal{U}$. A grid-based discretization of the safe set might

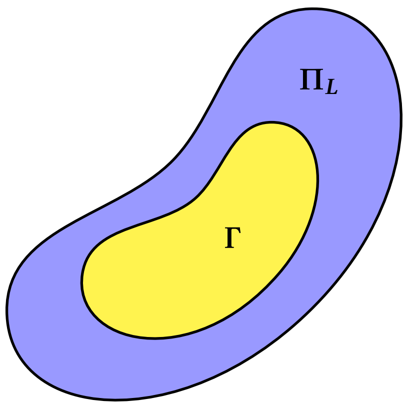


Figure 3.1: Safe set of allowable policies.

be possible by defining a discrete version of the state space $\mathcal{X}_\tau \subset \mathcal{X}$, for instance using some adaptive discretization scheme [57]. However, the optimization would still be rendered intractable as the policy function is still optimized over continuous values.

The optimization in (3.10) can be expressed slightly differently, in order to enable tractable policy updates that consider both performance measures and provides probabilistic guarantees for safety. Given a policy $\pi_{\boldsymbol{\theta}}$ parameterized by $\boldsymbol{\theta}$, then (3.10) can be viewed from a Lagrangian perspective. The problem can be posed as a safety-constrained Lagrangian optimization problem, where the Lagrangian penalty term is defined by safety constraints rather than terms based on specific performance requirements, regularizers that serve to augment the objective or typical maximum bounds on the states and inputs. Utilizing the probabilistic safety constraint on standard form in (3.7), then

$$\pi^*_{\boldsymbol{\theta}} = \underset{\pi \in \Pi_L}{\arg\max} \int_{\boldsymbol{x} \in \mathcal{X}} J_{\boldsymbol{\theta}}(\boldsymbol{x}) - \lambda c^{(r)}(\boldsymbol{x}, \boldsymbol{u}), \tag{3.11}$$

is an equation for policy updates formulated as a safety-constrained optimization problem. Since the constraints are probabilistic, the optimization problem is termed *chance-constrained.* In (3.11), $\lambda \geq 0$ is the Lagrange multiplier for the safety constraint.

In a reinforcement learning setting, the aim is to maximize expected cumulative reward, or equivalently, minimize the negative of the return. By using reward as a performance measure and specifying the objective in (3.11) using the maximized return from (2.42), then inserting the definition of $c^{(r)}(\cdot)$ yields

$$\begin{aligned} \pi^*_{\boldsymbol{\theta}} = \underset{\pi \in \Pi_L}{\arg\max} \int_{\boldsymbol{x} \in \mathcal{X}} & (R(\boldsymbol{x}, \pi(\boldsymbol{x})) + \gamma v(\boldsymbol{x})) \\ & - \lambda \left(\sqrt{\frac{p}{1-p}} Var^{\frac{1}{2}}\{CBC^{(r)}(\boldsymbol{x}, \boldsymbol{u})\} + \zeta - E\{CBC^{(r)}(\boldsymbol{x}, \boldsymbol{u})\} \right). \end{aligned} \tag{3.12}$$

The objective in (3.12) can be viewed as a surrogate objective function. This surrogate objective can then be used in a reinforcement learning algorithm in order to perform policy updates to learn a policy that is both optimal and safe with high probability. Such surrogates have been utilized in [27], in order to encode risk-based safety requirements in addition to performance requirements during policy evaluation and optimization, as well as during safety verification in exploration on the real system dynamics. By replacing the integral in (3.11) with a finite sum, approximate dynamic programming can be used to perform approximate policy updates by

$$\begin{aligned}\pi_{\boldsymbol{\theta}}^* = \arg\max_{\pi \in \Pi_L} \sum_{t=0}^{T} &\left(R(\boldsymbol{x}_t, \pi(\boldsymbol{x}_t)) + \gamma v(\boldsymbol{x}_t)\right) \\ &- \lambda \left(\sqrt{\frac{p}{1-p}} Var^{\frac{1}{2}}\{CBC^{(r)}(\boldsymbol{x}_t, \boldsymbol{u}_t)\} + \zeta - E\{CBC^{(r)}(\boldsymbol{x}_t, \boldsymbol{u}_t)\} \right).\end{aligned} \tag{3.13}$$

A theoretical safety-constrained policy update scheme for a parameterized policy is summarized in Algorithm 2. In practice, stochastic gradient descent can be used to episodically refine the estimate of the optimal policy in (3.12).

Algorithm 2: Safety-constrained policy update

```
  Inputs: Control-affine dynamics model terms $f$ and $g$,
          control barrier function $h$,
          probability threshold $p$,
          horizon $T$,
          randomly initialized policy $\pi_{\boldsymbol{\theta},0}$
          initial state $\boldsymbol{x}_0$
1 for $n = 1, 2, \dots$ do
2     for $t = 1$ to $T$ do
3         $\boldsymbol{u}_t \sim \pi_{\boldsymbol{\theta},n-1}(\boldsymbol{x}_{t-1})$
4         $\boldsymbol{x}_t \sim f(\boldsymbol{x}_{t-1}) + g(\boldsymbol{x}_{t-1})\boldsymbol{u}_t$
5         $CBC^{(r)}(\boldsymbol{x}_t, \boldsymbol{u}_t) \leftarrow \mathcal{L}_f^{(r)} h(\boldsymbol{x}_t) + \mathcal{L}_g \mathcal{L}_f^{(r-1)} h(\boldsymbol{x}_t)\boldsymbol{u}_t + \boldsymbol{k}_\alpha \eta(\boldsymbol{x}_t)$
6         $c^{(r)}(\boldsymbol{x}_t, \boldsymbol{u}_t) \leftarrow \sqrt{\frac{p}{1-p}} Var^{\frac{1}{2}}\{CBC^{(r)}(\boldsymbol{x}_t, \boldsymbol{u}_t)\} + \zeta - E\{CBC^{(r)}(\boldsymbol{x}_t, \boldsymbol{u}_t)\}$
7     end
8     $\pi_{\boldsymbol{\theta},n} \leftarrow \arg\max_{\pi \in \Pi_L} \sum_{t=0}^{T} R(\boldsymbol{x}_t, \pi(\boldsymbol{x}_t)) + \gamma v(\boldsymbol{x}_t) - \lambda c^{(r)}(\boldsymbol{x}_t, \boldsymbol{u}_t)$
9 end
```

When optimizing a policy using a reinforcement learning algorithm, in order not to drive the system to possibly unsafe states outside the safe set, actions should only be applied to the true system dynamics if they are verified as safe. In model-based reinforcement learning, actions are applied to a model estimate defined for instance by a Gaussian process model, so in this case no guarantees of safety are needed during policy optimization. While model-based learning algorithms do not need safety checks for every sampling step taken on the model, every state-action pair to be evaluated on the real system dynamics should still be verified as safe. In order to maintain the guarantees of safety of the learning framework however, exploration of the environment, which implies applying actions to the true system dynamics, during data collection for model improvement must then be safety-constrained.

3.4 Safe exploration

A safety-constrained policy optimization scheme is aimed at learning a policy that satisfies both performance requirements and safety constraints. Similarly, a safe exploration scheme should be aimed at safely learning about the unknown system dynamics in order to improve the policy by only sampling regions of the state space that are guaranteed to be safe with high probability. In particular, any exploration scheme that samples data from the real system to inform a stochastic process posterior estimate should verify the safety of a state-action pair $(\boldsymbol{x}, \boldsymbol{u})$ prior to applying an action to the real system and observing the subsequent state. As established in Section 3.2, the probabilistic safety constraint in (3.5) serves as an effective condition for this purpose.

One alternative for safe exploration is to explicitly verify the safety of every potential state-action pair generated by a particular allowable policy. However, this essentially amounts to evaluating the safe state-action set $\mathcal{S}$ at every iteration, which is intractable in practice as the set is continuous. As a consequence, $\mathcal{S}$ cannot be practically utilized during exploration.

Alternatively, the concentration bound formulation on the probabilistic safety constraint in (3.7), expressed as a function of the posterior mean and variance of the CBC, can be utilized in order to verify the safety of each candidate state-action pair selected to be sampled on the real system. This means that a safe exploration scheme can be developed, by choosing a candidate state to be sampled on the real system, selecting a candidate action from some exploration policy, and then verifying the safety of the state-action pair. If the safety constraint in (3.7) is less than zero, the pair is deemed safe with high probability, and thus the action can safely be applied to the real system in order to collect data for regression.

The posterior of the model of the system dynamics in (2.32) is uncertain. As established above, and which can also be observed in Definition 3, this means that predictions of the safety constraint value $CBC(\cdot)$ for a given state-action pair will also be uncertain. Because of this, it is necessary to investigate the bounds on this uncertainty, which can be accomplished by constructing high-probability confidence intervals on $CBC(\cdot)$.

In order to construct confidence intervals based on the first- and second order moments of a statistical model, some additional assumptions are needed. From [58], a *well-callibrated* statistical model is defined as reliable through an inequality using the mean and covariance functions of the posterior of the model. The matrix variate Gaussian process posterior distribution in (3.2) is control-affine, and thereby a specification of a general nonlinear model, so a well-callibrated model is assumed as follows:

Assumption 3 (Well-callibrated control-affine Gaussian process model).
Let $\mu_k(\boldsymbol{x}, \boldsymbol{u}) := \boldsymbol{M}_k(\boldsymbol{x})\underline{\boldsymbol{u}}$ *and* $\Sigma(\boldsymbol{x}, \boldsymbol{u}) := \underline{\boldsymbol{u}}^T \boldsymbol{B}_k(\boldsymbol{x}, \boldsymbol{x}')\underline{\boldsymbol{u}} \otimes \boldsymbol{A}$ *be the posterior mean and covariance functions of the statistical model of the dynamics in* (3.1)*, conditioned on k noisy measurements.*

With $\sigma_k(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{tr}(\Sigma_k^{\frac{1}{2}}(\boldsymbol{x}, \boldsymbol{u}))$*, there exists a* $\beta_k > 0$ *such that with probability at least* $(1-\delta)$ *for some* $\delta > 0$ *it holds for all* $k \geq 0$, $\boldsymbol{x} \in \mathcal{X}$ *and* $\boldsymbol{u} \in \mathcal{U}$ *that*

$$\|F(\boldsymbol{x})\underline{\boldsymbol{u}} - \mu_k(\boldsymbol{x}, \boldsymbol{u})\|_1 \leq \beta_k \sigma(\boldsymbol{x}, \boldsymbol{u}).$$

Safety must be ensured during the exploration step. By utilizing Assumption 3 for the mean μ and variance σ of the GP, as well as selecting a coefficient β, then confidence intervals on $CBC(\cdot)$ can be constructed as follows:

$$Q_k(\boldsymbol{x}, \boldsymbol{u}) = [CBC(\boldsymbol{x}, \boldsymbol{u}) \pm L\beta_k \|\sigma_k(\mathbf{x}, \mathbf{u})\|_2],$$

where L is a Lipschitz constant of the CBC and $\|\cdot\|_2$ denotes the two-norm.

A safe exploration state-action set $\mathcal{E} = \mathcal{S} \cap maxQ_k(\boldsymbol{x}, \boldsymbol{u})$, augmented from the safety constraints and the upper confidence interval on the control barrier condition, can then be constructed as the intersection-set

$$\mathcal{E} = \{(\boldsymbol{x}, \boldsymbol{u}) \mid P(CBC^{(r)}(\boldsymbol{x}, \boldsymbol{u}) > \zeta \mid \boldsymbol{x}, \boldsymbol{u}) \geq p \ \cap \ CBC(\boldsymbol{x}, \boldsymbol{u}) \leq maxQ_k(\boldsymbol{x}, \boldsymbol{u})\}, \quad (3.14)$$

where $maxQ_k(\mathbf{x}, \mathbf{u})$ is the upper confidence interval of Q_k.

From Assumption 2, it follows that the control barrier condition is Lipschitz continuous. Together with Assumption 3, and choosing $\delta = (1-p)$, these two assumptions then ensure that $CBC(\boldsymbol{x}, \boldsymbol{u})$ is contained in the confidence interval Q_k with probability p. This means that state-action pairs that are in $\mathcal{E}$ are also in $\mathcal{S}$. Hence, in practice, each state-action pairs only needs to satisfy the conditions in (3.4) used to define the constrained safe set $\mathcal{S}$.

3.4.1 Action selection strategies

The control actions that are going to be applied to the true system dynamics, if they are verified as safe, have to be produced by an exploration policy. These actions are used for sampling data in order to learn the unknown dynamics. Depending on what policy, or other potential means of selecting candidate actions, is used during the safe exploration scheme, different action selection strategies can be used. Which actions are selected for exploration affects what data is sampled from the true system, which in turn can affects the Gaussian process regression. There are different ways to select actions during exploration to learn unknown dynamics.

Random selection
Random action selection is a relatively simple action selection strategy, where actions are sampled from a random policy, followed by the verification of the candidate state-action pairs. When a control barrier function is predefined, the condition in (3.7) can be checked for each pair in such a way that so a random sampling scheme can be implemented to search for states that yield safe control actions according to the probabilistic safety constraints.

Random action selection for exploration may result in a less efficient exploration scheme, as unsafe actions may be encountered often. However, it may be easier to implement in practice than other exploration policies or action selection strategies. In addition, the selection process itself is very efficient since a policy is sampled directly and no optimization is required.

Bayesian optimization
Alternatively, an optimization problem can be formulated in order to find the best candidate action according to some measure, which can then be verified as safe. Note that this is not an optimization loop to find an optimal control policy, but a separate optimization for the safe exploration scheme used to apply actions on the true dynamics in order to gather data for regression.

The candidate actions can be selected according to the solution of a Bayesian optimization problem. More specifically, acquisition functions can be utilized in Bayesian optimization to select an optimal sample observation in order to learn about unknown dynamics and update the stochastic model function prior in an effective manner. Some options for various acquisition functions are entropy search or upper confidence bound maximization [59]. By maximizing the acquisition function, they can be used in relation with the safe exploration scheme to find a suitable state-action pair for exploring the environment and learning more about the model dynamics.

Recently, Upper Confidence Bound (UCB) has been proposed as an acquisition function with provable regret bounds [59]. The UCB is a function that is optimistic in the face of uncertainty, and is often used as a strategy to trade off exploration with exploitation [60]. Additionally, by optimizing the UCB, the candidate action returned will maximize the information gain for the Gaussian process model, and a state will be sampled, if the actions are deemed safe, where the model is the most uncertain. Letting μ and σ be the mean and variance of the GP model, then for a $\beta > 0$, the upper confidence bound acquisition function can be expressed as [59]

$$A(\boldsymbol{x}) = \mu(\boldsymbol{x}) + \beta\sigma(\boldsymbol{x}), \tag{3.15}$$

and an action is then selected to verify as safe by the action selection rule

$$\boldsymbol{u}(\boldsymbol{x}) = \arg\max_{\boldsymbol{u}\in\mathcal{U}} A(\boldsymbol{x}). \tag{3.16}$$

If the action selected by (3.16) yields a safe state-action pair, then it can be applied to the true dynamics.

3.4.2 Global optimization

An alternative way to ensure safety during exploration is to formulate a global optimization problem that optimizes the exploration policy. The policy π_{exp}, used to generate candidate actions for which to apply to the true dynamics for data collection, may generate unsafe actions. Through a minimally invasive optimization step, the exploration policy can be modified to ensure safe action selection during exploration. A controller can be synthesized by solving a safety-constrained Quadratic Program (QP), in which the safety constraints are defined by (3.7) such that the optimization can be constrained in a similar fashion as in Section 3.3. This safety-constrained quadratic program is termed CBF-QP, and can be expressed as

$$\boldsymbol{u}(\boldsymbol{x}) = \arg\min_{\boldsymbol{u}\in\mathcal{U}} \frac{1}{2}\|\boldsymbol{u} - \pi_{\text{exp}}(\boldsymbol{x})\|^2 \quad , \qquad \text{s.t. } c^{(r)}(\boldsymbol{x}, \boldsymbol{u}) < 0. \tag{3.17}$$

The idea behind utilizing a probabilistic CBF-QP is to apply the optimization in (3.17) in cases where a state-action pair is rejected by the safety verification procedure. Independent of the type of exploration policy or other action selection strategy chosen, the QP will act as a safety filter during exploration. The benefit of a safety filter is that it can provide actions that are known to be safe with high probability for the data collection step. However, a global optimization problem can also be computationally inefficient.

3.4.3 Backup policies

A backup policy can be used if the safety verification process during exploration rejects too many state-action pairs as unsafe. In this case, a different control policy can be used in order to perform off-policy action selection to sample data safely for Gaussian process regression. This leads to sacrifices in performance during the exploration phase, but at the same time can lead to a more straight-forward way of sampling safely and providing enough variety in the data to condition the posterior model estimate.

However, using a backup policy can be quite restrictive, as finding a safe backup policy is not easy in practice for every type of environment with unknown dynamics. When utilizing a notion of safety defined with regards to stability, a traditional locally stabilizable policy can easily be used. In the case of safety defined with regards to control barrier functions, on the other hand, a locally safe policy might not easily be synthesized or calculated. Using an action selection strategy directly may therefore is acceptable, since the overarching goal is safety during exploration, where unsafe state-action pairs should be rejected. Alternatively, a safety filter can be utilized.

3.5 Safe learning with CBFs

The methods for safe policy optimization and safe exploration can be combined to develop a theoretical framework for safe reinforcement learning. As a starting point for safe learning, a prior control-affine Gaussian process model is provided to the algorithm, as well as a user-specified probability threshold p. The inter-triggering threshold variable ζ must also be fixed, and randomly initialized parameterized policies must be provided.

The `SafeLearningCBF` algorithm includes two steps: policy optimization and Gaussian process regression. Each episode, the current optimal policy estimate is used to generate actions to sample the Gaussian process model. The resulting state-action pairs are then used to calculate values for probabilistic safety constraints, and the safety-constrained policy optimization problem from (3.11) is solved. Next, during the safe exploration stage, actions are generated by an exploration policy and the resulting candidate state-action pairs are verified as safe, before they are applied to the true system dynamics. Finally, the posterior estimate of the Gaussian process model is updated with the safe state-action pairs.

A version of the safe learning algorithm using a random controller in order to suggest candidate actions for the safe exploration scheme, and without a global safety filter, is shown in Algorithm 3. Another version of the algorithm using a exploration policy as well as a global safety filter for safe candidate action selection is shown in Algorithm 4.

Practical considerations

Notice that in both Algorithm 3 and Algorithm 4, the time variable t indicates a specific time step, as the algorithm is run iteratively with one-step predictions of the model and one-step simulations of true dynamics. As such, the notation $\boldsymbol{x}_t$ indicates the state at time t, while $\boldsymbol{x}_{t+1}$ indicates the next state at time $t+1$. This means the term $\boldsymbol{x}_{t+1} = F(\boldsymbol{x}_t)\underline{\boldsymbol{u}_t}$ indicates that the dynamics F are sampled and the subsequent state is returned.

For matrix variate Gaussian process regression, a *state-control* dataset can be constructed from the collection of safe observations, where the derivative matrix $\dot{\boldsymbol{X}}_{1:k}$ can be found by finite difference methods. In a practical implementation of the safe learning framework, simple $\mathcal{K}_\infty$ functions are defined in order to construct $\boldsymbol{k}_\alpha$. The GP model should be defined with a smooth kernel when it is initialized. A typical choice is a squared exponential kernel, which is what was chosen for the implementation in Chapter 4.

Algorithm 3: `SafeLearningCBF` with random exploration

```
Inputs: True dynamics $F(\boldsymbol{x})\underline{\boldsymbol{u}}$,
        Prior $\mathcal{GP}$ dynamics model,
        control barrier function $h$,
        probability threshold $p$,
        time horizon $T$,
        randomly initialized exploration policy $\pi_{\text{exp}}$,
        randomly initialized policy $\pi_0$,
        initial state $\boldsymbol{x}_0$,
        Empty dataset $D_{\text{true}}$ for data sampled from true dynamics,
        Empty dataset $D_{\text{model}}$ for data sampled from model
1  for $n = 1, 2, \dots$ do
       /* Policy optimization */
2      Sample $T$ steps from $\mathcal{GP}$ model using $\pi_{\boldsymbol{\theta},(n-1)}$
3      Store $\{\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}, r_{t+1}\}$ in $D_{\text{model}}$ for all steps $t \leq T$
4      Compute $c^{(r)}(\boldsymbol{x}, \boldsymbol{u})$ for state-action pairs $D_{\text{model}}$ using (3.7)
5      Update policy $\pi_{\boldsymbol{\theta},n}$ using (3.13)
       /* Safe exploration */
6      Initialize state $\boldsymbol{x}_t = \boldsymbol{x}_0$
7      for $t = 1$ to $T$ do
8          $\boldsymbol{u}_t \sim \pi_{\text{exp}}(\boldsymbol{x}_t)$
9          $CBC^{(r)}(\boldsymbol{x}_t, \boldsymbol{u}_t) \leftarrow \mathcal{L}_f^{(r)} h(\boldsymbol{x}_t) + \mathcal{L}_g \mathcal{L}_f^{(r-1)} h(\boldsymbol{x}_t)\boldsymbol{u}_t + \boldsymbol{k}_\alpha \eta(\boldsymbol{x}_t)$
10         $c^{(r)}(\boldsymbol{x}_t, \boldsymbol{u}_t) \leftarrow \sqrt{\frac{p}{1-p}} Var^{\frac{1}{2}}\{CBC^{(r)}(\boldsymbol{x}_t, \boldsymbol{u}_t)\} + \zeta - E\{CBC^{(r)}(\boldsymbol{x}_t, \boldsymbol{u}_t)\}$
11         if $c^{(r)}(\boldsymbol{x}_t, \boldsymbol{u}_t) < 0$ then
               /* Sample true dynamics */
12             $\boldsymbol{x}_{t+1} = F(\boldsymbol{x}_t)\underline{\boldsymbol{u}_t}$
13             Store $\{\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}\}$ in $D_{\text{true}}$
14         end
15     end
16     Update $\mathcal{GP}$ model with safe observations from $D_{\text{true}}$
17 end
```

Algorithm 4: `SafeLearningCBF` with global optimization

```
Inputs: True dynamics $F(\boldsymbol{x})\underline{\boldsymbol{u}}$,
        prior $\mathcal{GP}$ dynamics model,
        control barrier function $h$,
        probability threshold $p$,
        time horizon $T$,
        randomly initialized exploration policy $\pi_{\text{exp}}$,
        randomly initialized policy $\pi_0$,
        initial state $\boldsymbol{x}_0$,
        empty dataset $D_{\text{true}}$ for data sampled from true dynamics,
        empty dataset $D_{\text{model}}$ for data sampled from model
1  for $n = 1, 2, \ldots$ do
       /* Policy optimization */
2      Sample $T$ steps from $\mathcal{GP}$ using $\pi_{\boldsymbol{\theta},(n-1)}$
3      Store $\{\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}, r_{t+1}\}$ in $D_{\text{model}}$ for all steps $t \leq T$
4      Compute $c^{(r)}(\boldsymbol{x}, \boldsymbol{u})$ for state-action pairs in $D_{\text{model}}$ using (3.7)
5      Update policy $\pi_{\boldsymbol{\theta},n}$ using (3.13)
       /* Safe exploration */
6      initialize state $\boldsymbol{x}_t = \boldsymbol{x}_0$
7      for $t = 1$ to $T$ do
8          $\boldsymbol{u}_t \sim \pi_{\text{exp}}(\boldsymbol{x}_t)$
9          $CBC^{(r)}(\boldsymbol{x}_t, \boldsymbol{u}_t) \leftarrow \mathcal{L}_f^{(r)} h(\boldsymbol{x}_t) + \mathcal{L}_g \mathcal{L}_f^{(r-1)} h(\boldsymbol{x}_t)\boldsymbol{u}_t + \boldsymbol{k}_\alpha \eta(\boldsymbol{x}_t)$
10         $c^{(r)}(\boldsymbol{x}_t, \boldsymbol{u}_t) \leftarrow \sqrt{\frac{p}{1-p}} Var^{\frac{1}{2}}\{CBC^{(r)}(\boldsymbol{x}_t, \boldsymbol{u}_t)\} + \zeta - E\{CBC^{(r)}(\boldsymbol{x}_t, \boldsymbol{u}_t)\}$
11         if $c^{(r)}(\boldsymbol{x}_t, \boldsymbol{u}_t) \geq 0$ then
               /* Apply safety filter */
12             Solve QP in (3.17) to find safe candidate $\boldsymbol{u}_t$
13         end
           /* Sample true dynamics */
14         $\boldsymbol{x}_{t+1} = F(\boldsymbol{x}_t)\underline{\boldsymbol{u}_t}$
15         Store $\{\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}\}$ in $D_{\text{true}}$
16     end
17     Update $\mathcal{GP}$ model with safe observations from $D_{\text{true}}$
18 end
```

4 | Simulations and results

In this chapter, aspects of the safe learning framework is tested in simulation by running experiments on a practical implementation of Algorithm 3. First, the experimental setup of the simulation of two second-order systems is explained. Then, results of experiments on using probabilistic safety constraints for safe exploration, model-free and model-based safety-constrained reinforcement learning are presented.

4.1 Mountain Car system

In order to test the safe learning framework, it is necessary to evaluate it on some environment consisting of simulated system dynamics, as well as a reward function to gauge the performance of a control policy. A Mountain Car environment, which is a second-order dynamical system that simulates the behavior of a car driving up a mountainside, is one of the environments used to test the framework. The Mountain Car environment is a version of the `MountainCar-Continuous` environment from OpenAI Gym [61], which is a continuous control task often used to test reinforcement learning algorithms.

The Mountain Car system simulates the dynamics of a car driving up a mountainside. The goal of the control task is for the car to reach a desired goal state on the top of one side of the mountain. The amount of force that can be applied to the car by a control policy is restricted by design, in order to disallow a trivial solution of a control policy which simply applies enough force for the car to drive straight up. Instead, a policy must be found that learns to "swing" the car back and forth in order to gain enough momentum to reach the goal state at the top.

The Mountain Car environment is illustrated in Figure 4.1. Force can be applied to the car so that it "swings" back and forth from one side of the mountain to the other. The goal state, which defines the desired goal of the control task and the successful end of an

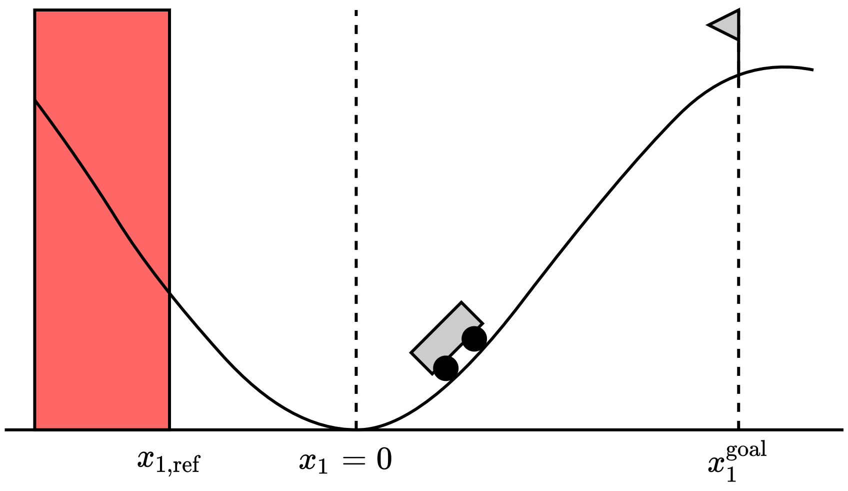


Figure 4.1: Mountain Car environment.

episode, is defined by a value along the x_1-axis and is illustrated by a flag at the point x_1^{goal} in Figure 4.1.

The Mountain Car system is a second-order dynamical system, and can be expressed on control-affine form as

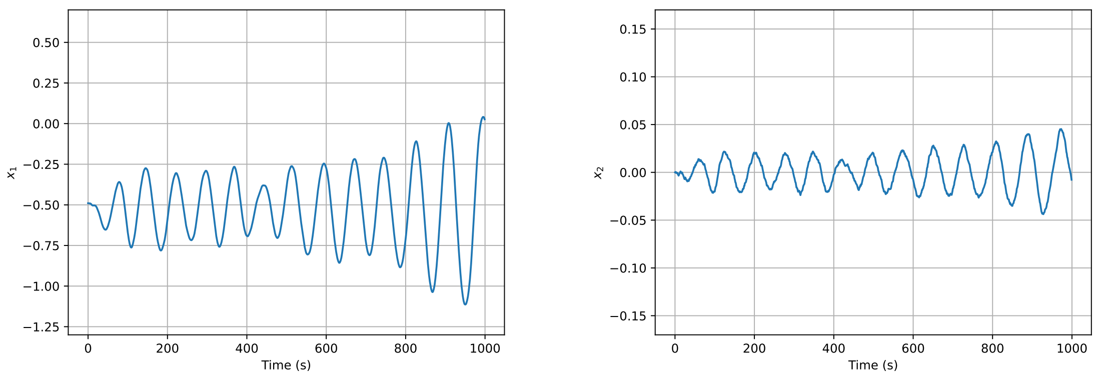
$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = f(\boldsymbol{x}) + g(\boldsymbol{x})u = \begin{bmatrix} x_2 \\ -\beta \cos(3x_1) \end{bmatrix} + \begin{bmatrix} 0 \\ \alpha \end{bmatrix} u, \quad \alpha, \beta > 0, \tag{4.1}$$

where x_1 is the position, x_2 the velocity, and u is the one-dimensional control action input. u represents the force applied to the car, and in the implementation it is limited to a range $[-u_{\text{max}}, u_{\text{max}}]$ in order to disallow the car to simply drive up the right side of the mountain to reach the goal. The coefficients were chosen as $\alpha = 3.0$ and $\beta = 0.0025$, while the maximum force was chosen as $u_{\text{max}} = 1.0$.

A sample trajectory of the Mountain Car dynamical system in (4.1) using action samples from a random policy over 1000 time steps is shown in Figure 4.2.

Control barrier function

A safe set for the Mountain Car system can be chosen as the compliment to an unsafe linear region of the position state space. An unsafe region can then be defined as every point along the position-axis the Mountain Car must avoid to the left of some reference $x_{1,\text{unsafe}}$, as illustrated by the red shaded area in Figure 4.1. This unsafe region can be expressed as the range $[-\infty, x_{1,\text{unsafe}}]$, and as can be seen it is independent of the velocity of the Mountain Car system at any point. A control barrier function can be defined based



(a) Position state. **(b)** Velocity state.

Figure 4.2: Sample simulation from the Mountain Car environment.

on the unsafe region, and expressed as

$$h(\boldsymbol{x}) = x_1 - x_1^{ref} \quad , \quad C = \{\boldsymbol{x} \in \mathcal{X} \mid h(\boldsymbol{x}) \geq 0\} \tag{4.2}$$

The control barrier function in (4.2) is of relative degree $r = 2$, since the control action input u appears in the second derivative, as can be observed from (4.1). The unsafe region was defined by choosing $x_{1,\text{unsafe}} = -0.8$.

Reward function

The control problem for the Mountain Car environment can be defined as reaching the reference goal state at the right side of the mountain. As such, the reward function used for reinforcement learning should be related to the goal state. The reward function chosen, which is on the same form as the one used in [61], is given by

$$\begin{aligned} R(\boldsymbol{x}) = -\epsilon u^2,\ \epsilon > 0 \quad \forall \quad & x_1 < x_1^{\text{goal}} \\ R(\boldsymbol{x}) = 100 \quad \forall \quad & x_1 \geq x_1^{\text{goal}}, x_2 \geq x_2^{\text{goal}} = 0. \end{aligned} \tag{4.3}$$

The reward function in (4.3) is designed so that the environment returns a negative reward whenever the car has not made it up the mountainside to the goal. As can be observed from (4.3), the reward function is proportional to the negative of the squared action for any position state to the left of the goal, with a form such that a larger action input gives a more negative reward. Furthermore, only in the case where the goal position is reached, and when the velocity of the car is zero, will the agent receive a positive reward of 100. The coefficient for the reward function was chosen as $\epsilon = 0.1$.

4.2 Inverted pendulum system

The other environment used to test the safe learning framework is an inverted pendulum environment. The inverted pendulum is a second-order continuous dynamical system system, which simulates a pendulum swinging around a fixed point. An optimal policy will in this case learn to stabilize the pendulum at a reference point, for instance at the top of its trajectory.

The inverted pendulum environment is illustrated in Figure 4.3, where a torque can be applied by a control policy to swing the pendulum, and where the angle θ denotes the angular deviation from the vertical resting position of $\theta = 0$.

The pendulum system can be expressed on control-affine form as

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\theta} \\ \dot{\omega} \end{bmatrix} = f(\boldsymbol{x}) + g(\boldsymbol{x})u = \begin{bmatrix} \omega \\ -\frac{g}{l}\sin\theta \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix} u, \tag{4.4}$$

where θ is the pendulum angle and ω is the angular velocity, and where g, m and l is the mass and length of the pendulum and gravity constant, respectively. u denotes the one-dimensional control action input, and it is limited to a range $[-u_{\max}, u_{\max}]$. The coefficient were chosen as $g = 10.0$, $m = 1.0$ and $l = 1.0$, while the maximum torque was chosen as $u_{\max} = 7.5$.

Control barrier function

A safe set for the inverted pendulum environment can be chosen as the compliment to an unsafe radial region of the angle state space, as illustrated by the shaded red area in Figure 4.3. The unsafe region can then be defined as every point along the angle-axis that the pendulum system must avoid on either side of a reference θ_{unsafe}, independent of the angular velocity at this point. This unsafe region can be expressed as the range $[\theta_{\text{unsafe}} - \Delta\theta, \theta_{\text{unsafe}} + \Delta\theta]$. A control barrier function can then be defined based on the radial unsafe region, and expressed as

$$h(\boldsymbol{x}) = \cos(\Delta\theta) - \cos(\theta - \theta_{\text{unsafe}}). \tag{4.5}$$

As in Section 4.1, since the inverted pendulum system is of second-order, the control barrier function is of relative degree $r = 2$. The unsafe region was defined by choosing $\theta_{\text{unsafe}} = 45^\circ$ and $\Delta\theta = 22.5^\circ$.

Reward function

For a reference tracking problem such as the one defined for the inverted pendulum

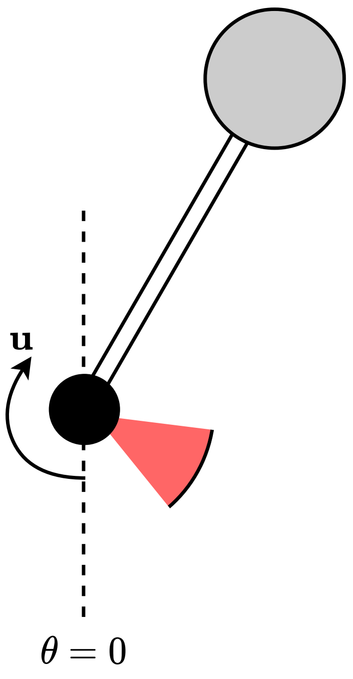


Figure 4.3: Inverted pendulum environment.

environment, there are many potential reward functions that can be used in order to incentivize the system to learn a desired behavior. The reward function should be designed such that a position close to a static reference yields high reward, while a position far away from the reference gives little or no reward. Alternatively, a reward function similar to the one defined for the Mountain Car environment can be defined, where every reward apart from the goal state yields a negative value.

One choice in the case of the pendulum system is a negative squared exponential, given by

$$R(\boldsymbol{x}) = \epsilon_1 e^{-\epsilon_2(\theta-\theta_{ref})^2} \quad , \quad \epsilon_1, \epsilon_2 > 0. \tag{4.6}$$

In (4.6), the angle from the state vector is utilized to denote the reference error, and the squaring of the exponential ensures that the reward remains positive in cases where $\theta - \theta_{ref} < 0$. The coefficients for the reward function were chosen as $\epsilon_1 = 5.0$ and $\epsilon_2 = 1.0$.

4.3 Learning control policies

The policy optimization scheme was implemented as an actor-critic method, and an augmented, practical version of Algorithm 1 was implemented to solve the safety-constrained policy update in (3.13). The neural networks used to parameterize the policy and state-value function were implemented using PyTorch [3]. The architecture of the networks are

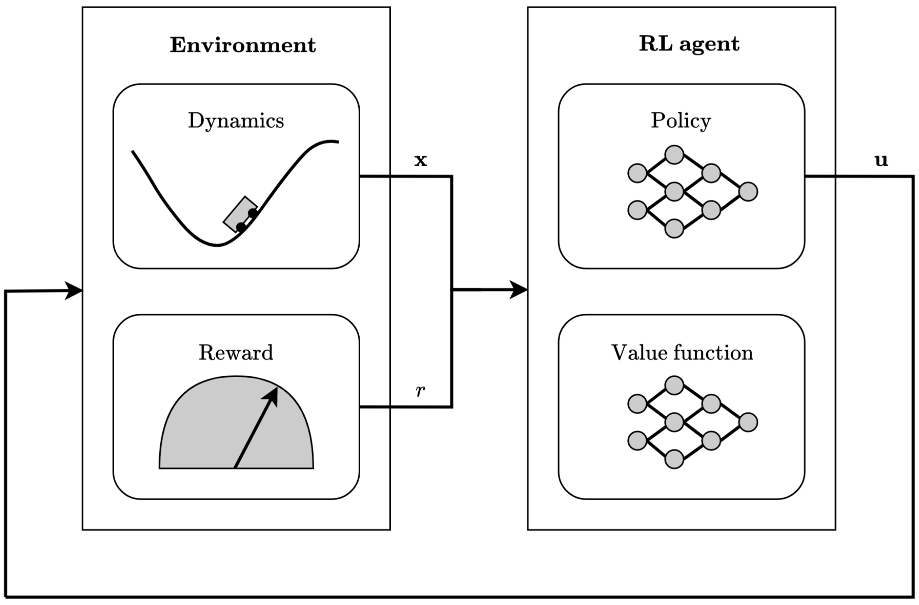


Figure 4.4: Control system architecture for safe learning framework.

summarized in Table 4.1 and Table 4.2. The structure of the reinforcement learning loop is shown in Figure 4.4.

Layer type	Input shape	Output shape	Activation function
Mean layer	400	1	Linear$(\cdot)$
Variance layer	400	1	Softplus$(\cdot) + 1 \times 10^{-5}$

Table 4.1: Actor neural network architecture.

Layer type	Input shape	Output shape	Activation function
Linear layer	400	1	Linear$(\cdot)$

Table 4.2: Critic neural network architecture.

Both neural networks were trained using Adam optimizers. The learning rate for the actor was set as 4×10^{-5} and the learning rate for the critic set as 1×10^{-5}.

The inputs to the neural networks were featurized and scaled for both the policy and the state-value function. 1000 samples were randomly collected from the environment. A scaling function was then fitted based on this data, such that when input data to the network is passed through this scaling function it is transformed to a feature vector whose mean and standard deviation is 0.0 and 1.0, respectively. Additionally, the input vector, which originally is of dimension 2, was featurized. The featurization process involves sampling squared exponential kernels fitted on the scaled sample data, and creating a *feature union* containing 400 input features. Featurizing the inputs to the neural network can be beneficial, since the number of features used for training is increased. In turn, this will result in a larger and more flexible neural network that may yield more stable training.

Episode termination criteria were defined in order to impose requirements on when an episode is considered finished and training should be terminated. In case of the Mountain Car environment, training terminates when the car reaches the goal state and the velocity is near zero. In case of the inverted pendulum, the termination criteria were defined as the angle being outside a range defined by a cone about the reference value, or when the pendulum system entered the unsafe region.

4.4 Verifying probabilistic safety constraints

The probabilistic formulation of the safety constraints are used to verify if a state-action pair is safe, such that actions can be applied to the true system dynamics in order to actively gather data to improve a statistical model of the dynamics.

In order to quantify the success of the safe exploration scheme in rejecting unsafe actions and thereby prevent unsafe exploration on the true system dynamics, the rate at which the safety verification procedure makes erroneous predictions of the safety of a state-action pair can be recorded. There are two different types of errors that can be made during the safety verification process during exploration: Safe actions can be wrongfully rejected, and unsafe actions can be wrongfully admitted. These error types can be considered analogous to false positives and false negatives often used in the testing of statistical hypotheses, and are related to the accuracy of the probabilistic safety constraint and its effectiveness in stopping unsafe actions and letting through safe actions.

Figure 4.5 illustrates the safety constraint error types, as well as the success types. The expression $h(\boldsymbol{x}) \geq 0$ denotes that the resulting next state produced by applying an action $\boldsymbol{u}$ on the true dynamics is safe, thereby indicating the action is safe. Similarly, the expression $h(\boldsymbol{x}) < 0$ indicates it is unsafe. Likewise, the expression $c^{(r)}(\boldsymbol{x}, \boldsymbol{u}) < 0$ denotes that the state-action pair is admitted by the safe exploration scheme, and thereby deemed safe by the probabilistic safety constraint verification. Similarly, the expression $c^{(r)}(\boldsymbol{x}, \boldsymbol{u}) \geq 0$ indicates the pair is rejected and thereby deemed unsafe.

The percentage of unsafe actions that are wrongfully admitted by the safe exploration scheme is termed the *damage percentage*. The damage percentage is of particular interest, as it is typically a more severe error measure than the percentage of safe actions that are wrongfully rejected. This is because unsafe actions applied to true dynamical systems in safety-critical scenarios can cause harmful consequences, while stopping safe actions from being applied may make the model regression less effective but does not cause harm to the system.

	$h(\mathbf{x}) \geq 0$	$h(\mathbf{x}) < 0$
$c^{(r)}(\mathbf{x}, \mathbf{u}) < 0$	Admit safe state-action pair	Admit unsafe state-action pair
$c^{(r)}(\mathbf{x}, \mathbf{u}) \geq 0$	Reject safe state-action pair	Reject unsafe state-action pair

Figure 4.5: Safety constraint error types.

4.5 Experiment 1: Safe exploration

In this experiment, the safe exploration scheme was tested using the Mountain Car system. During safe exploration, safety verification is utilized to attempt to safely learn a Gaussian process. Unsafe state-action pairs that are encountered should ideally be rejected while actively gathering data on the real system for use in the Gaussian process regression. In order to validate the safe exploration scheme, a fixed set of state-action pairs were used as a dataset.

500 state-action pairs were sampled on the true Mountain Car system dynamics using a randomized control policy with a fixed seed, and saved for use in the validation of the safe exploration scheme to ensure consistency in the state-action pairs evaluated each episode. A region of the state space was sampled such that the system crossed into the unsafe region. A successful safe exploration scheme should in this case reject unsafe actions.

The performance of the safe exploration scheme with probabilistic safety constraints for different values of the inter-triggering threshold variable ζ was tested by fixing the probability threshold at $p = 0.95$ and doing one pass of Gaussian process regression to inform the prior model. The amount of admitted safe, admitted unsafe, rejected safe and rejected unsafe actions were then counted, and recorded as a percentage of safety constraint errors over the dataset of 500 state-action pairs. The result is shown in Table 4.3.

Similarly, the performance of the safety exploration scheme with probabilistic safety constraints was tested for different values of the probability threshold p, while fixing the inter-triggering threshold variable at $\zeta = 0.1$. The amount of admitted safe, admitted unsafe, rejected safe and rejected unsafe actions were then counted, and recorded as a percentage of safety constraint errors over the dataset of 500 state-action pairs. The result is shown in Table 4.4.

ζ	Reject safe	Reject unsafe	Admit unsafe	Admit safe
0.01	0.0%	9.8%	47.2%	43.0%
0.1	10.0%	57.0%	0.0%	33.0%
1.0	43.0%	57.0%	0.0%	0.0%
10.0	43.0%	57.0%	0.0%	0.0%

Table 4.3: Safety constraint error percentages for different threshold values of ζ.

p	Reject safe	Reject unsafe	Admit unsafe	Admit safe
0.99	10.0%	57.0%	0.0%	33.0%
0.95	10.0%	57.0%	0.0%	33.0%
0.80	10.0%	57.0%	0.0%	33.0%

Table 4.4: Safety constraint error percentages for different threshold values of p.

The safety constraint error rates were then tested by episodically gathering data safely and improving the Gaussian process model. Safety verification was performed on the fixed dataset each episode, and the error percentages recorded. The result is shown in Figure 4.6.

Note that in this instance, the safety constraint errors are denoted as ratios of the total number of safe and unsafe actions. This means that "Reject safe" denotes the percentage of safe actions that are rejected out of the total number of safe actions, and "Reject unsafe" denotes the percentage of safe actions that are rejected out of the total number of unsafe actions. Likewise, "Admit safe" denotes the percentage of safe actions that are admitted out of the total number of safe actions, and "Admit unsafe" denotes the percentage of safe actions that are admitted out of the total number of unsafe actions.

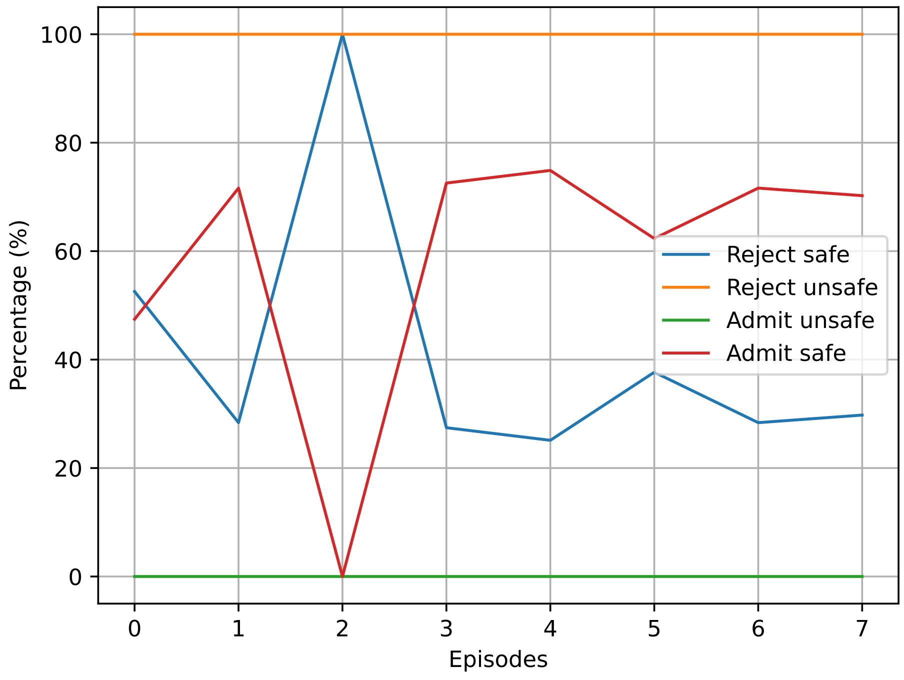


Figure 4.6: Safety constraint errors as ratios of total number of safe and unsafe state-action pairs.

4.6 Experiment 2: Model-free policy optimization

In this experiment, a model-free version of Algorithm 3 was tested using the Mountain Car system. A Gaussian process model was trained in parallel each episode using the safe exploration scheme, and the model was used for safety verification, but not to predict next states during reinforcement learning.

The safe learning framework was run for 8 episodes until the episodic returns converged. Figure 4.7 shows the returns when optimizing an unconstrained policy and using the true dynamics for optimization. Figure 4.8 shows the trajectory of the position x_1 when applying the learned, unconstrained policy to the true Mountain Car dynamics.

Figure 4.9 shows the returns when optimizing a safety-constrained policy and using the true dynamics for optimization, with $p = 0.95$, $\zeta = 0.1$ and a Lagrangian multiplier $\lambda = 0.05$. Figure 4.10 shows the trajectory of the position x_1 when applying the learned, safety-constrained policy to the true Mountain Car dynamics.

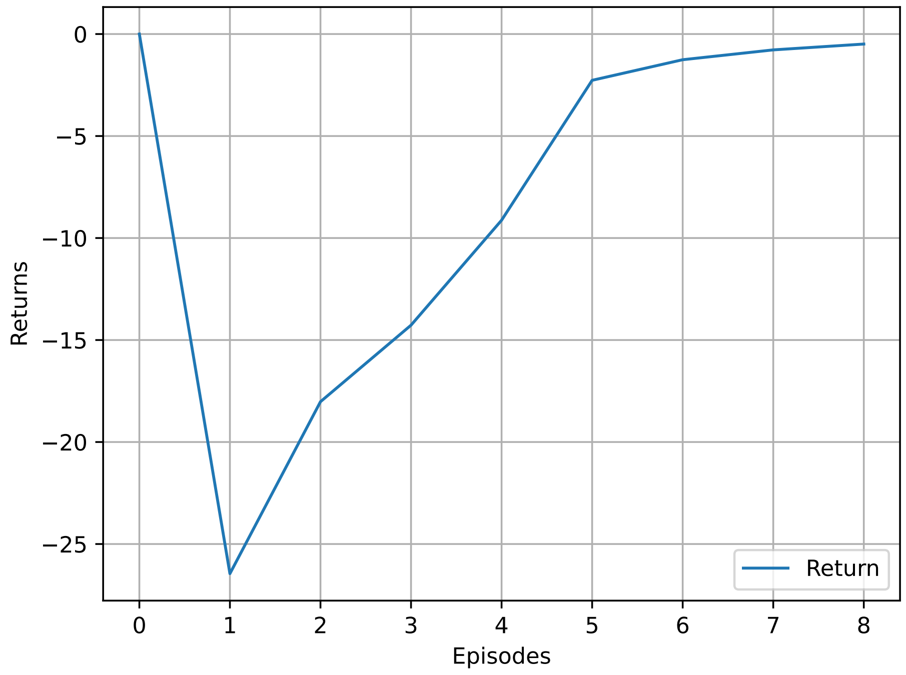


Figure 4.7: Episodic returns for unconstrained model-free learning on Mountain Car system.

4.7 Experiment 3: Model-based policy optimization

In this experiment, the model-based version of Algorithm 3 was tested using both the Mountain Car system and inverted pendulum system. A Gaussian process model was trained in parallel each episode using the safe exploration scheme, and the model was used for both safety verification and to augment the returns for the policy optimization.

For the Mountain Car system, the safe learning framework was run for 8 episodes until the until the episodic returns converged. Figure 4.11 shows the episodic returns when optimizing an unconstrained policy and using the GP model, with $p = 0.95$, $\zeta = 0.1$ and a Lagrangian multiplier $\lambda = 0.05$. Figure 4.12 shows the position values x_1 when applying the learned, unconstrained policy on the true Mountain Car dynamics.

Further, Figure 4.13 shows the episodic returns when optimizing a safety-constrained policy and using the GP model, and Figure 4.14 shows the position values x_1 when applying the learned, safety-constrained policy on the true Mountain Car dynamics.

For the inverted pendulum system, the safe learning framework was run for 100 episodes until the episodic returns converged. Figure 4.15 shows the episodic returns when opti-

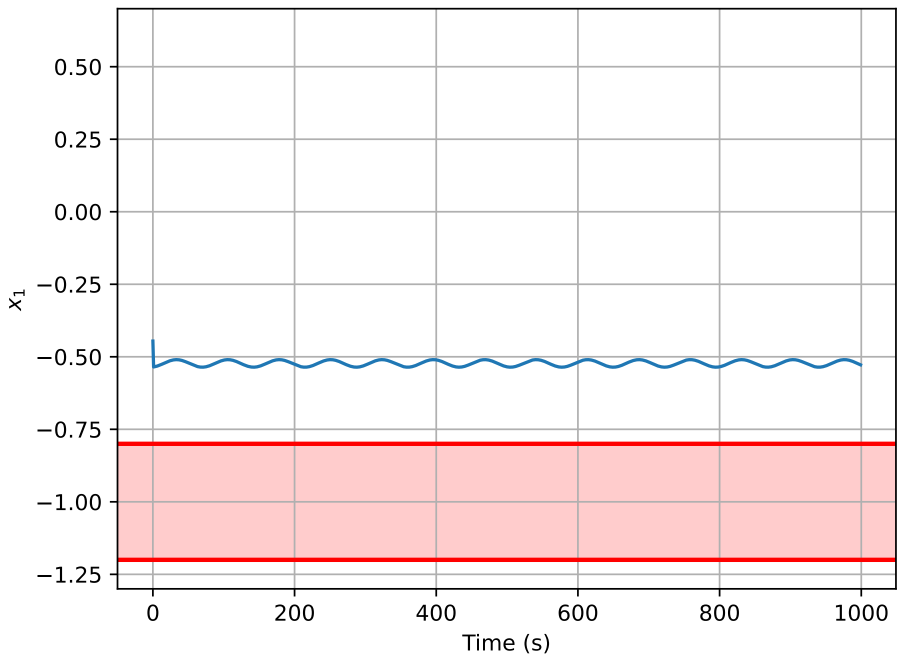


Figure 4.8: x_1 values for learned model-free unconstrained policy on Mountain Car system.

mizing a safety-constrained policy and using the GP model, with $p = 0.95$, $\zeta = 0.1$ and a Lagrangian multiplier $\lambda = 0.1$. Figure 4.16 shows the trajectory of the angle θ when applying the learned, safety-constrained policy to the true pendulum dynamics.

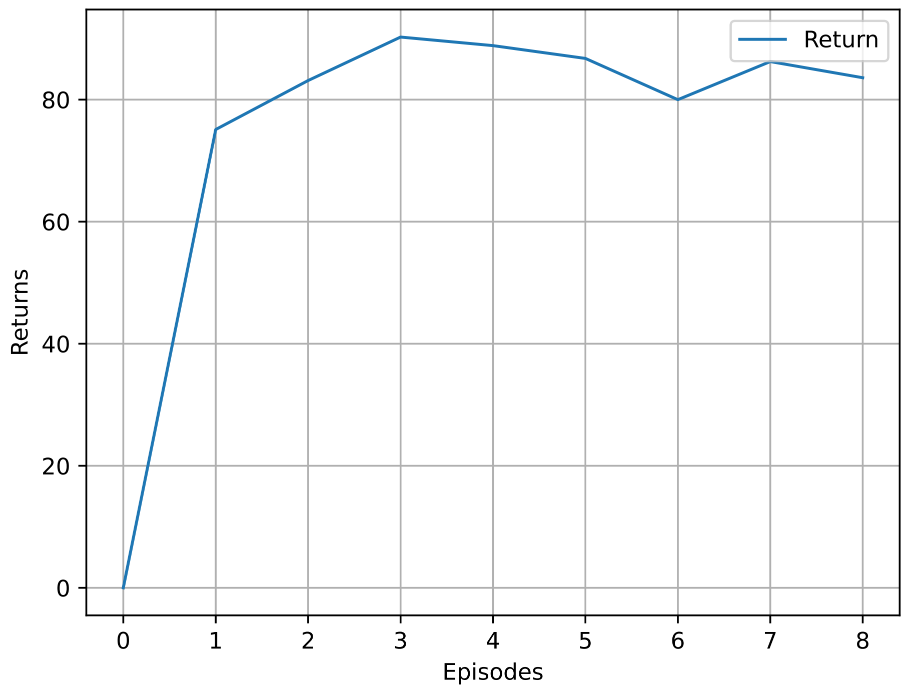


Figure 4.9: Episodic returns for safety-constrained model-free learning on Mountain Car system.

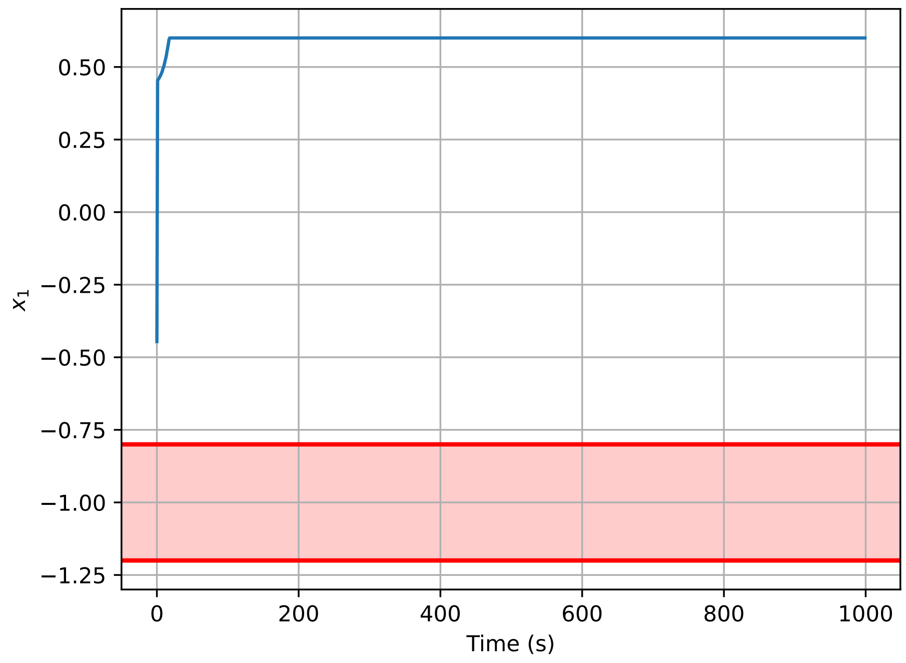


Figure 4.10: x_1 values for learned model-free constrained policy on Mountain Car system.

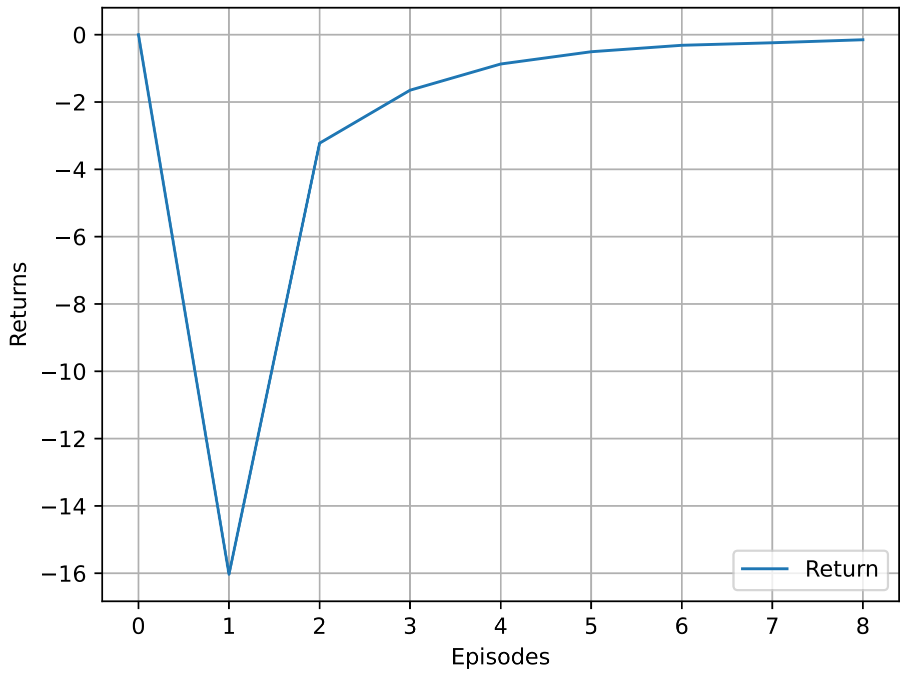


Figure 4.11: Episodic returns for unconstrained model-based learning on Mountain Car system.

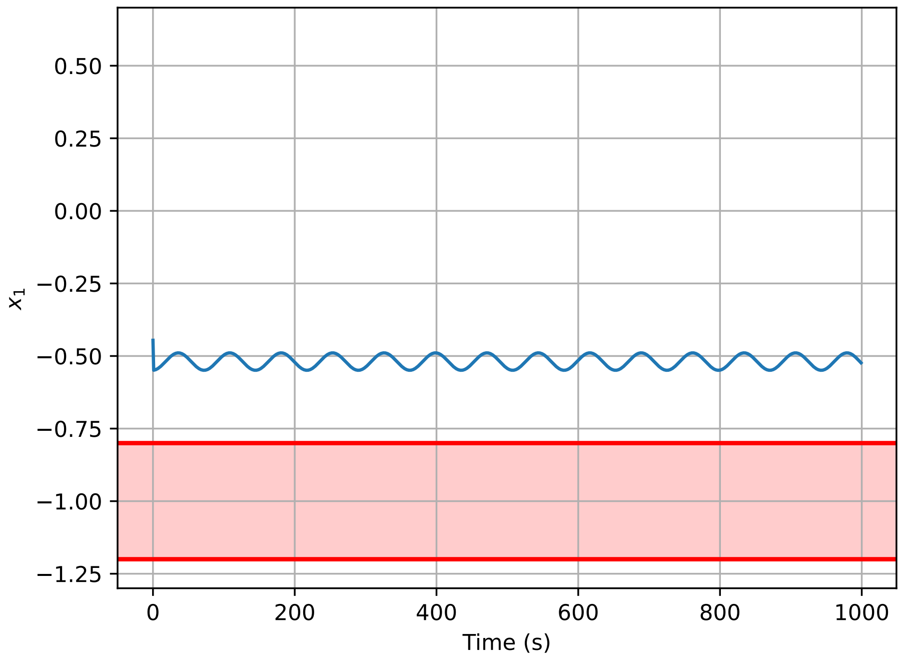


Figure 4.12: x_1 values for learned model-based unconstrained policy on Mountain Car system.

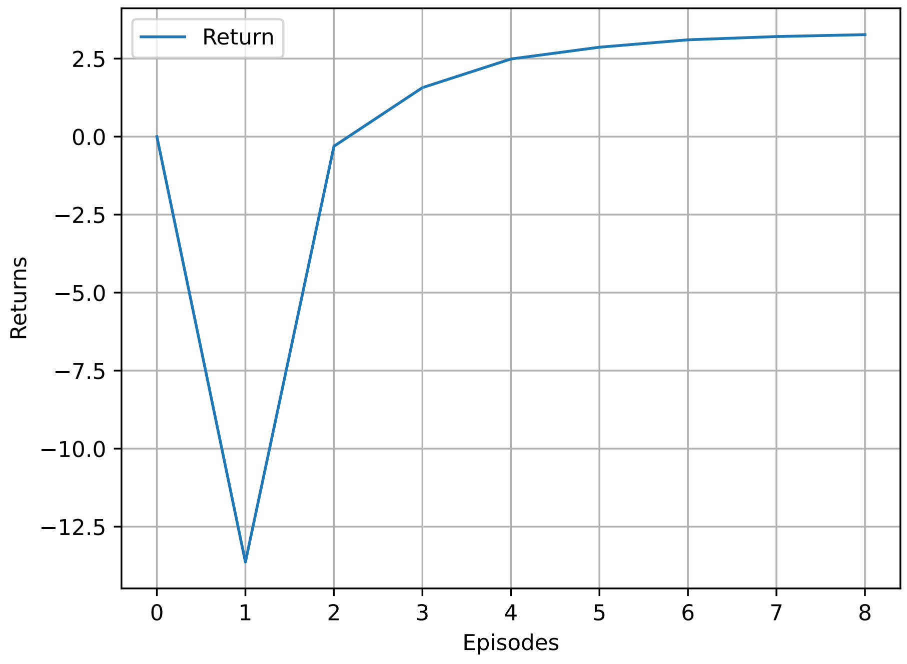


Figure 4.13: Episodic returns for safety-constrained model-based learning on Mountain Car system.

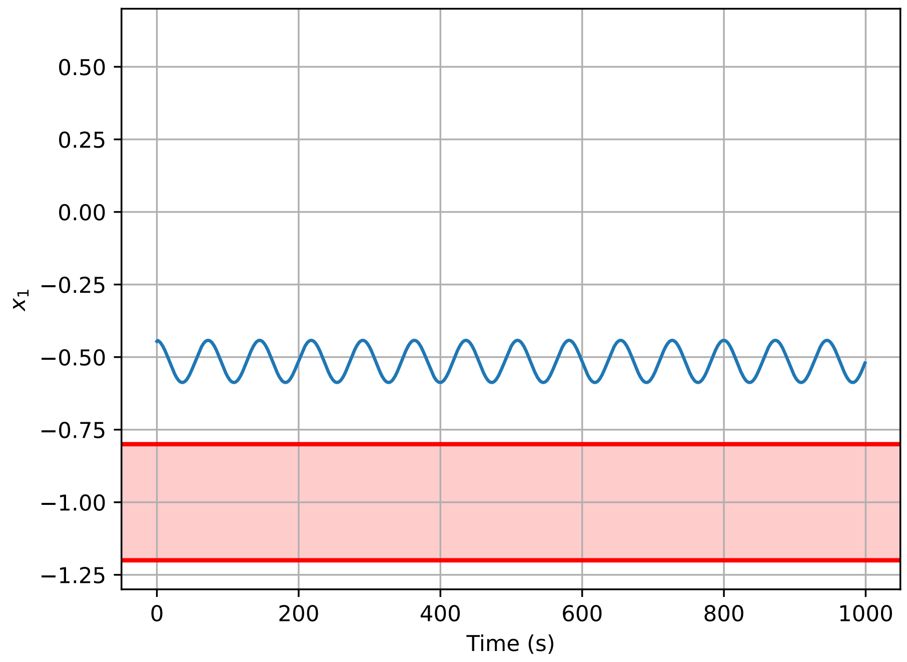


Figure 4.14: x_1 values for learned model-based constrained policy on Mountain Car system.

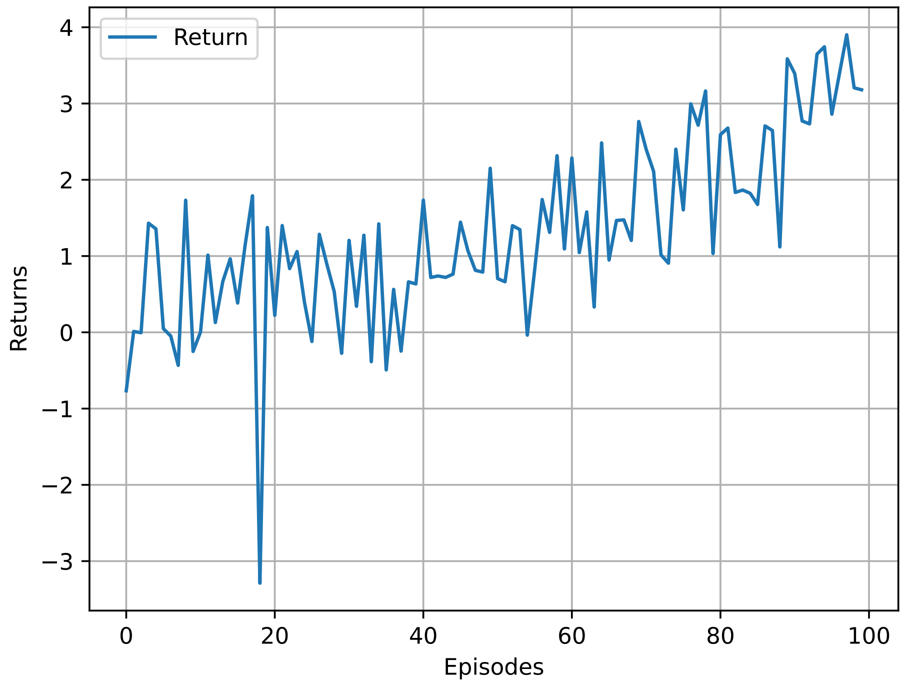


Figure 4.15: Episodic returns for model-based safety-constrained learning on inverted pendulum system.

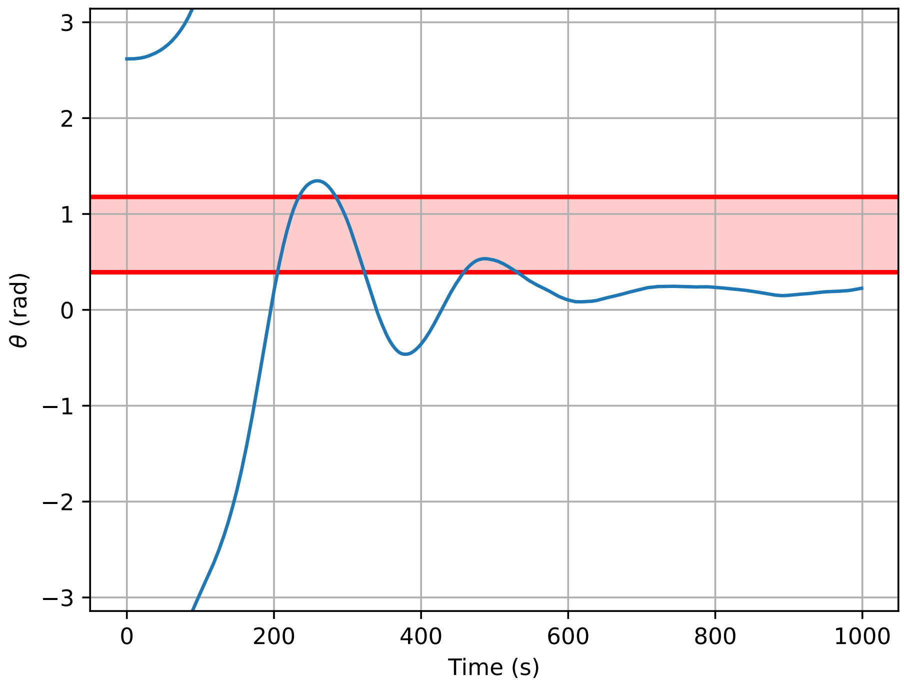


Figure 4.16: θ values for learned model-based policy on inverted pendulum system.

5 | Discussion

In this chapter, the experimental and theoretical results will be discussed. The results of the experiments on the second-order systems are discussed, and the validity and limitations of the theoretical algorithm are considered. The chapter is concluded with suggestions for further work.

5.1 Experimental results

The results from experiment 1 confirm that the method for safe exploration is able to successfully reject most unsafe actions. It can be observed from Figure 4.6 that the damage percentage, indicated by the percentage of admitted unsafe actions, is zero throughout the safe exploration period. Out of the total number of unsafe actions, none were admitted. When considering all state-action pairs in the validation dataset, this means no unsafe actions were wrongfully passed off as safe by the probabilistic safety constraints defined in (3.7) during safety verification.

Experiment 1 also reveals that the safe exploration scheme erroneously rejects some safe actions. As the posterior estimate of the Gaussian process model is updated episodically, the damage percentage remains at zero, while the percentage of wrongfully rejected safe actions fluctuate. As can be observed from Figure 4.6, around 30% of the total number of safe actions are rejected by the probabilistic safety constraints during exploration from episode 3 throughout episode 7. This shows a tendency of the safe exploration scheme to act in a somewhat overly passive way with regards to action admittance. This tendency to reject too many actions may indicate that the thresholds selected for the probabilistic constraint formulation yield a safety condition whose probabilistic bounds are too loose, in the sense that the boundary point at which the safety constraint $c^{(r)}(\cdot)$ in (3.7) switches sign does not occur at the point where a state trajectory crosses into the unsafe region of the state space. The resulting non-zero false rejection rate may alternatively be an effect of insufficient accuracy of the Gaussian process model estimate, which affects the predictions produced by the control barrier condition $CBC^{(r)}(\cdot)$. Nevertheless, the safe exploration scheme does not categorically reject all actions.

The damage percentage can be considered the most important measurement when evaluating the practical feasibility of the safe exploration method, since it describes an error type which generally is more critical to prevent in order to ensure system safety. Analogous to false negative predictions, falsely predicting that a large amount of unsafe actions are safe is likely unacceptable in safety-critical scenarios. Conversely, false positive analogues are less likely to cause major issues with regards to system safety. Depending on the exploration policy used, a high percentage of safe action rejections may make the safe exploration scheme less efficient, since unsafe actions can be encountered more often. It can also cause the data collected for model improvement, indicated by D_{model} in Algorithm 3, to become sparse, which can result in poor posterior model updates. In experiment 1, no unsafe actions were admitted, and less than 50% of the total number of safe actions were regularly rejected, which suggests that using probabilistic safety constraints to verify the safety of state-action pairs during exploration may be feasible.

The results in Table 4.4 show that the effect of the probability threshold p on the values of the probabilistic safety constraints is limited. For various values of p, the error percentages stay identical. On the one hand this result is expected, since the probabilistic safety constraint formulation in (3.7) holds when the kernel of the Gaussian process used is sufficiently smooth, and thereby the probability threshold can be selected as any value. On the other hand, as can be observed from the expression for the probabilistic safety constraint on standard form in (3.7), the coefficient $\sqrt{\frac{p}{1-p}}$ will not affect the value of $c^{(r)}(\cdot)$ considerably if the variance of the control barrier condition $CBC^{(r)}(\cdot)$ is low.

Table 4.3 illustrates that the inter-triggering time step variable ζ may have a more substantial effect than p on the values of the safety constraints used for safety verification. As can be observed from the recorded safety constraint error percentages in Table 4.3, a small value of ζ leads to almost all actions being admitted. Furthermore, it can be observed that a larger value for the probability threshold ζ results in less admittance of both safe and unsafe actions, to the point where all actions are rejected. The results also suggest that if the sampling time interval is sufficiently short, meaning the inter-triggering time between safety constraint value calculation steps is small, a low ζ value will perhaps provide more accurate safety verification. From Table 4.3, it can also be seen that a lower value for ζ will result in more admittance of actions overall. However, a value of $\zeta = 0.01$ results in more admitted unsafe action and a significant damage percentage. Thus, tuning the threshold ζ

for safety verification in relation to the sampling time can perhaps result in fewer safety constraint errors.

The results of experiment 2 show that safety-constrained policy optimization is possible in a model-free setting, as can be observed from the converging episodic returns in both Figure 4.7 and Figure 4.9. Furthermore, it can be observed from Figure 4.10 that the framework is able to learn a policy that reaches the Mountain Car environment goal state without entering the unsafe region of the state space, which is denoted by a shaded red region. However, Figure 4.8 reveals that the learned policy does not always drive the system to the goal state, even without safety constraints that augment the objective function. This effect may be due to the stochastic nature of the parameterized policy and the resulting complexity of training deep reinforcement learning agents. Furthermore, the impact of the safety constraints on the optimization objective may be dependent on the scale of the term $\lambda c^{(r)}(\cdot)$ in relation to the cumulative reward expression used in the objective.

Experiment 3 shows that safety-constrained policy optimization also may be possible in a model-based setting, however the learned policies do not always perform well. As can be seen from Figure 4.11 and Figure 4.9 for the Mountain Car system, as well as from Figure 4.15 for the inverted pendulum system, the episodic returns tend to converge over time. The learned model-based policies for the Mountain Car system manage to avoid the unsafe region. This is the case both for the unconstrained and safety-constrained policies, as can be seen from Figure 4.12 and Figure 4.10. These trajectories do reveal, however, that the policies are not sufficiently optimal and as a consequence are not able to reach the goal state.

Furthermore, experiment 3 reveals that the learned model-based policy for the inverted pendulum system is not able to avoid the unsafe region of the state space. Although the returns in Figure 4.15 increase over time, as can be observed from Figure 4.16, the pendulum enters the unsafe region denoted in the figure by a shaded red region.

A challenge related to the practical implementation of the model-based safe learning framework is to learn an equally good estimate of the unknown system dynamics throughout the entire state space. The discrepancies observed in Figure 4.6, where not all safe actions were admitted, can therefore be related to the accuracy of the posterior Gaussian process model. Likewise, the result of the learned policies in the closed-loop systems sometimes being unable to avoid unsafe regions can suggest that the one-step predictions of the Gaussian process model are not accurate enough. In general, this is considered a significant issue in model-based reinforcement learning.

Some of the challenges can stem from difficulties related to training a matrix variate Gaussian process model, which is a complex model representation. Training separate GP models per system dimension, as an alternative to matrix variate process regression, is not feasible when utilizing the GP model to calculate the safety constraints in (3.7), since it is necessary to capture the dependencies between the components in the drift and gain terms of the control-affine system model. The predictions returned by the expression for the probabilistic control barrier condition $CBC^{(r)}(\cdot)$ are defined based on the assumption that the structure of the control-affine model of the system dynamics is on a matrix variate form. As such, poor model estimates are likely to affect the performance of learned policies in a model-based setting, and can also affect the accuracy of the predictions made by the control barrier condition on probabilistic form. However, experiment 1 show that safe exploration can be feasible, while experiment 2 show that safety-constrained policy optimization can be implemented in practice, and although issues are encountered regarding the performance of the learned policies, it may be a promising way forward.

The effects of a poor model estimate can perhaps be mitigated by using a different type of Gaussian process model, such as a Coregularization model [38]. Alternatively, a neural network can be used to estimate the unknown dynamics instead of a stochastic process. Such a function approximator can be based on a Bayesian neural network [62], where all the network parameters are represented by separate probability distributions. The Bayesian neural network model can then be used to perform variational inference, which will result in approximate predictions of the system dynamics defined by a mean and variance. Such a model may more easily capture unknown dynamics from limited training data due to the flexibility in the neural network formulation, and some of the issues regarding training a matrix variate Gaussian process can perhaps be avoided.

5.2 Theoretical results

The framework for safe learning presented in Chapter 3 allows a reinforcement learning agent to adapt their model of the unknown system dynamics during the learning phase through a process of safe exploration. In theory, such a way of performing model-based learning will work in unstructured environments with no or little *a priori* knowledge of the system dynamics.

It was demonstrated theoretically that control barrier functions enable the construction of probabilistic safety constraints, which yield a sufficient condition for safety based on a probability measure induced by the posterior estimate of unknown system dynamics.

The proposed framework provides a permissive notion of safety, due to the set invariance imposed in Lemma 1, compared to for instance probabilistic bounds on the asymptotic stability of a system under a control policy. Furthermore, it will be easy to extend the number of control barrier conditions used, as more safety constraints can easily be added when performing safety verification without needing to change the method for safe exploration.

Many methods for safety-critical, learning-based control do not take performance requirements into major consideration [27]. The proposed safe learning algorithm does consider performance requirements, and does not disregard the optimization objective during policy optimization, since the expected return is still the main objective. The policy is instead safety-constrained by the introduction of a surrogate objective, which aims to provide a policy that is both an approximation of the optimal one as well as safe.

It should be noted that when performing safe learning using a model of the unknown dynamics represented by a stochastic system, all bounds are a type of confidence bounds, and conversely no deterministic guarantees can be made. This is likely to be a limitation in all frameworks for safe learning designed for use in situations where the environment is unknown and has to be modeled under uncertainty. Nevertheless, safety can in theory be guaranteed with high probability.

The proposed safe reinforcement learning framework will fit a large range of different problems, since CBFs, and for that matter CLFs which are defined on a similar form, are generic constraints on the behavior of a system under a control policy. In theory, the probabilistic constraints defined by (3.5) can be used as generic safety constraints for any type of chance-constrained optimization problem. Thus, the framework can likely be applied to many different types of control tasks where a control-affine model can be expressed on the form as in (3.1).

5.3 Further work

A natural extension to the safe learning framework would be to consider safety and stability constraints in combination. The proposed method for safe learning considers safety in terms of probabilistic safety constraint satisfaction based on control barrier functions. Significant work has also been done on ensuring safety in terms of stability guarantees, both indirectly through Lyapunov's method, and with more direct approaches by utilizing stability constraints based on control Lyapunov functions. As can be observed from Definition 2 and Definition 3, both CLFs and CBFs are expressed on an equivalent form, and the control Lyapunov condition can also be expressed as probabilistic. Thus, similarly to a CLF-CBF-QP, the combined constraints could be used to optimize a policy that is safety- *and* stability-constrained. However, as pointed out in [34], this would likely require a way to trade off stability and safety, as CLFs impose more restrictive system conditions. Alternatively, the stability constraint formulation could be used to verify the asymptotic stability of state-action pairs to be applied to the real system during exploration, in addition to safety verification.

Another topic for further work is to provide the Gaussian process used to model the unknown dynamics with prior information. This information could either be in the form of a set of state-action pairs sampled in advance of the learning process, or through data gathered from system trajectories recorded in a simulator or from human demonstrations. In any case, if care is taken to ensure the trajectories issued to the prior GP model are safe, they could effectively encode information about safe behavior in the model, which in turn could yield better learning efficiency and more accurate model predictions. Alternatively, a version of the framework could be tested where the Gaussian process is replaced by a Bayesian neural network.

The safe learning framework can also be easily extended to incorporate other types of safety constraints. One of the main points of motivation in the field of safe reinforcement learning is the ability to apply methods to safety-critical systems, which often naturally impose restrictions on how the system is allowed to interact with the environment. Typical constraints on the system state, like maximum velocity, minimum deflection or restricted regions of movement, are often required for safe, real-world operation. However, it would be interesting to exploit the generality of control barrier functions to formulate other conditions for system behavior, like survivability constraints that ensure persistent long term autonomous system operation [63] or energy-aware coverage control [64].

Lastly, it would be interesting to test out a different version of the safe learning framework based on Algorithm 4, which incorporates a safety filter to optimize the exploration policy in order to ensure safe action selection during exploration.

6 | Conclusion

In this thesis, a theoretical framework for safe model-based reinforcement learning using control barrier functions has been proposed and evaluated. Unlike many existing methods, which require accurate models of known system dynamics or impose more restrictive conditions for safety through asymptotic system stability, the proposed framework uses probabilistic safety constraints on the system states to provide high-probability guarantees of safety during exploration. Control barrier functions provide a simple way to define permissive safety constraints.

The safe learning framework utilizes a Gaussian process to model the unknown system dynamics. A safe exploration scheme was proposed, which actively gathers data samples to be used for matrix variate Gaussian process regression. Further, a method for chance-constrained policy optimization was proposed, which uses the model informed by safely collected data in order to optimize a policy subject to probabilistic safety constraints.

Experiments revealed that the method for safe exploration successfully rejected most unsafe actions when actively collecting data. Issues related to the accuracy of one-step predictions made by the Gaussian process model however, meant that not all policies learned by safety-constrained optimization were able to avoid unsafe regions when applied to the true dynamics.

The proposed framework for safe learning shows promise for control barrier functions to be used in order to ensure safety with high probability in unknown environments. The safe learning algorithm may be applied to safety-critical systems, but more research is needed to provide sufficiently accurate statistical model estimates. The use of a permissive notion of safety defined by probabilistic safety constraints opens up possibilities in the future of guaranteeing that autonomous systems learn to operate safely in real-world environments.

References

[1] V. Dhiman, M. J. Khojasteh, M. Franceschetti, and N. Atanasov, *Control barriers in bayesian learning of system dynamics*, 2020. arXiv: `2012.14964 [eess.SY]`.

[2] J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson, "Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration," in *Advances in Neural Information Processing Systems*, 2018.

[3] A. Paszke, S. Gross, F. Massa, A. Lerer, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 8024–8035.

[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* Cambridge, MA, USA: A Bradford Book, 2018.

[5] V. Mnih, A. P. Badia, M. Mirza, A. Graves, *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning*, vol. 48, PMLR, 2016, pp. 1928–1937.

[6] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2722–2730.

[7] D. Silver, A. Huang, C. Maddison, A. Guez, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 2016.

[8] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, pp. 604–609, 2020.

[9] H.-T. L. Chiang, N. Malone, K. Lesser, M. Oishi, and L. Tapia, "Path-guided artificial potential fields with stochastic reachable sets for motion planning in highly dynamic environments," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2015, pp. 2347–2354, 2015.

[10] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with autorl," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2007–2014, 2019.

[11] B. Thananjeyan, A. Balakrishna, U. Rosolia, F. Li, *et al.*, “Safety augmented value estimation from demonstrations (SAVED): safe deep model-based RL for sparse cost robotic tasks,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3612–3619, 2020.

[12] G. Kahn, P. Abbeel, and S. Levine, *Badgr: An autonomous self-supervised learning-based navigation system*, 2020. arXiv: `2002.05700 [cs.RO]`.

[13] S. Gu, E. Holly, T. Lillicrap, and S. Levine, *Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates*, 2016. arXiv: `1610.00633 [cs.RO]`.

[14] A. J. Taylor, V. Dorobantu, H. M. Le, Y. Yue, and A. Ames, “Episodic learning with control lyapunov functions for uncertain robotic systems*,” *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6878–6884, 2019.

[15] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, “Control barrier functions: Theory and applications,” in *2019 18th European Control Conference (ECC)*, 2019, pp. 3420–3431.

[16] A. Aswani, H. Gonzalez, S. Sastry, and C. Tomlin, “Provably safe and robust learning-based model predictive control,” *Automatica*, vol. 49, 2011.

[17] L. Lamport, “Proving the correctness of multiprocess programs,” *IEEE Transactions on Software Engineering SE-3*, vol. 2, pp. 125–143, 1977.

[18] B. Alpern and F. B. Schneider, “Defining liveness,” *Information Processing Letters*, vol. 21, no. 4, pp. 181–185, 1985.

[19] J. García and F. Fernández, “A comprehensive survey on safe reinforcement learning,” *Journal of Machine Learning Research*, vol. 16, no. 42, pp. 1437–1480, 2015.

[20] S. P. Coraluppi and S. I. Marcus, “Risk-sensitive and minimax control of discrete-time, finite-state markov decision processes,” *Automatica*, vol. 35, no. 2, pp. 301–309, 1999.

[21] P. Geibel and F. Wysotzki, “Risk-sensitive reinforcement learning applied to control under constraints,” *Journal of Artificial Intelligence Research*, vol. 24, pp. 81–108, 2005.

[22] T. M. Moldovan and P. Abbeel, *Safe exploration in markov decision processes*, 2012. arXiv: `1205.4810 [cs.LG]`.

[23] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[24] F. Berkenkamp, R. Moriconi, A. P. Schoellig, and A. Krause, “Safe learning of regions of attraction for uncertain, nonlinear systems with gaussian processes,” in *2016 IEEE 55th Conference on Decision and Control (CDC)*, 2016, pp. 4661–4666.

[25] Y. Sui, A. Gotovos, J. Burdick, and A. Krause, “Safe exploration for optimization with gaussian processes,” in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37, PMLR, 2015, pp. 997–1005.

[26] D. Sadigh and A. Kapoor, “Safe control under uncertainty with probabilistic signal temporal logic,” in *Proceedings of Robotics: Science and Systems XII*, 2016.

[27] K. Polymenakos, A. Abate, and S. Roberts, *Safe policy search with gaussian process models*, 2019. arXiv: `1712.05556 [stat.ML]`.

[28] F. M. Berkenkamp, M. Turchetta, A. P. Schoellig, and A. Krause, *Safe model-based reinforcement learning with stability guarantees*, 2017. arXiv: `1705.08551 [stat.ML]`.

[29] T. J. Perkins and A. G. Barto, “Lyapunov design for safe reinforcement learning,” *Journal of Machine Learning Research*, vol. 3, pp. 803–832, 2003.

[30] E. Uchibe and K. Doya, “Constrained reinforcement learning from intrinsic and extrinsic rewards,” in *2007 IEEE 6th International Conference on Development and Learning*, 2007, pp. 163–168.

[31] J. Achiam, D. Held, A. Tamar, and P. Abbeel, *Constrained policy optimization*, 2017. arXiv: `1705.10528 [cs.LG]`.

[32] A. Wachi and Y. Sui, “Safe reinforcement learning in constrained Markov decision processes,” in *Proceedings of the 37th International Conference on Machine Learning*, vol. 119, PMLR, 2020, pp. 9797–9806.

[33] R. Bobiti and M. Lazar, “A sampling approach to finding lyapunov functions for nonlinear discrete-time systems,” in *ECC ’16 : 15th European Control Conference, 29 June - 1 July 2016, Aalborg, Denmark*, Institute of Electrical and Electronics Engineers, 2016, pp. 561–566.

[34] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, “Control barrier functions: Theory and applications,” in *2019 18th European Control Conference (ECC)*, 2019, pp. 3420–3431.

[35] K. P. Wabersich and M. N. Zeilinger, *Safe exploration of nonlinear dynamical systems: A predictive safety filter for reinforcement learning*. 2018. arXiv: `1812.05506 [cs.SY]`.

[36] K. Garg and D. Panagou, *Control-lyapunov and control-barrier functions based quadratic program for spatio-temporal specifications*, 2019. arXiv: `1903.06972 [math.OC]`.

[37] A. Clark, “Control barrier functions for complete and incomplete information stochastic systems,” in *2019 American Control Conference (ACC)*, 2019, pp. 2928–2935.

[38] M. J. Khojasteh, V. Dhiman, M. Franceschetti, and N. Atanasov, “Probabilistic safety constraints for learned high relative degree system dynamics,” in *Learning for Dynamics and Control*, vol. 120, PMLR, 2020, pp. 781–792.

[39] S. Olsen, *A lyapunov-based framework for safe reinforcement learning*, 2020.

[40] J. Choi, F. Castañeda, C. J. Tomlin, and K. Sreenath, *Reinforcement learning for safety-critical control under model uncertainty, using control lyapunov functions and control barrier functions*, 2020. arXiv: `2004.07584 [eess.SY]`.

[41] H. K. Khalil, *Nonlinear systems; 3rd ed.* Prentice-Hall, 2002.

[42] E. Sonntag, “A lyapunov-like stabilization of asymptotic controllability,” *SIAM Journal of Control and Optimization*, vol. 21, no. 3, pp. 462–471, 1983.

[43] S. Prajna, “Barrier certificates for nonlinear model validation,” *Automatica*, vol. 42, pp. 117–126, Jan. 2006.

[44] K. Garg and D. Panagou, “Control-lyapunov and control-barrier functions based quadratic program for spatio-temporal specifications,” in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 1422–1429.

[45] Q. Nguyen and K. Sreenath, “Exponential control barrier functions for enforcing high relative-degree safety-critical constraints,” *2016 American Control Conference (ACC)*, pp. 322–328, 2016.

[46] S. J. Russell and P. Norvig, *Artificial Intelligence: a modern approach*, 3rd ed. Pearson, 2009.

[47] S. M. Ross, *Stochastic processes; 2nd ed.* Wiley, 1996.

[48] G. Alsmeyer, “Chebyshev’s inequality,” in *International Encyclopedia of Statistical Science*, M. Lovric, Ed. Springer Berlin Heidelberg, 2011, pp. 239–240.

[49] D. P. Dubhashi and A. Panconesi, *Concentration of measure for the analysis of randomized algorithms.* Cambridge University Press, 2009.

[50] C. M. Bishop, *Pattern Recognition and Machine Learning.* Springer, 2006.

[51] B. Shahriari, K. Swersky, Z. Wang, R. Adams, and N. D. Freitas, “Taking the human out of the loop: A review of bayesian optimization,” *Proceedings of the IEEE*, vol. 104, pp. 148–175, 2016.

[52] W. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality: Second Edition.* Wiley-Blackwell, 2011.

[53] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[54] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

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

[56] N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger, "Information-theoretic regret bounds for gaussian process optimization in the bandit setting," *IEEE Transactions on Information Theory*, vol. 58, no. 5, pp. 3250–3265, 2012.

[57] S. Shekhar and T. Javidi, "Bayesian function optimization with adaptive discretization," in *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2017, pp. 533–540.

[58] F. M. Berkenkamp, *Safe exploration in reinforcement learning: Theory and applications in robotics*, 2019.

[59] N. Srinivas, A. Krause, S. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, 2010, pp. 1015–1022.

[60] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[61] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, *et al.*, *Openai gym*, 2016. arXiv: `1606.01540 [cs.LG]`.

[62] E. Goan and C. Fookes, "Bayesian neural networks: An introduction and survey," *Lecture Notes in Mathematics*, pp. 45–87, 2020.

[63] G. Notomista, S. Ruf, and M. Egerstedt, "Persistification of robotic tasks using control barrier functions," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, Jan. 2018.

[64] A. Kwok and S. Martinez, "Energy-balancing cooperative strategies for sensor deployment," in *2007 46th IEEE Conference on Decision and Control*, 2007, pp. 6136–6141.