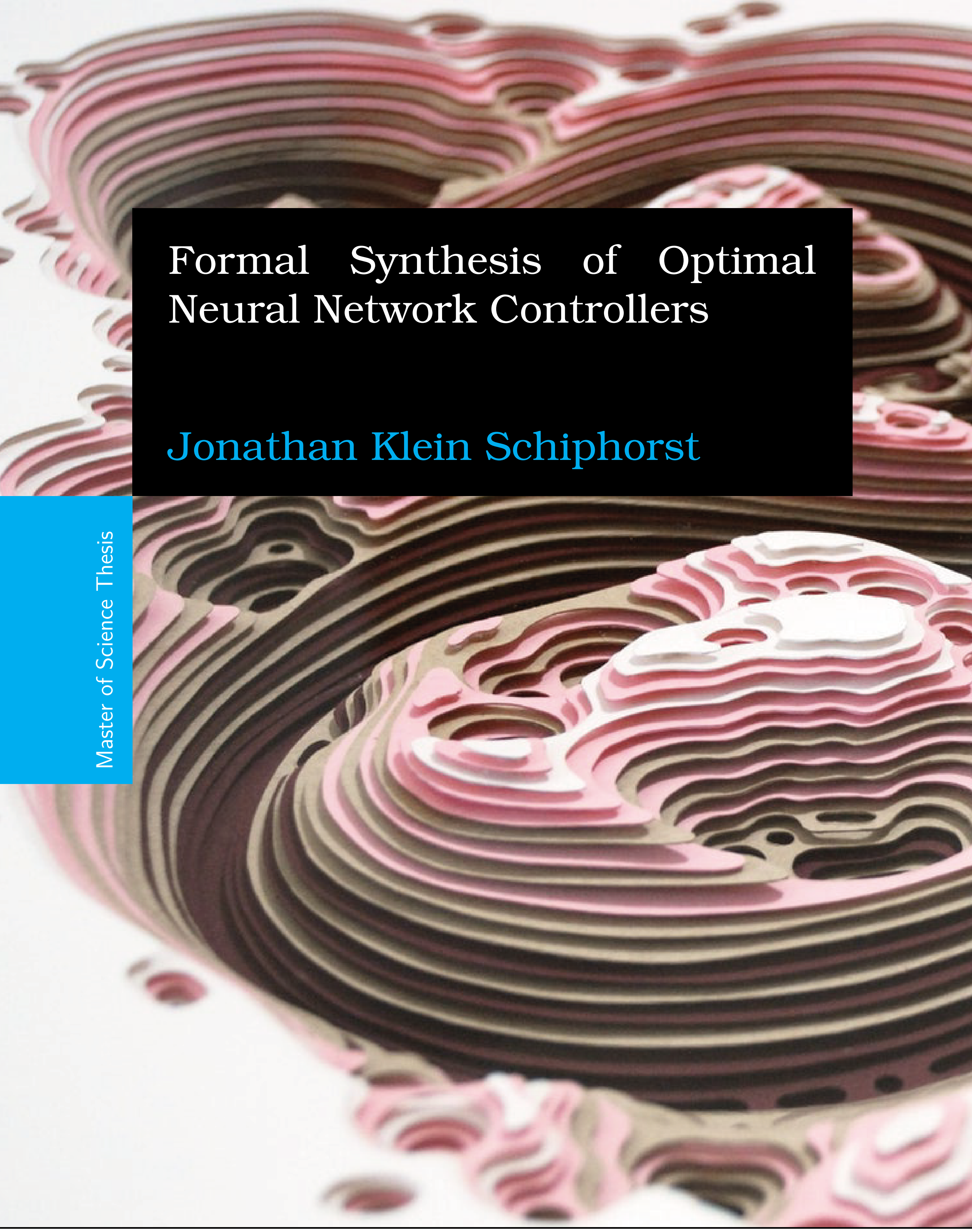
# Formal Synthesis of Optimal Neural Network Controllers

Jonathan Klein Schiphorst

Master of Science Thesis



Delft University of Technology

Delft Center for Systems and Control

# Formal Synthesis of Optimal Neural Network Controllers

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Jonathan Klein Schiphorst

March 26, 2021

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

# Abstract

Stability, safety and optimality are often sought-after properties in the field of controller synthesis. In the last century, linear control theory has matured to a level where scalable algorithms are widely available that are able to synthesize controllers with stability and optimality guarantee. However, the synthesis of safe controllers largely remains an open question. Furthermore, when nonlinear systems are considered, these methods often result in sub-optimal performance, or fail to provide a stabilizing solution at all. Current nonlinear controller synthesis methods typically lack stability, safety and/or optimality guarantee, or require computationally expensive online optimization. This thesis presents a novel method that is able to overcome these limitations.

The presented controller synthesis method combines techniques from Approximate Dynamic Programming (ADP), Lyapunov theory and barrier theory, to synthesize an optimal controller in the form of a Neural Network (NN). Alongside the controller a second NN is deployed, which is trained to serve as a certificate function to provide stability and/or safety guarantee. To assure the correctness of the procedure, Satisfiability Modulo Theory (SMT) and Counterexample-Guided Inductive Synthesis (CEGIS) are utilized.

We subject the method to a number of case studies, through which the versatility of the method is demonstrated. We show the method is able to work with linear and nonlinear systems, as well as systems with input and state constraints. Furthermore, we show the method yields controllers that are able to outperform linear controllers in terms of cost minimization.

# Table of Contents

# List of Figures

# List of Tables

# Acknowledgements

Almost exactly one year ago, I caught a last-minute flight back to the Netherlands. Due to the forthcoming pandemic, I had to quit my internship and leave Zürich after only two of the intended six months. One week later, as the world came to a stop, I was fortunate enough to be able to start writing this thesis.

I would like to thank my supervisor Dr. Ir. Manuel Mazo for setting me up with a thesis project on such short notice, and for his guidance throughout the project. I sincerely enjoyed your laid back, yet punctual style of running your lab.

I would like to thank my daily supervisor Dr. Ir. Cees Verdier for his extensive supervision and his dedication as a supervisor. During our weekly meetings you pointed me in the right direction when needed and your attention to detail has helped me bring this thesis together.

I want to thank Eva, my roommates, my friends from S&C and from Utrecht for making the best of the situation and enjoying the little things in life with me.

Lastly, I want to thank my brother, my sister and my parents for checking in on my every once in a while, when I forget to update them with my current activities and whereabouts, and for reminding me to *make haste slowly.*

Rotterdam,
March 26, 2021

Jonathan Klein Schiphorst

*Festina lente.*

# Chapter 1

# Introduction

## 1-1 Motivation

In the field of controller synthesis, stability and safety are often sought-after control specifications. A typical stability requirement for dynamical systems is asymptotic stability, which implies that the states of the considered system will eventually converge to a goal state or prescribed trajectory. Safety denotes that a certain set of unsafe states will never be visited. Stability and safety are qualitative specifications which the system either satisfies or violates. Formal methods [1] can be deployed to guarantee these qualitative specifications hold. A quantitative control specification is optimality, which requires the controller to minimize a certain cost along the trajectories of the system.

For linear systems, there exist numerous systematic procedures for automatic controller synthesis. In particular, the Linear-Quadratic Regulator (LQR) [2] is a design methodology to synthesize feedback controllers that not only stabilize the system, but are also optimal with respect to a quadratic cost function. For nonlinear systems, however, the controller synthesis methods do not share the same level of maturity as for linear systems. It has been argued that, due to the large diversity of nonlinear phenomena, developing a single design methodology that can handle all nonlinear models would result in overly conservative controllers, or might even be infeasible [3].

However, in recent years different control methods, e.g. Model Predictive Control (MPC) [4], Approximate Dynamic Programming (ADP) [5] and Reinforcement Learning (RL) [6], have proven themselves successful at finding optimal controllers for nonlinear systems. In particular, the combination of RL and Neural Networks (NNs), called deep RL, has shown notable successes in e.g. classical control problems [7], robot locomotion [8] and quadrotor control [9]. The effectiveness of the obtained controllers is mostly tested in simulation or through real world deployment. Providing stability or safety guarantees for controllers obtained by RL is often not considered and remains largely an open question. MPC approaches do address stability and safety guarantees, but require computationally expensive online optimization, which can be prohibitive in practical applications [10].

This leads to the goal of this thesis: to design an automatic controller synthesis procedure for continuous-time nonlinear systems that produces near-optimal controllers with stability and safety guarantees, which does not require expensive online computation. The presented approach is based on ADP, Lyapunov and barrier theory, and utilizes the approximation capabilities of NNs. To assure the correctness of the procedure, Satisfiability Modulo Theory (SMT) and CEGIS are utilized. The method produces optimal NN controllers that verifiably stabilize the system, while avoiding unsafe states.

## 1-2 Related work

The goal of this thesis is to design an automatic controller synthesis procedure for continuous-time nonlinear systems that produces (near-)optimal controllers with stability and safety guarantees, which does not require expensive online computation. First, the nomenclature used in this thesis will be defined. Thereafter, we give a concise overview of the strengths and weaknesses of formal controller synthesis methods. The methods are categorized into three main paradigms: abstraction and simulation, optimization, and certificates. Next, optimal control methods and their efforts to provide stability and safety guarantees are discussed.

### 1-2-1 Nomenclature

This work addresses topics from both control theory and reinforcement learning, which have independently discovered similar concepts. Consequently, both fields have developed different terms for identical subjects. In this section we define the terminology to avoid confusion.

This thesis follows the naming convention of [6], where techniques for approximately solving optimal control problems where the model is known are labeled ADP, whereas model-free approaches are called RL. Solving such problems results in a *control law*, which is a mathematical formula used by the controller to determine the *input*. *Policy* is an alternative to this term, which we will use interchangeably.

The notion of optimality is always defined with respect to a certain cost function, or reward function in RL. *Cost* and *reward* are effectively antonyms, as reward is equal to negative cost. The goal of RL is to maximize the expected cumulative future rewards (also called *value*), while the goal of optimal control is to minimize the expected cumulative future cost, also known as *cost-to-go*. In this work, the terms cost and value will be used.

### 1-2-2 Formal controller synthesis

**Abstraction and simulation** Let us first consider the abstraction and simulation approach, in which infinite systems are abstracted to finite systems, such as a symbolic model [11] [12]. If an abstraction has been found, controller synthesis and verification can be done on the finite system, for which there exist efficient and mathematically sound methods [13]. When abstracting an infinite system into a finite equivalent, the state and action spaces are partitioned. Software tools exist that are able to create abstractions and synthesize correct-by-design controllers for nonlinear or hybrid systems. The controllers are able to enforce specifications described in Linear Temporal Logic (LTL) [14]. Examples of such tools are SCOTS [15], PESSOA [16] and CoSyMa [17].

The strength of this approach is the fact that it can handle a large variety of system types, such as linear, nonlinear and hybrid systems. Also the expressiveness of LTL allows for complex controller design. However, due to the necessity of discretization, these methods suffer from the curse of dimensionality and therefore do not scale well for high-dimensional systems. To improve scalability, one could use multiscale discretization [18] [19], decomposition the main system into subsystems [20] [21], or use a discretization-free approach [22]. Another limiting factor is the size of the resulting controllers, which are stored in look-up tables, such as binary decision diagrams or sparse matrices. The size of these controllers can prohibit the deployment of these controllers in limited memory environments, such as an embedded system. Attempts have been made to reduce the size of the controllers by using piece-wise linear functions [23], and symbolic regression [24].

**Optimization** Signal Temporal Logic (STL) and optimization are central concepts to methods in the second paradigm. STL [25] allows for an assessment of the robustness of satisfaction of a logic formula. The assessment is captured in the magnitude and sign of an STL score. A positive score indicates satisfaction, a negative score indicates violation, and the magnitude indicates to which extent the logic formula is satisfied or violated. Cost functions that capture the intended controller behaviour and the STL score can be combined into an optimization problem, which can be solved by standard optimization solvers [1]. If the solver returns a positive score, the result is a trajectory that satisfies the STL formula. This effectively synthesizes open-loop controllers that are correct, robust, and have a certain level of optimality.

There are several ways to construct an optimization problem for formal controller synthesis. When dealing with discrete-time systems, STL formulas that are linear in state and input can be translated into mixed-integer linear constraints [26], posing the trajectory optimization problem as a mixed-integer linear program [27–31] or mixed-integer quadratic program [32]. A technique to encode STL specifications as mixed-integer constrains is provided in [33]. In [34, 35] the optimal control law was found by calculating the gradient of the STL score and utilizing gradient descent techniques. A third approach is to use SMT solvers for temporal logic control [36]. SMT solvers are tools that are able to check whether first-order logic formulae are satisfied [37]. In [38], SMT solvers and convex optimization techniques are combined in a method called satisfiability modulo convex optimization.

The techniques discussed above are called *trajectory optimization*, which is an open-loop control strategy. Attempts to obtain formal closed-loop controllers include [39] [40], which consider an MPC approach. The main drawback of MPC methods is that they require computationally demanding online optimization [1].

**Certificates** Methods in the certificates paradigm deploy so-called certificate functions, which existence implies certain control specifications for autonomous systems, such as stability and safety. Aleksandr Lyapunov proved for nonlinear systems that the existence of a positive definite function decreasing along the trajectories of the system is a sufficient condition for exponential stability [41]. Such functions became known as Lyapunov functions.

Lyapunov stability does not guarantee that the trajectories of a system will not visit certain (bad) states, before eventually converging to the equilibrium. To address this, set invariance theory was used to provide conditions for safety, i.e. any trajectory starting inside an invariant set will never reach the complement of the set [42]. More recently barrier certificates were introduced as a tool to formally prove safety of nonlinear and hybrid systems [43] [44], which independently rediscovered the conditions from [42]. Analogous to Lyapunov stability, one can prove safety of a system by finding a barrier function which renders a certain safe set invariant.

Synthesizing Lyapunov and barrier functions is not trivial and traditionally requires expert knowledge, hence the need for automated procedures to obtain such functions. Early approaches were focused on Semidefinite Programming (SDP) and Sum-of-Squares (SOS) decomposition [45], which are able to find polynomial Lyapunov functions for polynomial systems. The main drawback of this approach is that it is restricted to fixed-degree polynomial functions. As shown in [46], global asymptotical stability of a polynomial system does not imply the existence of a polynomial Lyapunov function. Furthermore, SDPs are known to have numerical sensitivity issues, which makes it harder to find solutions that fully satisfy the Lyapunov conditions [47]. The SOS method scales to modest dimensions (10-15 states [48]), but attempts are made to make the method scale better by leveraging the sparsity of the underlying SDP [49] or by exploiting the problem properties to eliminate the Lagrange multipliers [50] in the optimization problem.

To go beyond the limitations of SOS methods, a promising approach is Counterexample-Guided Inductive Synthesis (CEGIS) [51] in conjunction with NNs. A candidate certificate in the form of a NN is iteratively refined using counterexamples, i.e. states where the certificate conditions are not satisfied. This framework is used to synthesize Lyapunov functions for nonlinear systems [52] and piecewise linear systems [53], and to synthesize barrier functions [54].

CEGIS is used to synthesize controllers in [55], which considers nonlinear continuous-time systems. A Lyapunov certificate in the form of a NN for is synthesized alongside a linear policy. The certificate network is trained using stochastic gradient decent to iteratively minimize a cost function called *Lyapunov risk*, which measures the degree of violation of the Lyapunov conditions. The Lyapunov conditions are then verified using an SMT solver. The solver will either confirm that the conditions hold, or produces a counterexample, which is added to the training set for the next iteration of learning. The policy gains are adjusted to obtain the largest region of attraction possible. The policies are, however, restricted to be linear, which limits the possibility for complex controller design. The method of [55] is expanded in [56], where neural networks are utilized to represent both Lyapunov and barrier certificates, as well as the policy. In their experiments, however, the policies are restricted to be linear. Neural policies are synthesized in [57] with safety guarantee. Experiments show the ability to synthesize stabilizing controllers as well, but the system was not formally verified to adhere to the Lyapunov conditions. Other formal synthesis approaches utilize the CEGIS method in conjunction with linear matrix inequalities [58] or genetic programming [59].

### 1-2-3 Optimal controller synthesis

The field of optimal control is concerned with synthesizing controllers that minimize a cost function over a (possibly infinite) horizon along the trajectories of a system. Central to optimal control methods are so-called *value* functions, which describe the expected cumulative future cost as a function of the system states. We distinguish two categories of optimal controller synthesis methods. Methods in the first category use the value function implicitly, whereas methods in the second category deploy an explicit value function.

**Implicit value function** The Pontryagin's Maximum Principle (PMP) is a necessary condition for an open-loop control strategy to evolve the states of the system along the optimal trajectory between two states [60]. As this condition is not sufficient, the solutions obtained by PMP methods are candidate optimal solutions [61] and need extra verification related to their existence [62].

An alternative method is MPC, where a sequence of cost minimizing control actions is computed over a finite horizon by solving an optimization problem at each sampling instant. The value function is therefore calculated implicitly. After the first control action of the sequence is deployed, the state is sampled again and the process repeats. State and input constraints can be enforced by utilizing a model of the system to predict the future states over the finite horizon. In order to guarantee infinite horizon stability, a terminal cost term can be added to the optimization problem to ensure the cost of the finite horizon optimal control problem is an upper bound on the infinite horizon cost [10]. The need for computationally expensive online optimization originally made MPC only applicable to systems which allow low sampling rates, e.g. systems in the process industry [63]. Although computation power became more accessible in recent years, proving MPC successful in e.g. robotics [64] and autonomous driving [65], the computational burden of the method is still a major drawback.

**Explicit value function** A sufficient condition for optimality involves an explicit value function in the form of a partial differential equation called the Hamilton-Jacobi-Bellman (HJB) equation [66]. Finding the optimal control law requires the solution of the HJB equation. For linear systems with quadratic cost, the solution of HJB equation reduces to the Continuous Algebraic Riccati Equation (CARE). The CARE can be solved analytically, resulting in the LQR solution [2]. For nonlinear systems, however, solving the HJB equation analytically is generally more difficult. ADP approaches circumvent this difficulty, by utilizing a function approximator to represent the solution of the HJB equation, from which a near-optimal closed-loop controller can be derived. A class of techniques called Generalized Policy Iteration (GPI) [67] is able to jointly improve the controller and value function, until they reach their respective optimal solution. If the exact solution of the optimal value function is known, the resulting policy is inherently stable, as the optimal value function serves as a Lyapunov function if the cost function is undiscounted and positive definite [68]. However, due to approximation, the obtained value function is most likely sub-optimal, possibly breaking the stability guarantee. Therefore, additional verification is needed to verify the Lyapunov conditions.

Optimal controller synthesis methods exist methods for systems with model uncertainties, such as $H_\infty$ [69], or when knowledge of the model is completely absent, such as Q-learning [70], Actor-Critic [71], and Proximal Policy Optimization [72]. In this work we will not further consider these methods, as we assume the system dynamics are fully known and deterministic.

## 1-3 Research goal & approach

**Table 1-1:** Comparison of the controller synthesis methods in literature.

| Method | Advantages | Limitations |
|---|---|---|
| Abstraction-based | – LTL allows for rich controller specifications | – Curse of dimensionality<br>– Look-up table controller |
| Optimization-based | – Inherent optimality guarantee | – Online computational cost |
| Certificate-based | – Suitable for continuous-time systems | – Optimality not considered |
| MPC | – Allows for constraints | – Challenging to provide guarantees<br>– Online computational cost |
| ADP | – Closed-loop controllers<br>– Suitable for continuous-time systems | – Solving HJB analytically often not possible<br>– Approximation breaks stability guarantee |

An overview of the advantages and limitations of the main controller synthesis methods discussed in Section 1-2 is given in Table 1-1. The reviewed methods are limited by least one of the following aspects: the resulting controllers

1. lack stability and/or safety guarantee,
2. are not optimal,
3. require expensive online computation,

The goal of this thesis is to propose a novel method that overcomes these limitations. The research goal is formalized as follows:

**Research goal.** Design an automatic controller synthesis procedure for continuous-time non-linear systems that produces (near-)optimal controllers with stability and safety guarantees that does not require expensive online computation.

Given the research goal, certificate-based methods show a number of distinct advantages compared to other approaches. First, certificate-based methods do not require online optimization, as they allow for offline synthesis of closed-form controllers. Furthermore, certificate methods are capable of dealing with continuous-time systems. However, certificate-based methods do not consider optimality. Conversely, ADP approaches yield optimal controllers, but lack formal guarantee of safety and stability. We therefore propose a method that combines ADP with Lyapunov and barrier theory to synthesize controllers that are safe, stable and optimal. A NN is utilized to approximate the value function, which concurrently serves as the Lyapunov and barrier function. To assure the correctness of the procedure, SMT and CEGIS are utilized.

## 1-4 Contributions beyond the State of the Art

The contributions of this thesis are best described in comparison to other formal neural controller synthesis methods for continuous-time systems that utilize NNs in conjunction with

**Table 1-2:** An overview of formal neural certificate synthesis approaches

| Category | Paper | Systems | Certificate | Verification | Policy | Optimality |
|---|---|---|---|---|---|---|
| Verification | [52] | Nonlinear | Lyapunov | SMT | × | × |
| | [54] | Hybrid | Barrier | SMT | × | × |
| | [53] | Piecewise linear | Lyapunov | MILP | × | × |
| Synthesis | [55] | Nonlinear | Lyapunov | SMT | linear | × |
| | [56] | Nonlinear | Lyapunov & barrier | Lipschitz | nonlinear | × |
| | [57] | Nonlinear | Barrier | SMT | nonlinear | × |

SMT solvers. We divide the approaches in literature into two categories: *verification* and *synthesis*. Methods in the verification category [52–54] solely focus on the certification of safety and stability for systems with a given controller, by training a Lyapunov or barrier function, respectively. The methods utilize SMT or Mixed-Integer Linear Programming (MILP) for verification.

In contrast, methods in the synthesis category do allow for controller synthesis. [55] synthesizes Lyapunov certificates for nonlinear continuous-time systems and a linear policy which obtains the largest region of attraction possible. The policies are, however, restricted to be linear, which limits the possibility for complex controller design. The method of [55] is expanded in [56], where neural networks are utilized to represent both Lyapunov and barrier certificates, as well as the policy. In their experiments, however, the policies are restricted to be linear. Neural policies are synthesized in [57] with safety guarantee. Experiments show the ability to synthesize stabilizing controllers as well, but the system was not formally verified to adhere to the Lyapunov conditions. Furthermore, optimality is not considered during controller synthesis. The above described methods are summarized in Table 1-2.

Our method is able to synthesize optimal neural controllers for continuous-time nonlinear systems, with formal stability guarantee. We take an ADP approach to synthesize a value function in the form of a NN, which serves a Lyapunov function for stability certification, verified by an SMT solver. By imposing an extra condition on the Lie derivative of the network, the value network simultaneously functions as approximate HJB equation, which is subsequently used to improve the policy in a process called Formal Neural Policy Iteration (FNPI).

## 1-5 Thesis outline

The outline of the remainder of this thesis is as follows. The preliminary theory used in this thesis is given in Chapter 2. Conditions for stability, safety and optimality, as well as an introduction to and formal verification of NNs are discussed. This chapter concludes with the problem statement. In Chapter 3, the developed formal controller synthesis method is presented. The method is subjected to a number of case studies, which are evaluated and discussed in Chapter 4. This thesis is concluded with an overall conclusion and suggestions for future work in Chapter 5.

Chapter 2

# Preliminaries and Problem Statement

In this chapter the preliminary theory used in this thesis is covered. Stability, safety and optimality certification is adressed in Section 2-1, which involves finding a certificate function satisfying specific conditions. In this work, the certificate functions will be represented by Neural Networks (NNs). Therefore, we cover the fundamentals of NNs and methods to verify relevant properties of NNs in Section 2-2. This chapter is concluded with the problem statement in Section 2-3.

## 2-1 Conditions for stability, safety and optimality

We address the synthesis of optimal controllers for $n$-dimensional continuous-time dynamical systems, described by

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)), \quad \boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{2-1}$$

where $\boldsymbol{x}(t) \in X \subseteq \mathbb{R}^n$ denotes the state vector, $\boldsymbol{u}(t) \in U \subseteq \mathbb{R}^m$ denotes the input vector, and $f : X \times U \to \mathbb{R}^n$ is a Lipschitz-continuous vector field. $X$ and $U$ define the state space and input space of the system, respectively. The input is determined by the policy

$$\boldsymbol{u}(t) = \pi(\boldsymbol{x}(t)), \tag{2-2}$$

which is a mapping from state to input, $\pi : X \to U \subseteq \mathbb{R}^m$. Combining Equations (2-1) and (2-2) results in a closed-loop system, given by $\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \pi(\boldsymbol{x}(t)))$.

The certification of stability, safety and optimality of a dynamical system involves finding a certificate function satisfying certain conditions on both the function itself and its time derivative, which is equal to the Lie derivative with respect to vector field $f$.

**Definition 1** (Lie derivative)**.** The Lie derivative of a continuously differentiable scalar function $G : X \to \mathbb{R}$, with respect to an $n$-dimensional vector field $f$, is defined as

$$\dot{G}(\boldsymbol{x}) = \nabla G(\boldsymbol{x}) \cdot f(\boldsymbol{x}) = \sum_{i=1}^{n} \frac{\partial G}{\partial x_i} \frac{dx_i}{dt}$$

First, let us consider stability. A system is stable if the trajectories of the system stay bounded. Asymptotic stability guarantees all solutions that start out near the equilibrium point will eventually converge to the equilibrium.

**Definition 2** (Asymptotic stability [73])**.** Consider the system in (2-1) defined over domain $D \in \mathbb{R}^n$, which contains the origin, and $f(0) = 0$. The equilibrium point $x = 0$ of (2-1) is stable if for each $\epsilon > 0$ there exists a $\delta(\epsilon) > 0$, such that

$$\|\boldsymbol{x}(t_0)\| < \delta \implies \|\boldsymbol{x}(t)\| < \epsilon, \quad \forall t \geq t_0.$$

The system is asymptotically stable if the equilibrium is stable and, for some $\delta > 0$,

$$\|\boldsymbol{x}(t_0)\| < \delta \implies \lim_{t \to \infty} \|\boldsymbol{x}(t)\| = 0.$$

Aleksandr Lyapunov proved for nonlinear systems that the existence of a positive definite function decreasing along the solution trajectories is a sufficient condition for asymptotic stability [41].

**Theorem 1** (Lyapunov function for stability certification [73])**.** Consider the system in (2-1) and domain of interest $D \subseteq \mathbb{R}^n$, which contains the origin, and $f(0) = 0$. If there exists a continuously differentiable function $V : X \to \mathbb{R}$ that satisfies

$$V(0) = 0, \quad V(\boldsymbol{x}) > 0 \;\; \forall x \in \{D \mid x \neq 0\}, \quad \dot{V}(\boldsymbol{x}) \leq 0 \;\; \forall x \in D, \tag{2-3}$$

then the origin is stable. Moreover, if

$$V(0) = 0, \quad V(\boldsymbol{x}) > 0 \;\; \forall x \in \{D \mid x \neq 0\}, \quad \dot{V}(\boldsymbol{x}) < 0 \;\; \forall x \in D \backslash \{0\}, \tag{2-4}$$

then $V$ is a Lyapunov function and the system is asymptotically stable at the origin.

Intuitively, a Lyapunov function maps the system states $x$ into energy-like values which, by the conditions of (2-4), decrease over time along the system trajectories and are bounded from below. Lyapunov stability does not guarantee that the trajectories of a system will not visit certain (bad) states. A solution of an asymptotically stable system may arbitrarily diverge from the equilibrium point before eventually converging to the equilibrium. To address this, set invariance theory was used to provide conditions that prove safety, i.e. any trajectory starting inside an invariant set will never reach the complement of the set [42].

**Theorem 2** (Barrier function for safety certification [43])**.** Consider the system in (2-1), initial set $X_0$ and unsafe set $X_u$. If there exists a continuously differentiable function $B : X \to \mathbb{R}$, that satisfies

$$B(x) \leq 0 \;\; \forall x \in X_0, \quad B(x) > 0 \;\; \forall x \in X_u, \quad \dot{B}(x) \leq 0 \;\; \forall x \in \{x \in D \mid B(x) = 0\}, \tag{2-5}$$

then $B$ is a Barrier function and the system is safe, i.e. all trajectories starting in $X_0$ will never reach $X_u$.

Stability and safety are qualitative specifications which the system either satisfies or violates. A quantitative specification for dynamical systems is optimality, which requires the control law to minimize the total cost over a certain period of time, defined by $\int_0^T e^{-\rho t}\ell(x,\pi(x))dt$. In order to assess whether a policy is optimal, a so-called value function

$$J^{\pi}(x) = \int_0^T e^{-\rho t}\ell(x,\pi(x))dt, \tag{2-6}$$

is introduced, which defines for each state the discounted additive cost over horizon $T$. In this work, we consider the undiscounted ($\rho = 0$) infinite horizon ($T \to \infty$) case. The value function corresponding to the optimal policy $\pi^*(x)$, is a solution to the Hamilton-Jacobi-Bellman (HJB) equation, which is a sufficient condition for optimality.

**Theorem 3** (Value function for optimality certification [66])**.** Consider the system in (2-1) and the infinite-horizon additive cost, defined by value function $J(\boldsymbol{x}(t)) = \int_0^\infty \ell(\boldsymbol{x}(t),\pi(\boldsymbol{x}(t)))dt$. Suppose $J(\boldsymbol{x}(t))$ is a solution to the HJB equation, i.e. $J$ is continuously differentiable in $\boldsymbol{x}(t)$ and the following holds

$$0 = \min_{\boldsymbol{u}\in U}\left[\ell(\boldsymbol{x}(t),\boldsymbol{u}(t)) + \dot{J}(\boldsymbol{x}(t),\boldsymbol{u}(t))\right],\ \forall \boldsymbol{x} \in X. \tag{2-7}$$

Suppose also that $\pi^*(\boldsymbol{x})$ attains the minimum in equation (2-7) for all $\boldsymbol{x}$. Further assume that the control input trajectory, $\boldsymbol{u}^*(t) = \pi^*(\boldsymbol{x}(t))$ along any solution is piecewise continuous as a function of $t$, and that there exists at least one equilibrium, $\boldsymbol{x}_e$, with $f(\boldsymbol{x}_e,\pi^*(\boldsymbol{x}_e)) = 0$ and $J(\boldsymbol{x}_e) = 0$. Then $J$ is equal to the optimal value function, $J(\boldsymbol{x}) = J^*(\boldsymbol{x})$ for all $\boldsymbol{x}$, and the control trajectories $\boldsymbol{u}^*(t)$ are optimal.

The conditions of Theorem 2-4, 2-5 and 2-7 give sufficient conditions for stability, safety and optimality, respectively. These specifications can be realized by synthesizing multiple certificate functions, or by combining specifications into one certificate function. In this work, we synthesize the value function $J(\boldsymbol{x}(t))$ to serve as a candidate Lyapunov function. To this end, we pose the following conditions on the cost function and on the policy:

$$\pi(0) = 0, \tag{2-8a}$$
$$\ell(x,\pi(x)) > 0, \quad \forall x \neq 0, \tag{2-8b}$$
$$\ell(0,\pi(0)) = 0. \tag{2-8c}$$

If these conditions are met, $J(\boldsymbol{x}(t))$ complies with the conditions from 2-4 and therefore guarantees stability.

Combining stability and safety into one certificate function is less trivial, as the conditions in Equation (2-3) and (2-5) are conflicting. In order to achieve safety, one has to synthesize a barrier function which is negative for all initial states and positive for all unsafe states, whereas for safety the Lyapunov function is strictly positive everywhere except at the equilibrium point. Stability and safety certificates are successfully combined in [74] by taking the weighted sum of two independently designed certificate functions, and [75] provides conditions for a single function to serve as a Lyapunov and barrier function.

## 2-2 Neural networks

Neural Networks (NNs) are widely used in numerous fields of machine learning, popular due to their expressive power and flexibility. NNs with at least one hidden layer and non-polynomial activation function are universal function approximators, which can, given enough parameters, approximate any function and its derivatives to arbitrary precision [76, 77]. Because of this property, NNs are suitable to serve as certificate functions. In this section we cover the fundamentals of NNs and how they are trained. Subsequently, we will discuss how to verify certain properties of NNs to be able to formally serve as certificate functions.

### 2-2-1 Fundamentals

Consider an $n$-layer NN that represents a function $\mathbf{y} = \mathbf{f}(\mathbf{x})$ with input $\mathbf{x} \in \mathcal{D}_\mathbf{x} \subseteq \mathbb{R}^{k_0}$ and output $\mathbf{y} \in \mathcal{D}_\mathbf{y} \subseteq \mathbb{R}^{k_n}$, with $k_0$ and $k_n$ the input and output dimension, respectively. Each layer $i$ in $\mathbf{f}$ is a function $\mathbf{f}_i : \mathbb{R}^{k_{i-1}} \to \mathbb{R}^{k_i}$ which maps its input $z_{i-1}$ of dimension $k_{i-1}$ to the output $z_i$ with dimension $k_i$:

$$\mathbf{z}_i = \mathbf{f}_i\left(\mathbf{z}_{i-1}\right) = \boldsymbol{\sigma}_i\left(\mathbf{W}_i \mathbf{z}_{i-1} + \mathbf{b}_i\right), \tag{2-9}$$

where $\mathbf{z}_{i-1}$ first undergoes a linear transformation $\hat{\mathbf{z}}_i := \mathbf{W}_i\mathbf{z}_{i-1} + \mathbf{b}_i$, parameterized by weight matrix $\mathbf{W}_i$ and bias vector $\mathbf{b}_i \in \mathbb{R}^{k_i}$. Note that $\mathbf{z}_0 = \mathbf{x}$ and $\mathbf{z}_n = \mathbf{y}$. Subsequently, $\hat{\mathbf{z}}_i$ undergoes an element-wise activation function $\boldsymbol{\sigma}_i : \mathbb{R}^{k_i} \to \mathbb{R}^{k_i}$. When stacking the layers together, one ends up with the full representation of the NN function:

$$\mathbf{f} = \mathbf{f}_n \circ \mathbf{f}_{n-1} \circ \cdots \circ \mathbf{f}_1, \tag{2-10}$$

where $\circ$ denotes function composition. Examples of popular activation functions are the `Rectified Linear Unit (ReLU)`, `sigmoid` and `tanh`. However, to be able to use a NN as certificate function, it should be continuously differentiable, which the `ReLU` activation function is not. We therefore consider the `Sigmoid Linear Unit (SiLU)` activation function [78], which is a smooth approximation of the `ReLU` activation function, defined by $\texttt{SiLU}(x) = \dfrac{x}{1+e^{-x}}$. The output of the `ReLU` and `SiLU` activation functions and their derivative are shown in Figure 2-1.

### 2-2-2 Training

The parameters of the weight matrices and bias vectors of each layer are lumped into the parameter vector $\boldsymbol{\theta}$. To obtain the desired network parameters, the network is subjected to a learning process called training. A popular training method is back-propagation [79] in conjunction with a gradient based optimization method [80]. During this process, a performance criterion of the networks outputs, captured in a so-called loss function $L(\mathbf{f}(\mathbf{x}))$, is to be minimized. This is done by calculating the partial derivatives of the loss function with respect to the network parameters, i.e. $\nabla_\theta L(\mathbf{f}(\mathbf{x}))$. The partial derivatives are efficiently calculated by a technique called automatic differentiation [81], in which the partial derivative is propagated backwards through the network.

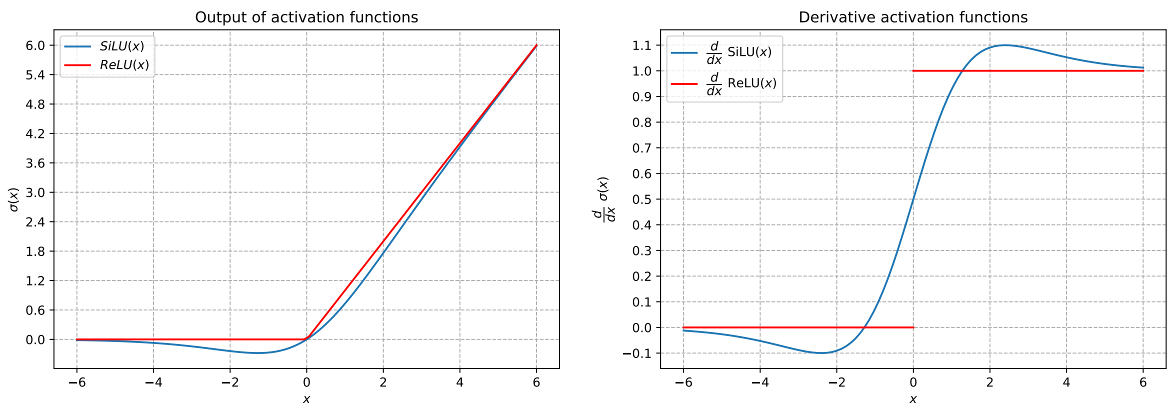


**Figure 2-1:** Left: output of the ReLU and SiLU activation function, Right: derivative of the ReLU and SiLU activation function

Subsequently, the network parameters are updated in the direction of the negative gradient. A widely used optimization step is gradient descent

$$\theta_{k+1} = \theta_k - \lambda \nabla_\theta L(\mathbf{f}(\mathbf{x})),$$

where $\lambda$ denotes the learning rate. Other update rules can be used, such as Adam [82] or AdaGrad [83], which can result in faster convergence.

### 2-2-3 Neural network verification

Because of the *black-box* nature of NNs, it is hard to guarantee certain input-output properties. NNs trained on large sets of data may still output incorrect results in unseen environments and are prone to adversarial attacks [84]. To use a NN as a certificate function, one needs a verification method, that is able to verify whether certain input-output relationships of the network hold, i.e. the following assertion:

$$\mathbf{x} \in \mathcal{X} \Rightarrow \mathbf{y} = \mathbf{f}(\mathbf{x}) \in \mathcal{Y}, \tag{2-11}$$

where $\mathcal{X} \subseteq \mathcal{D}_\mathbf{x}$ is a set of inputs and $\mathcal{Y} \subseteq \mathcal{D}_\mathbf{y}$ is a set of outputs.

In the context of formal controller synthesis, important properties of the verification techniques are *soundness* and *completeness* [85]. In order for a technique to be sound it has to be correct when claiming the assertion holds. Completeness requires that the solver never returns *unknown* and it has to be correct when declaring the assertion is violated.

When using a NN as a certificate function, one has to check whether the certificate conditions (e.g. of Equation (2-3), (2-5) or (2-7)) hold. These conditions can be captured in an assertion like the one in Equation (2-11). Techniques that achieve this are categorized into three groups: primal optimization, dual optimization and symbolic analysis.

In primal optimization, the structure of the network is encoded as constraints to an optimization problem. Due to the piecewise linearity of ReLU activation functions, a network with ReLU activation functions can be encoded as a set of mixed integer linear constraints, which turns the optimization problem into a Mixed-Integer Linear Programming (MILP) problem

[86]. Methods that use this approach are NSVerify [87] and MIPVerify [88], which are both complete. Both methods produce counterexamples for which the assertion is violated. In addition, MIPVerify also returns the maximum allowable disturbance for states to stay in the allowable output set $\mathcal{Y}$.

The methods ILP [89] and Reluplex [90] transform networks with ReLU activation functions into a set of linear constraints, and use Linear Programming (LP) techniques, such as the simplex method, to find the maximum allowable disturbance. In addition Reluplex performs tree search in the function space, which makes it a complete approach. In the ILP method, the network is linearized at a reference point, which makes this method incomplete, as it only considers one linear segment of the network. Sherlock [91] combines a series of MILP feasibility problems alternating with local search steps. Sherlock is not a complete method.

Instead of encoding the structure of the network as constraints, one could also consider dual optimization. The objective in dual optimization correspond to the constraints in primal optimization, which makes the objective more complicated, but the overall problem might be easier to solve. Constructing the dual problem from the primal problem often involves relaxation methods such as Lagrangian relaxation, which makes these approaches incomplete [85]. Efforts in this category include Duality [92], ConvDual [93] and Certify [94]. Duality can handle any type of activation function and Certify allows activation functions that are differentiable everywhere except for countably many points. ConvDual only handles the ReLU activation function.

To verify the Lyapunov conditions of Equation (2-3), one could also unroll the NN into a symbolic expression, as is done in [55]. Subsequently, Satisfiability Modulo Theory (SMT) solvers can be used to verify that the assertions hold or return counterexamples, which could be used in a Counterexample-Guided Inductive Synthesis (CEGIS) scheme. For an overview of the theory and recent advancements of SMT solvers, one can refer to [37]. Popular SMT solvers are Z3 [95], which handles NNs with polynomial activation functions, and dReal [96], which is able to reason over transcendental functions.

Instead of verifying the certificate conditions throughout a continuous subset, one could also verify a finite set of points and exploit the Lipschitz constant of the certificate function, to extend the verification to an infinite set [97].

The different verification methods discussed in this section, the type of activation functions the methods allow, the verification approach and whether the method is complete are summarized in Table 2-1.

**Table 2-1:** An overview of verification methods for neural networks

| Category | Method name | Activation | Approach | Completeness |
|---|---|---|---|---|
| Primal optimization | NSVerify [87] | ReLU | Naive MILP | Yes |
| | MIPVerify [88] | ReLU and Max | MILP with bounds | Yes |
| | ILP [89] | ReLU | Iterative LP | No |
| | Reluplex [90] | ReLU | Simplex | Yes |
| | Sherlock [91] | ReLU | Local and global search | No |
| Dual optimization | Duality [92] | ReLU | Lagrangian relaxation | No |
| | ConvDual [93] | ReLU | Convex relaxation | No |
| | Certify [94] | Differentiable | Semidefinite relaxation | No |
| Symbolic analysis | SMT-based [55] | Differentiable | NN as symbolic expression | No |
| | Sampling-based [97] | Lipschitz continuous | Lipschitz method | No |

## 2-3 Problem statement

In this section we define the framework for the to be designed controller synthesis procedure, based on the reviewed preliminaries of Section 2-1. The research goal states that the synthesis procedure should produce controllers that are (near-)optimal and have stability and safety guarantee. We split the research goal into two sub-goals:

**Problem 1** (Optimal stability)**.** Provided with a cost function, synthesize a controller with formal guarantee of stability, that minimizes the the infinite horizon additive cost.

**Problem 2** (Safe and optimal stability)**.** Provided with a cost function, synthesize a controller with formal guarantee of stability and safety, that minimizes the the infinite horizon additive cost.

To compare the performance of our controllers with controllers from related literature, we introduce a third problem:

**Problem 3** (Cost minimization)**.** Provided with a cost function, the synthesized controller should attain a lower infinite horizon additive cost than a linear controller, where the linear controller is an Linear-Quadratic Regulator (LQR) controller for the linearized system.

To meet these requirements, we utilize certificate functions to guarantee stability and safety. We develop a synthesis procedure based on Approximate Dynamic Programming (ADP), in which two NNs are trained to serve as policy and as certificate function. By complying with the conditions from Equations (2-8), the NN serves as a value function, as well as a Lyapunov function. To assure the correctness of the procedure, the conditions from Theorem 1 will be checked by one of the techniques from Section 2-2-3.

# Chapter 3

# Methodology

In this chapter, we detail the methodology used to tackle Problem 1, 2 and 3. To this end, we present a framework based on deep generalized policy iteration [67], called Formal Neural Policy Iteration (FNPI). The method yields two Neural Networks (NNs), a *policy network*, which stabilizes the system and minimizes cost, and a *value network*, which serves as a Lyapunov function that provides stability guarantee.

The procedure is provided with a dynamical system, a domain of interest, a cost function, network topologies, an initial policy and a stopping condition. Subsequently the synthesis procedure will occur, which consists of four routines: initialization, neural policy evaluation, neural policy improvement, and system verification.

First, during initialization, the policy network is trained to approximate the initial policy. Thereafter, the procedure alternates between neural policy evaluation, during which the value network is trained to represent the value function for the current policy, and neural policy improvement, in which the policy network is optimized with respect to the value function. Before the procedure is allowed to switch between these routines, a system verification step will occur, which guarantees stability of the controller throughout the procedure. The procedure terminates when a stopping condition has been met. A visual representation of how these routines interact is given in Figure 3-1.

This chapter is further organized as follows. First, we describe the four routines in detail, starting with initialization in Section 3-1-1. Next, neural policy evaluation is described in Section 3-1-2, neural policy improvement is described in Section 3-1-3 and system verification is covered in Section 3-1-4. Potential stopping conditions are discussed in Section 3-1-5. We have implemented the procedure described in this chapter in a prototype tool called FNPI, which is discussed in Section 3-2.

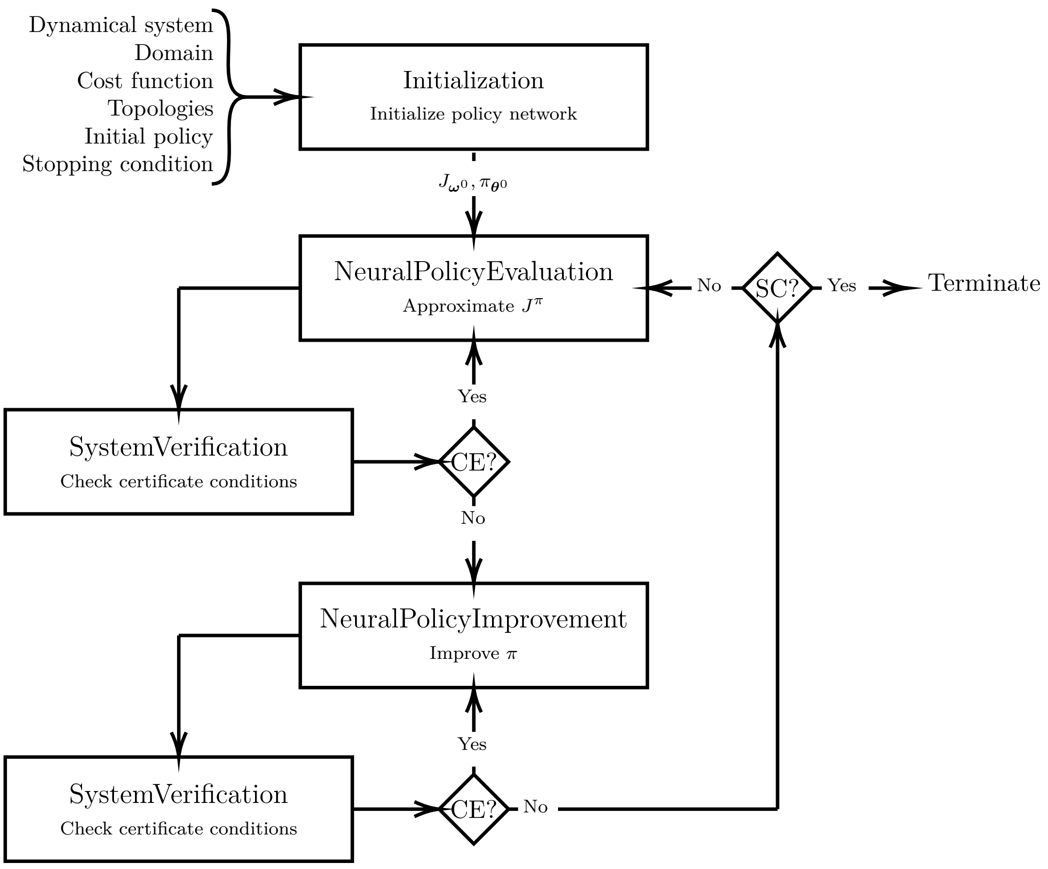


**Figure 3-1:** Block diagram of Formal Neural Policy Iteration. CE denotes counter-example. SC denotes stopping condition.

## 3-1 Formal Neural Policy Iteration

### 3-1-1 Initialization

Upon initialization, the following entities are to be provided to the procedure:

1. A dynamical system.
2. A domain of interest.
3. A cost function.
4. The topologies of the policy and value networks.
5. An initial admissible policy.
6. A stopping condition.

Potential stopping conditions will be discussed in Section 3-1-5. The other entities and the conditions they have to abide by will be discussed hereafter.

Let us consider nonlinear continuous-time dynamical systems of the form

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)), \quad \boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{3-1}$$

where $\boldsymbol{x}(t) \in X \subseteq \mathbb{R}^n$ denotes the state vector, $\boldsymbol{u}(t) \in U \subseteq \mathbb{R}^m$ denotes the input vector, and $f : X \times U \to \mathbb{R}^n$ is a Lipschitz-continuous vector field. $X$ and $U$ define the state space and input space of the system, respectively. Domain of interest $D \subset \mathbb{R}^n$ is a compact set that contains the origin of the system. We assume the plant is stabilizable on $D$ at the origin. That is, there exists an admissible policy $\pi(\boldsymbol{x}(t))$, where $\boldsymbol{u}(t) = \pi(\boldsymbol{x}(t))$. such that the system is asymptotically stable on $D$ [67].

**Definition 3** (Admissible policy [98])**.** A control policy $\pi(\boldsymbol{x}(t)) = \boldsymbol{u}(t)$ is defined as admissible with respect to (3-2) on $D$, denoted by $\pi(\boldsymbol{x}(t)) \in \Psi(D)$, if $\pi(\boldsymbol{x}(t))$ is continuous on $D$, $\pi(0) = 0$, $\pi(\boldsymbol{x}(t))$ stabilizes (3-1) on $D$, and $\forall \boldsymbol{x}_0 \in D$, $J(\boldsymbol{x}_0)$ is finite.

Furthermore, consider cost function $\ell(\boldsymbol{x}(t), \boldsymbol{u}(t))$. In order to make the value network serve as a Lyapunov function, the cost function should have the following properties:

$$\ell(\boldsymbol{x}(t), \pi(\boldsymbol{x}(t))) > 0, \quad \forall x \neq 0, \quad \ell(0, \pi(0)) = 0.$$

Lastly, a network topology comprises the amount of layers, the amount of nodes per layer, the activation function per layer and the learning rate $\alpha$ of the network. The output of the value network $J_{\boldsymbol{\omega}}$ and policy network $\pi_{\boldsymbol{\theta}}$ are defined by parameter vectors $\boldsymbol{\omega}$ and $\boldsymbol{\theta}$, respectively. Once these entities are provided to the procedure, the initial admissible policy will be approximated by the policy network, which we denote by $\pi_{\boldsymbol{\theta}^0}$. In our work, we use the Linear-Quadratic Regulator (LQR) of the linearized system around the origin, unless stated otherwise. Afterwards, the method transitions to the neural policy evaluation routine.

## 3-1-2 Neural Policy Evaluation

In the neural policy evaluation step, the value network is trained to approximate the value function with respect to the current policy and cost function, i.e

$$J_{\boldsymbol{\omega}}(\boldsymbol{x}(t)) \approx \int_t^{\infty} \ell(\boldsymbol{x}(t), \pi_{\boldsymbol{\theta}}(\boldsymbol{x}(t)))dt. \tag{3-2}$$

In order to make this improper integral converge, the evaluated policy needs to be admissible.

The differential equivalent of Equation (3-2) is the associated Hamiltonian of the system [66, 67, 98]

$$H\left(\boldsymbol{x}(t), \pi_{\boldsymbol{\theta}}(\boldsymbol{x}(t)), \frac{\partial J_{\boldsymbol{\omega}}(\boldsymbol{x}(t))}{\partial \boldsymbol{x}}\right) = \ell(\boldsymbol{x}(t), \pi_{\boldsymbol{\theta}}(\boldsymbol{x}(t))) + \frac{\partial J_{\boldsymbol{\omega}}(\boldsymbol{x}(t))}{\partial \boldsymbol{x}} f(\boldsymbol{x}(t), \pi_{\boldsymbol{\theta}}(\boldsymbol{x}(t))), \tag{3-3}$$

for which the following holds if equipped with an admissible policy

$$H\left(\boldsymbol{x}(t), \pi_{\boldsymbol{\theta}}(\boldsymbol{x}(t)), \frac{\partial J_{\boldsymbol{\omega}}(\boldsymbol{x}(t))}{\partial \boldsymbol{x}}\right) = 0, \quad J_{\boldsymbol{\omega}}(0) = 0. \tag{3-4}$$

This provides us with a method to train a the value network $J_{\boldsymbol{\omega}}$ to represent the value function.

Specifically, we define *value loss* as the mean squared error between the Lie derivative of the value network over vector field $f$ and the cost function [67]:

$$L_v = \frac{1}{N} \sum_{x_i \in \mathcal{X}} \left( \ell(x_i, \pi_{\boldsymbol{\theta}}(x_i)) + \frac{\partial J_{\boldsymbol{\omega}}(x_i)}{\partial \boldsymbol{x}} f(x_i, \pi_{\boldsymbol{\theta}}(x_i)) \right)^2, \tag{3-5}$$

where $\mathcal{X}$ denotes a set of samples of the state vector in $D$ and $N = |\mathcal{X}|$. To be able to use the value network as a Lyapunov function, it is necessary that the Lie derivative over vector field $f$ is strictly smaller than zero everywhere except at the equilibrium point. The value loss defined in Equation (3-5) does not explicitly account for this, therefore we define an additional loss term, called *certificate loss*

$$L_c = \frac{1}{N} \sum_{x_i \in \mathcal{X}} \max \left( 0, \frac{\partial J_{\boldsymbol{\omega}}(x_i)}{\partial \boldsymbol{x}} f(x_i, \pi_{\boldsymbol{\theta}}(x_i)) \right), \tag{3-6}$$

which penalises samples that violate the third condition of Equation (2-4). The NN parameters are updated to minimize the weighted sum of the two loss terms described above:

$$L_{Tot} = \beta_v L_v + \beta_c L_c, \tag{3-7}$$

where $\beta_v$ and $\beta_c$ are scaling terms to put emphasis on the loss terms. How these scaling terms are determined is further explained in Section 3-1-4.

Pseudocode for the neural policy evaluation routine is given in Algorithm 1. To ensure compliance with the first condition of Equation (2-4), i.e. $J_{\boldsymbol{\omega}}(0) = 0$, we translate the value network at each iteration, by removing the offset from the bias term of the last layer of the network. This is denoted by the `MoveOriginToZero()` function in Algorithm 1. The routine runs until a certain `StoppingCondition` has been satisfied. This could be when a maximum number of iterations has occurred, or when $L_{Tot}$ falls below a certain threshold.

**Algorithm 1** Neural policy evaluation

```
1:  procedure NeuralPolicyEvaluation($J_{\boldsymbol{\omega}}, \alpha, \beta, \pi, \ell, \mathcal{X}$)
2:      $N \leftarrow |\mathcal{X}|$
3:      $k \leftarrow 0$
4:      while not StoppingCondition do                ▷ e.g. max number of iterations
5:          $L_{v^k} \leftarrow \frac{1}{N} \sum_{i=1}^{N} (\ell(x_i, \pi(x_i)) + \frac{\partial J_{\boldsymbol{\omega}^k}(x_i)}{\partial \boldsymbol{x}} f(x_i, \pi(x_i)))^2$
6:          $L_{c^k} \leftarrow \frac{1}{N} \sum_{i=1}^{N} (\max(0, \frac{\partial J_{\boldsymbol{\omega}^k}(x_i)}{\partial \boldsymbol{x}} f(x_i, \pi(x_i)))$
7:          $L_{Tot^k} \leftarrow \beta_v L_{v^k} + \beta_c L_{c^k}$
8:          $\boldsymbol{\omega}^{k+1} \leftarrow \boldsymbol{\omega}^k - \alpha \nabla_{\boldsymbol{\omega}^k} L_{Tot^k}$      ▷ Update weights with preferred optimizer
9:          MoveOriginToZero()
10:         $k \leftarrow k + 1$
11:     end while
12: end procedure
```

### 3-1-3 Neural Policy Improvement

The next step in the process is to improve the policy with respect to the current value function in a routine called policy improvement. This is done by making the policy *greedy*, i.e. making the policy minimize the current Hamiltonian

$$\pi_{\boldsymbol{\theta}}(\boldsymbol{x}(t)) = \arg\min_{u} \left( \ell(\boldsymbol{x}(t), \boldsymbol{u}(t)) + \frac{\partial J_{\boldsymbol{\omega}}(\boldsymbol{x}(t))}{\partial \boldsymbol{x}} f(\boldsymbol{x}(t), \boldsymbol{u}(t)) \right). \tag{3-8}$$

For input affine control systems of the form $\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t)) + g(\boldsymbol{x}(t))\boldsymbol{u}(t)$ subject to a cost function that is quadratic in the input, i.e. $\ell(\boldsymbol{x}(t), \boldsymbol{u}(t)) = \ell_1(\boldsymbol{x}(t)) + \boldsymbol{u}(t)^\top R \boldsymbol{u}(t)$, with $R$ symmetric and positive definite, the Hamilton-Jacobi-Bellman (HJB) equation takes the form

$$0 = \min_{\boldsymbol{u} \in U} \left[ \ell_1(\boldsymbol{x}(t)) + \boldsymbol{u}(t)^\top R \boldsymbol{u}(t) + \frac{\partial J(\boldsymbol{x}(t))}{\partial \boldsymbol{x}} \left[ f(\boldsymbol{x}(t)) + g(\boldsymbol{x}(t))\boldsymbol{u}(t) \right] \right], \tag{3-9}$$

which is quadratic in $\boldsymbol{u}(t)$. Therefore, the closed-form solution for the minimizing $\boldsymbol{u}(t)$ can be obtained by taking the partial derivative with respect to $\boldsymbol{u}(t)$ and setting it equal to zero [99], obtaining

$$0 = 2\boldsymbol{u}(t)^\top R + \frac{\partial J(\boldsymbol{x}(t))}{\partial \boldsymbol{x}} g(\boldsymbol{x}(t)) \tag{3-10}$$

$$\implies \boldsymbol{u}^*(t) = -\frac{1}{2} R^{-1} g^\top(\boldsymbol{x}(t)) \frac{\partial J^\top(\boldsymbol{x}(t))}{\partial \boldsymbol{x}} \tag{3-11}$$

However, using the result from Equation (3-11) will result in long verification times, as the expression involves the partial derivative of $J$. Therefore, we approximate the optimal policy with a NN, called the *policy network*, with the goal to obtain a more compact representation of the optimal policy. The policy network parameters are updated by standard NN optimization algorithms, to minimize the *policy loss*, defined as

$$L_p = \frac{1}{N} \sum_{x_i \in \mathcal{X}} \left( \ell(x_i, \pi_{\boldsymbol{\theta}}(x_i)) + \frac{\partial J_{\boldsymbol{\omega}}(x_i)}{\partial \boldsymbol{x}} f(x_i, \pi_{\boldsymbol{\theta}}(x_i)) \right), \tag{3-12}$$

where $\mathcal{X}$ denotes a set of samples of the state vector in $D$ and $N = |\mathcal{X}|$. To ensure the policy remains stabilizing during this routine, we add the same *certificate loss* of Equation (3-6) to obtain the total loss term

$$L_{Tot} = \beta_p L_p + \beta_c L_c, \tag{3-13}$$

where $\beta_p$ and $\beta_c$ are scaling terms to put emphasis on the loss terms. How these scaling terms are determined is further explained in Section 3-1-4.

Pseudocode for the policy improvement procedure is given in Algorithm 2. To ensure compliance with the condition of Equation (2-8a), i.e. $\pi(0) = 0$, we translate the policy network at each iteration, by removing the offset from the bias term of the last layer of the network. This is denoted by the `MoveOriginToZero()` function in Algorithm 2. The routine runs until a certain `StoppingCondition` has been satisfied. Possible stopping conditions are a maximum number of iterations has occurred, or when $L_{Tot}$ decreases less than a certain threshold during an update step.

**Algorithm 2** Neural policy improvement

```
1: procedure NeuralPolicyImprovement($J, \alpha, \beta, \pi_{\boldsymbol{\theta}}, \ell, \mathcal{X}$)
2:     $N \leftarrow |\mathcal{X}|$
3:     $k \leftarrow 0$
4:     while not StoppingCondition do                      ▷ e.g. max number of iterations
5:         $L_{p^k} \leftarrow \frac{1}{N}\sum_{i=1}^{N}(\ell(x_i, \pi_{\boldsymbol{\theta}^k}(x_i)) + \frac{\partial J(x_i)}{\partial \boldsymbol{x}} f(x_i, \pi_{\boldsymbol{\theta}^k}(x_i)))$
6:         $L_{c^k} \leftarrow \frac{1}{N}\sum_{i=1}^{N}(\max(0, \frac{\partial J(x_i)}{\partial \boldsymbol{x}} f(x_i, \pi_{\boldsymbol{\theta}^k}(x_i)))$
7:         $L_{Tot^k} \leftarrow \beta_p L_{p^k} + \beta_c L_{c^k}$
8:         $\boldsymbol{\theta}^{k+1} \leftarrow \boldsymbol{\theta}^k - \alpha \nabla_{\boldsymbol{\theta}^k} L_{Tot^k}$          ▷ Update weights with preferred optimizer
9:         MoveOriginToZero()
10:        $k \leftarrow k+1$
11:    end while
12: end procedure
```

## 3-1-4 System verification

In the system verification step, a Satisfiability Modulo Theory (SMT) solver is utilized to find a state $x$ that violates the certificate conditions of Equation (2-4) on domain of interest $D$.

An SMT solver is a tool to formally verifying first order logic formulae. Given such a formula, the solver returns a point where the formula is satisfied, or `unsat` if no such point exists. The solver can be used to verify the certificate conditions of Equation (2-4), by constructing a first-order logic formula that expresses the negation of the these conditions. If the SMT solver finds a state for which the logic formula is satisfied, the certificate conditions do not hold and the state is considered a counter-example. If no such state can be found, the solver returns `unsat` and the certificate function is guaranteed to adhere to the conditions.

In this work we use SMT solver dReal [96], which is able to reason over transcendental functions. dReal is a $\delta$-complete solver, which entails that when it returns an `unsat` decision it will always be correct, but a `sat` decision comes with a $\delta$-error bound. In practice this might result in incorrect counter-examples, which can become problematic around the origin, where the certificate function is close to zero. We therefore omit a small region with radius $\epsilon$ around the origin to circumvent this complication.

The conditions for stability in Equation (2-4) are captured in the following first-order logic formulae:

$$\Phi_{L_1}(x) := \forall x \in D \setminus \left\{ x \in \mathbb{R}^n \,\middle|\, \sum_{i=1}^{n} x_i^2 \leq \epsilon \right\},\ V(x) > 0, \tag{3-14a}$$

$$\Phi_{L_2}(x) := \forall x \in D \setminus \left\{ x \in \mathbb{R}^n \,\middle|\, \sum_{i=1}^{n} x_i^2 \leq \epsilon \right\},\ \dot{V}(x) < 0. \tag{3-14b}$$

Notice that the first condition of Equation (2-4), i.e. $V(0) = 0$, is not explicitly verified, as the origin of the certificate network is translated to zero to ensure this property.

The conditions involving the Lie derivative of the certificate function are significantly more costly than the conditions on the certificate function itself. In order to reduce the total verification time, we prioritise the conditions of 3-14, such that the conditions on the Lie derivative are only checked if the other condition is deemed `unsat`, similar to [54].

To this end, the weights, biases and activation functions of the policy network and value network are used to represent the networks as symbolic expressions. The Lie derivative of the value network over vector field $f$ is constructed by taking the partial derivative and calculating the dot product with the system dynamics. The symbolic expression of the policy network is inserted into the system dynamics to create a single expression that solely depends on $x$.

Subsequently, dReal will check $\Phi_{L_1}$ and $\Phi_{L_2}$. If the solver returns `unsat`, the method is allowed to transition to the next routine. If a counter-example is found, the previous routine was unsuccessful and has to be further elaborated. In this case, the counter-example will be added to the training set and $\beta_c$ from Equation (3-7) will be increased to put more emphasis on the certificate loss.

The method also allows for verification of barrier functions. The conditions for safety in Equation (2-5) are captured in the following first-order logic formulae:

$$\Phi_{B_1}(x) := \forall x \in X_0,\ B(x) < 0, \tag{3-15a}$$

$$\Phi_{B_3}(x) := \forall x \in X_u,\ B(x) > 0, \tag{3-15b}$$

$$\Phi_{B_3}(x) := \forall x \in \{x \in D \mid B(x) = 0\},\ \dot{B}(x) < 0, \tag{3-15c}$$

with initial set $X_0 \in D$ and unsafe set $X_u \in D$.

### 3-1-5 Stopping conditions

The system verification steps in FNPI ensure the policy remains stabilizing throughout the process. Therefore, the process can be terminated any time a system verification step is successfully completed. Figure 3-2 schematically illustrates how the routines of FNPI interact in the ideal case. Because the process alternates between neural policy evaluation and neural policy improvement, the value function and policy will be jointly driven towards their respective optimum. If the optimum has been attained, the conditions from Theorem 3 are met, i.e.

$$\ell(\boldsymbol{x}(t), \pi(\boldsymbol{x}(t))) + \dot{J}(\boldsymbol{x}(t), \pi(\boldsymbol{x}(t))) = 0,\ \forall \boldsymbol{x} \in D.$$

If the process is terminated too early, the policy will likely be far from its optimum. On the other hand, if the process runs for a large number of iterations, the synthesis time might become intractable. We therefore propose the following stopping conditions, which the user can adapt to its needs.

1. Maximum number of iterations. The user defines the amount of iterations after which the process will terminate. If the user is not satisfied with the controller performance, additional iterations can be performed.

2. Maximum synthesis time. The user defines the maximum amount of time the synthesis procedure can consume. The procedure will return the last verified policy.

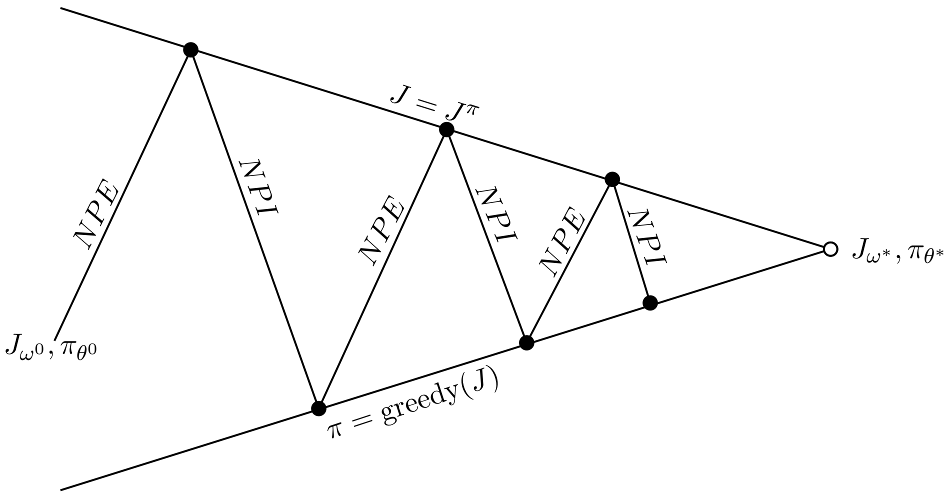

$J = J^{\pi}$
NPE
NPI
NPE
NPI
NPE
NPI
$J_{\omega^0}, \pi_{\theta^0}$
$J_{\omega^*}, \pi_{\theta^*}$
$\pi = \text{greedy}(J)$

**Figure 3-2:** Schematic view of Formal Neural Policy Iteration. NPE denotes neural policy evaluation, NPI denotes neural policy improvement and the black circles indicate a system verification step.

3. Policy improvement threshold. If during the neural policy improvement step the policy loss of Equation (3-12) fails to fall below a user defined threshold, the policy will be sufficiently close the the optimum and the process can therefore be terminated after a final system verification step.

## 3-2 Software implementation of FNPI

The methodology discussed in this chapter has been implemented in a prototype tool called FNPI. The learning of the network parameters is done with deep learning framework PyTorch [100]. PyTorch's dynamic computation graph allows for fast computation of the loss functions defined in Equations (3-5), (3-6) and (3-12), which involve calculating the Lie derivative of the network. The verification of the certificate conditions of Equations (2-4) and (2-5) is done with SMT solver dReal. dReal's delta-completeness allows for verification of nonlinear inequalities over the reals, therefore permitting the use of nonlinear activation functions in the networks.

The tool is available at: github.com/JonathanKLSCH/formal-neural-controller-synthesis. More details on the structure of the tool is given in Appendix A.

Chapter 4

# Results

In this chapter we subject the discussed procedure to various systems. First, in Section 4-1, a linear system with quadratic cost is considered, to demonstrate that the method is able to converge to the analytical solution. Section 4-2 presents a scalability study of verification times with respect to the size of the certificate networks. In Section 4-3-2, we consider systems with input and state constraints. Case studies on nonlinear plants are described in Section 4-4. Finally, the results are summarized and discussed in Section 4-5.

All experiments are performed on a laptop workstation running MacOS Big Sur 11.2.1, with a 2 GHz Quad-Core Intel Core i7 processor and 8 GB 1600 MHz DDR3 RAM.

## 4-1 Verification of the procedure with linear systems

To verify the correctness of the synthesis method, we test the method on linear systems with quadratic cost. There exists an analytical solution of the value function and policy, provided by Linear-Quadratic Regulator (LQR) theory [2]. We show for the double integrator that, after seven iterations, the method has converged to the analytical solution.

### 4-1-1 Derivation of the Linear Quadratic Regulator

Consider a system with linear dynamics $\dot{x} = Ax + Bu$ and a quadratic cost function of the form $\ell(x, u) = x^\top Qx + u^\top Ru$ with symmetric matrices $Q$ and $R$, which are positive semi-definite and positive definite, respectively. Assuming the optimal value function takes the form $J^*(x) = x^\top Sx$, the Hamilton-Jacobi-Bellman (HJB) equation takes the form

$$0 = \min_u \left[ x^\top Qx + u^\top Ru + 2x^\top SAx + 2x^\top SBu \right], \tag{4-1}$$

which is quadratic in $u$. The minimum of the function can be found by taking the partial derivative with respect to $u$ and setting it equal to zero

$$\frac{\partial}{\partial u}\left[x^\top Qx + u^\top Ru + 2x^\top SAx + 2x^\top SBu\right] = 2u^\top R + 2x^\top SB = 0 \tag{4-2}$$

$$\implies u^* = -\underbrace{R^{-1}B^\top S}_{K} x \tag{4-3}$$

Substituting $u^*$ back into 4-1 results in the Continuous Algebraic Riccati Equation (CARE), which can be used to compute $S$:

$$0 = x^\top \left[Q - SBR^{-1}B^\top S + SA + A^\top S\right] x. \tag{4-4}$$

## 4-1-2 Double integrator

Consider a two-dimensional system with linear dynamics

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \tag{4-5}$$

and quadratic cost function $\ell(x,u) = x^\top Qx + u^\top Ru$ with matrices $Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and $R = 1$.

We consider domain of interest $D = 0.1 \leq \sqrt{x_1^2 + x_2^2} \leq 3$

The analytical solution of the optimal value function and optimal policy are, respectively,

$$J^*(x) = x^\top \begin{bmatrix} \sqrt{3} & 1 \\ 1 & \sqrt{3} \end{bmatrix} x, \quad \pi^*(x) = -\begin{bmatrix} 1 & \sqrt{3} \end{bmatrix} x. \tag{4-6}$$

For our method, we use a single-layer linear network without bias for the policy, i.e. $\pi(x) = -\begin{bmatrix} k_1 & k_2 \end{bmatrix} x$, initialized at $K = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}$. The value function is approximated by a 2-5-1 value network $J(x)$, with a quadratic activation function $\sigma(x) = x^2$ for the middle layer and linear activation function for the final layer.

After 7 iterations the controller gains are within $1e^{-4}$ of the analytical solution and the mean squared error between $J^*(x)$ and $J(x)$ over 2601 samples, uniformly distributed between $x_1 \in [-3,3]$ and $x_2 \in [-3,3]$, is $4e^{-4}$. Figure 4-1 shows the evolution of the controller gains over 7 iterations. Figure 4-2 shows the output of $J(x)$.

Next, the value network and policy network are passed on to the `Verifier`, which first transforms the networks into their symbolic counterpart. Finally, the first-order logic formula of Equation (3-14) is evaluated, for which dReal returned `unsat`. Therefore, the policy stabilizes the system on $D$, as $J(x)$ serves as a Lyapunov function.

## 4-1-3 Conclusion

In this section, we verified the correctness of the synthesis procedure, by showing the method is able to converge to the analytical solution of the policy and the value function. Furthermore, Satisfiability Modulo Theory (SMT) slover dReal confirms that the Lyapunov conditions hold for the value network, therefore the value network is a Lyapunov function and formally verifies stability.

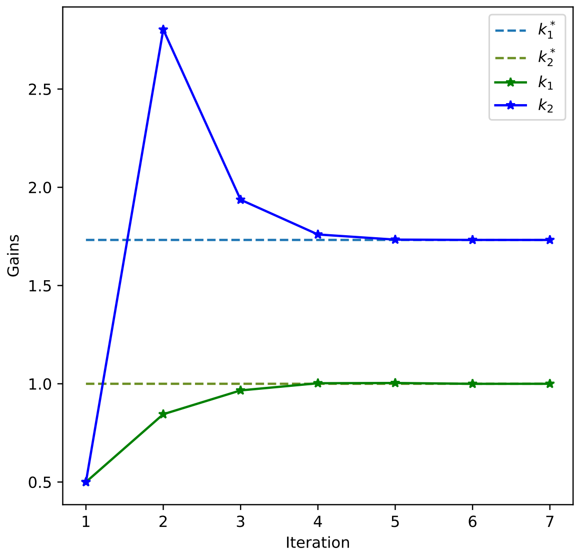


**Figure 4-1:** Evolution of the controller gains over 7 iterations.

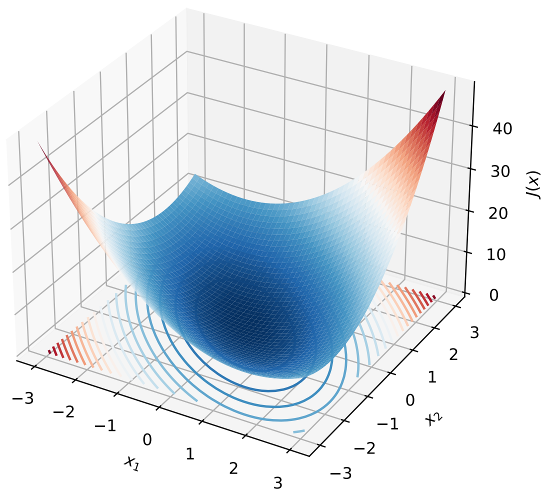


**Figure 4-2:** Output of value network $J(x)$

## 4-2 Scalability of the method

In the previous section, we verified the correctness of the synthesis procedure, by demonstrating that the method is able to converge to the analytical solution and that the value function can serve as a Lyapunov function. In the case study, we used relatively small networks for the policy and the value function. For more intricate systems, such as systems with input and state constraints or nonlinear systems, we need Neural Networks (NNs) that can sufficiently approximate the value function and the policy. The expressiveness of NNs increases as the amount of layers and nodes per layer increases, but the verification time increases as well.

To get a grasp on how the verification times scale with various network topologies, we synthesized Lyapunov functions for a two-dimensional linear plant with one input. The value and policy networks consist of one, two or three hidden layers with 4 to 100 nodes per layer. The activation functions are restricted to be smooth, therefore Quadratic, Tanh and Sigmoid Linear Unit (SiLU) activation functions are used. The policy network is initialized to approximate the LQR solution. Thereafter, we synthesize Lyapunov functions for linear systems with two states with one input, two states with two inputs, three states with one input and finally four states with one input.

### 4-2-1 Value network scaling

Figure 4-3 depicts the verification times with various value network topologies for a two-dimensional system. The policy is linear and the gains are initialized at the optimal solution, where after the value network was trained until convergence. We can conclude that the verification times of single hidden layer value networks grows approximately linear for all activation functions, while the verification times two and three layer value networks grows exponentially, regardless of the activation function.

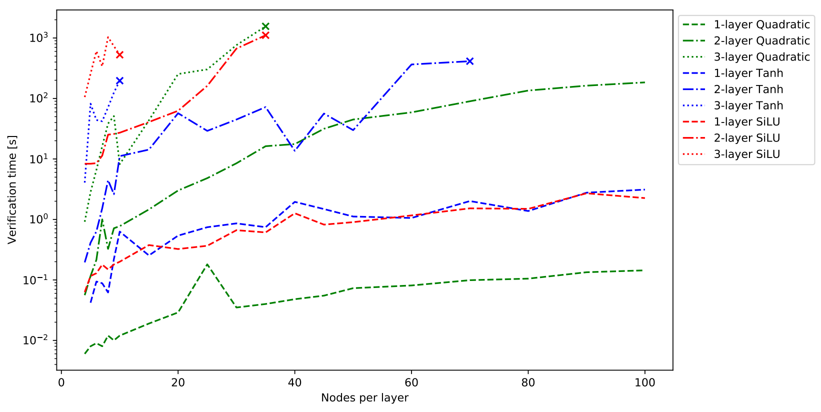


**Figure 4-3:** Verification times for various value network topologies. $\times$ denotes that topology was the maximum amount of nodes possible before the verification time limit was reached.

## 4-2-2 Policy network scaling

We conduct a similar experiment for the verification times of the system with various policy network topologies. The value network is a $2-20-20-1$ network with quadratic activation function for the hidden layers and linear activation for the final layer. The results are plotted on log scale in Figure 4-4. We conclude that the verification times scale better compared to when the value network is scaled, which can be explained by the fact that the value networks Lie derivative has to be evaluated. However, verification times for two and three hidden layer policy networks also grows exponentially.

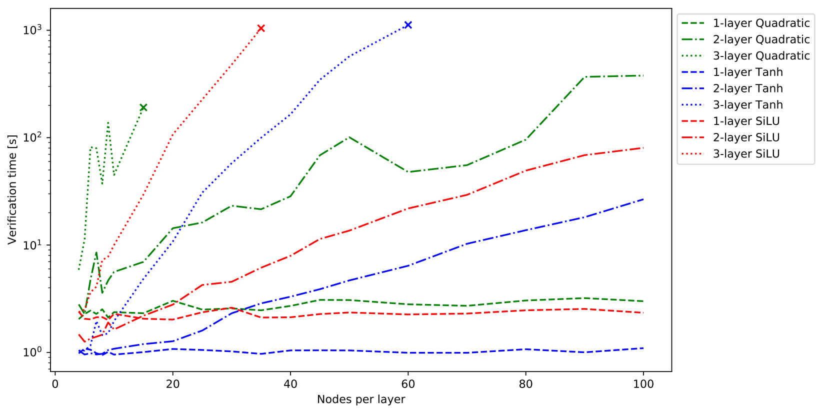


**Figure 4-4:** Verification times for various policy network topologies. $\times$ denotes that topology was the maximum amount of nodes possible before the verification time limit was reached.

## 4-2-3 Single layer scaling

The scaling experiments from the previous paragraphs show, for both the value network and policy network, linear scaling for single hidden layer networks and exponential scaling for two and three hidden layer networks. Motivated by these results and the fact that a network with sufficient nodes in a single hidden layer with non-polynomial activation function is a

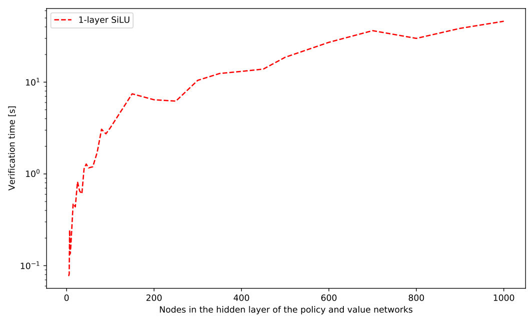


**Figure 4-5:** Verification times for single hidden layer policy and value networks.

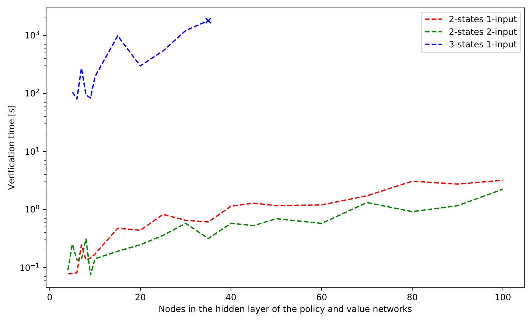


**Figure 4-6:** Verification times for various system dimensions. $\times$ denotes the topology was the maximum amount of nodes possible before the verification time limit was reached.

universal function approximator [76, 77], we evaluate how the verification times scale when using a single layer network with SiLU activation for both the policy as the value function. The results are shown in Figure 4-5, which shows approximately linear scaling up to 1000 nodes.

### 4-2-4 Higher-dimensional systems

The experiments in the Sections 4-2-1, 4-2-2 and 4-2-3 were conducted on a two-dimensional linear system with one input. Networks with a single hidden layer with SiLU activation function were found to have good function approximation capabilities while having a short verification time. In this section we investigate how the verification times scale with other system dimensions when using single hidden layer networks with SiLU activation functions for both the policy and value network.

The systems considered are linear with two states with one input, two states with two inputs, three states with one input and finally four states with one input. The results are plotted in Figure 4-6. It becomes apparent that the amount of inputs has little effect on the verification time, as the verification times for systems with two states and one or two inputs is of the same order of magnitude. The amount of states, however, does affect the verification time significantly, as the verification times for a system with three states and one input is three orders of magnitude higher, compared to a system with two states and one input. Furthermore, we were unable to verify a four dimensional system with one input, as the verification time limit was reached for all network topologies.

### 4-2-5 Conclusion

From the results from Sections 4-2-1 and 4-2-3, we conclude that the use of single hidden layer networks with non-polynomial activation functions is sufficient for the function approximation capabilities of the network, and beneficial to the verification time. This is in contrast to [54], where they use deep networks (up to 10 layers) with linear and polynomial activation functions over adjacent layers.

## 4-3 State & input constraints

In the previous sections we demonstrated that, for linear systems with quadratic cost, the method is able to converge to the analytical solution and the value network serves as a Lyapunov function. Furthermore, we concluded that single hidden layer networks make the best trade-off between approximation capabilities and verification time.

In this section we demonstrate the flexibility of the synthesis framework, by showing we can use input constraints and *cost function shaping* to achieve more complex control specifications.

### 4-3-1 Input constraints

In order to enforce input constraints, we can append the policy network with an extra layer with bounded activation function, such as $\texttt{HardTanh}(x) = c \cdot \max(-1, \min(1, x))$, where positive constant $c$ is the weight parameter of the layer. By setting the bias and weight of the final layer to zero and $c$, respectively, and *freezing* the parameters thereafter, the outputs remain constraint to $[-c, c]$ throughout the synthesis procedure.

We take the double integrator of Equation (4-5), a $2 - 300 - 1 - 1$ policy network with $\texttt{SiLU} - \texttt{Linear} - \texttt{HardTanh}$ activation and a $2 - 300 - 1$ value network with $\texttt{SiLU} - \texttt{Linear}$ activation. The policy network is initialized at the LQR solution, but truncated at $[-2, 2]$. We perform a neural policy evaluation step, to train the value network, which is subsequently verified to serve as a Lyapunov function. The output of the policy network and the value network are depicted in Figure 4-7 and Figure 4-8, respectively.

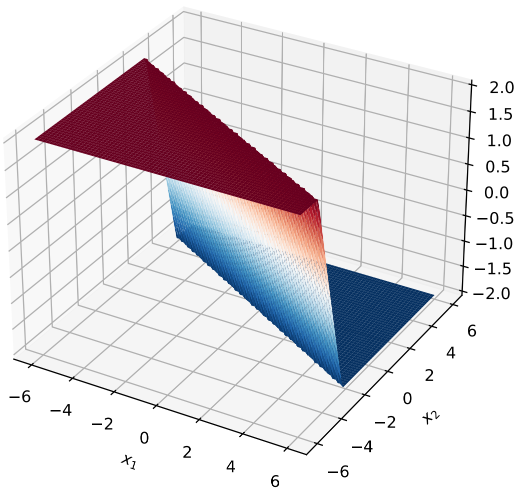


**Figure 4-7:** Constrained policy for the double integrator

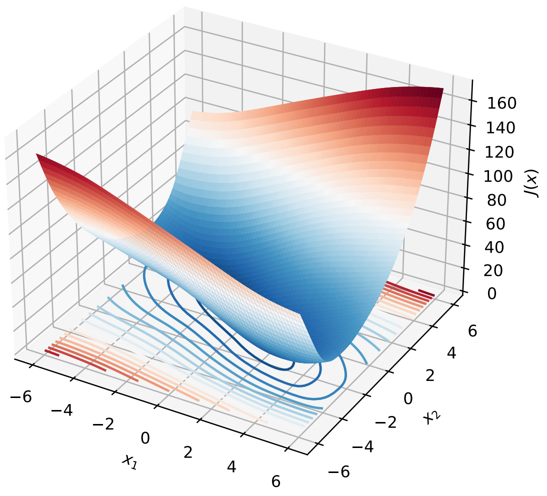


**Figure 4-8:** Value function for the double integrator with constrained policy.

### 4-3-2 State constraints via cost function shaping

In the policy evaluation step of Section 3-1-2, the value network is trained to match its Lie derivative with the negative cost function, i.e. $\frac{\partial J(\boldsymbol{x}(t))}{\partial \boldsymbol{x}} f(\boldsymbol{x}(t), \boldsymbol{u}(t)) \approx -\ell(\boldsymbol{x}(t), \boldsymbol{u}(t))$. The cost function can, therefore, be used as a design tool to synthesize controllers with certain specifications. The cost function is not restricted to be linear, but can consist of multiple terms, as long as the cost function and the policy comply with the conditions from Equation (2-8).

In the next two examples we show we can shape the cost function such that the synthesized controllers avoid certain regions of the state space. Furthermore, we show that for one example, the value function can serve as a barrier certificate, in order to give safety guarantees.

**Velocity constraint** Consider the double integrator system from Equation (4-5). The control objective is to stabilize the system at the origin, while avoiding $|x_2| > 3$. To this end, the cost function, depicted in Figure 4-9, is a combination of quadratic cost $\ell_1$ and an exponential function $\ell_2$, defined by

$$\ell_1(x, u) = x^\top Q x + u^\top R u, \tag{4-7}$$

$$\ell_2(x) = \left(\frac{x_2}{b}\right)^8, \tag{4-8}$$

$$\ell_{Tot}(x, u) = \ell_1(x, u) + \ell_2(x), \tag{4-9}$$

with $Q = \begin{bmatrix} 1 & 0 \\ 0 & 0.001 \end{bmatrix}$, $R = 0.01$ and $b = 0.97$. For the policy and the value function, we use two $2 - 500 - 1$ NNs with learning rate $\alpha = 0.01$, SiLU activation functions for the hidden layer and a linear activation function for the last layer. For the initial policy, we use linear controller obtained by calculating the LQR gains, using $\ell_1$ only.

The procedure performed three iterations. Figure 4-10 shows the trajectories for various starting conditions of the double integrator, subject to the initial policy and the improved policy after one and three iterations. The improved controllers are able to avoid states where $|x_2| > 3$, while remaining stabilizing. Moreover, all trajectories attain a lower additive cost compared to the initial policy, as detailed in Table 4-1.

**Table 4-1:** Cost comparison between the initial policy and the optimized policy for 10 trajectories of the double integrator with velocity constraint.

| Trajectory | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Initial policy | $2.77 \times 10^6$ | $4.66 \times 10^5$ | $4.67 \times 10^4$ | $1.86 \times 10^3$ | $1.90 \times 10^1$ | $1.90 \times 10^1$ | $1.86 \times 10^3$ | $4.67 \times 10^4$ | $4.66 \times 10^5$ | $2.77 \times 10^6$ |
| $3^{rd}$ iteration | 111.31 | 45.06 | 20.72 | 8.80 | 2.21 | 2.28 | 9.37 | 24.97 | 83.43 | 360.03 |
| Difference | $2.77 \times 10^6$ | $4.66 \times 10^5$ | $4.67 \times 10^4$ | $1.85 \times 10^3$ | $1.68 \times 10^1$ | $1.67 \times 10^1$ | $1.85 \times 10^3$ | $4.67 \times 10^4$ | $4.66 \times 10^5$ | $2.77 \times 10^6$ |

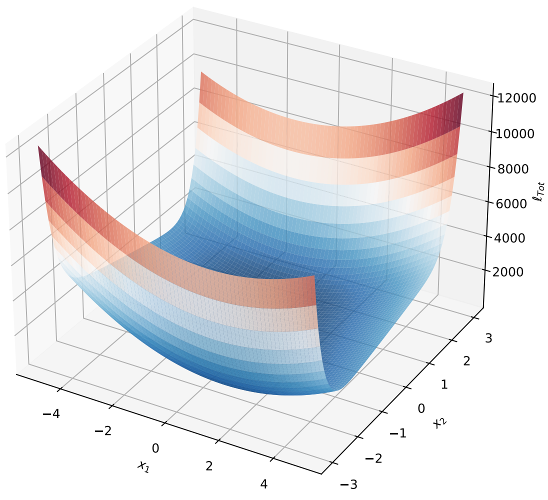


**Figure 4-9:** Shape of cost function $\ell_{Tot}$.

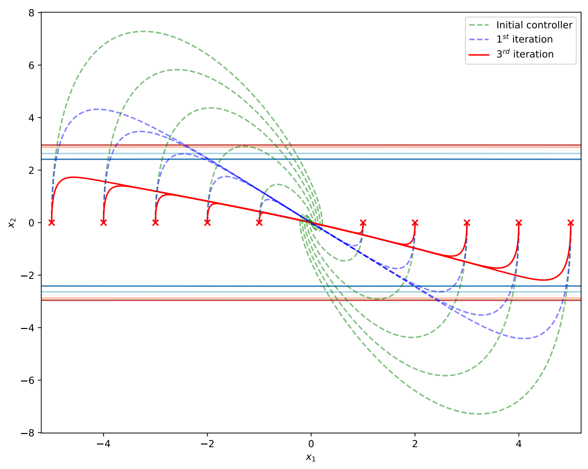


**Figure 4-10:** Trajectories of the double integrator subject to the initial linear policy, the resulting policy after one iteration and after three iterations. The color of the $\times$ marks denotes which policy attains the lowest additive cost for that trajectory.

**Specific region constraint** Consider again the double integrator system from Equation (4-5). The cost function, depicted in Figure 4-11, is a combination of quadratic cost $\ell_1$ and a two-dimensional Gaussian function $\ell_2$, defined by

$$\ell_1(x,u) = x^\top Q x + u^\top R u, \tag{4-10}$$

$$\ell_2(x) = P \cdot \exp\left(-\frac{(x_1-c_1)^2+(x_2-c_2)^2}{2\sigma^2}\right), \tag{4-11}$$

$$\ell_{Tot}(x,u) = \ell_1(x,u) + \ell_2(x), \tag{4-12}$$

with $Q = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$, $R = 1$, $P = 180$, $\sigma = 0.4$ and $[c_1, c_2] = [2.4, -1.7]$. For the policy and the value function, we use two $2-200-200-1$ NNs with learning rate $\alpha = 0.01$, SiLU activation functions for the hidden layers and a linear activation function for the last layer. For the initial policy, we use linear controller obtained by calculating the LQR gains, using $\ell_1$ only.

The procedure performed two iterations. Figure 4-12 shows the trajectories for various starting conditions of the double integrator subject to the initial policy and the improved policy after one and two iterations. The improved controllers are able to avoid the region of states where the effect of $\ell_2$ is significant. As detailed in Table 4-2, trajectories $6-10$ attain a lower additive cost when subject to the optimized policies. For trajectories $1-5$, however, the linear controller attains a lower additive cost. As the effect of $\ell_2$ is negligible for trajectories $1-5$, and the linear policy is optimal with respect to $\ell_1$, deviations from these trajectories will be sub-optimal. Furthermore, the NN used for this experiment is too large to be verified by dReal in reasonable time.

**Table 4-2:** Cost comparison between the initial policy and the optimized policy for 10 trajectories of the double integrator with specific region constraint.

| Trajectory | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Initial policy | $4.33 \times 10^{-1}$ | 1.73 | 3.93 | 8.24 | $2.44 \times 10^{1}$ | $6.65 \times 10^{1}$ | $1.12 \times 10^{2}$ | $1.33 \times 10^{2}$ | $1.38 \times 10^{2}$ | $1.38 \times 10^{2}$ |
| $2^{nd}$ iteration | $7.58 \times 10^{-1}$ | 3.01 | 6.10 | $1.34 \times 10^{1}$ | $2.58 \times 10^{1}$ | $4.48 \times 10^{1}$ | $6.37 \times 10^{1}$ | $7.82 \times 10^{1}$ | $8.75 \times 10^{1}$ | $6.73 \times 10^{1}$ |
| Difference | $-3.25 \times 10^{-1}$ | $-1.28$ | $-2.16$ | $-5.11$ | $-1.30$ | $2.16 \times 10^{1}$ | $4.79 \times 10^{1}$ | $5.47 \times 10^{1}$ | $5.03 \times 10^{1}$ | $7.04 \times 10^{1}$ |

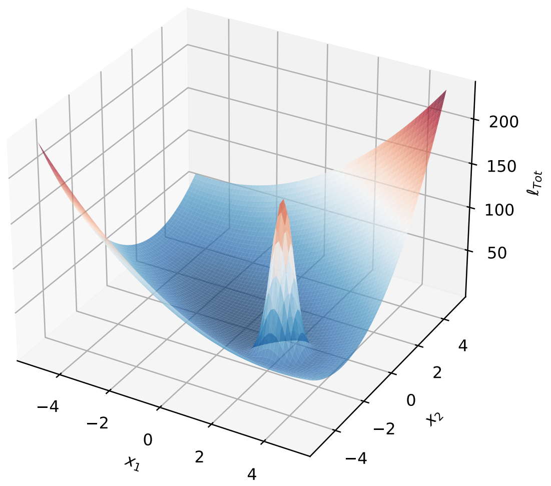


**Figure 4-11:** Shape of cost function $\ell_{Tot}$.

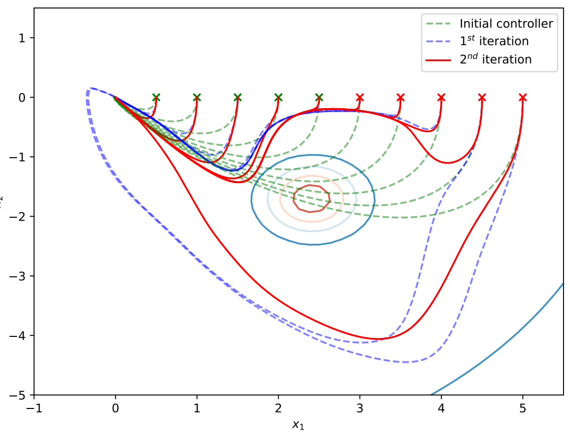


**Figure 4-12:** Trajectories of the double integrator subject to the initial linear policy, the resulting policy after one iteration and after two iterations. The color of the $\times$ marks denotes which policy attains the lowest additive cost for that trajectory.

### 4-3-3 Using the value function as a barrier function.

In this section, we investigate the possibility of using the value function as a barrier function for safety certification. The value network can be translated, by adding the offset to the bias term of the last layer of the network. Hereby, a certain level set of the value network can be translated to zero, rendering the interior of that level set negative and the exterior of the level set positive. Since the Lie derivative of the value network is strictly negative everywhere except at the origin, the conditions from Theorem 2 are met and the value function serves as a barrier function. Therefore, all trajectories starting in the interior of the level set will never reach the exterior of the level set.

Figure 4-13 and 4-14 show the level sets of the value functions from the examples of the previous section. The level sets in red are fully contained in the safe region of the respective experiments. One can conclude from Figure 4-13, that, for the velocity constraint problem, the procedure adapts the level set to the safe region to a certain extent. Moreover, the level set is significantly larger than the level set of the value function of the initial policy. However, for the specific region constraint problem, the level set does not exclude the constraint region. Although the trajectories in Figure 4-12 are able to avoid the constraint region, the value function in this experiment does not have the adequate shape to serve as a barrier function and therefore the system and policy are not verifiably safe.

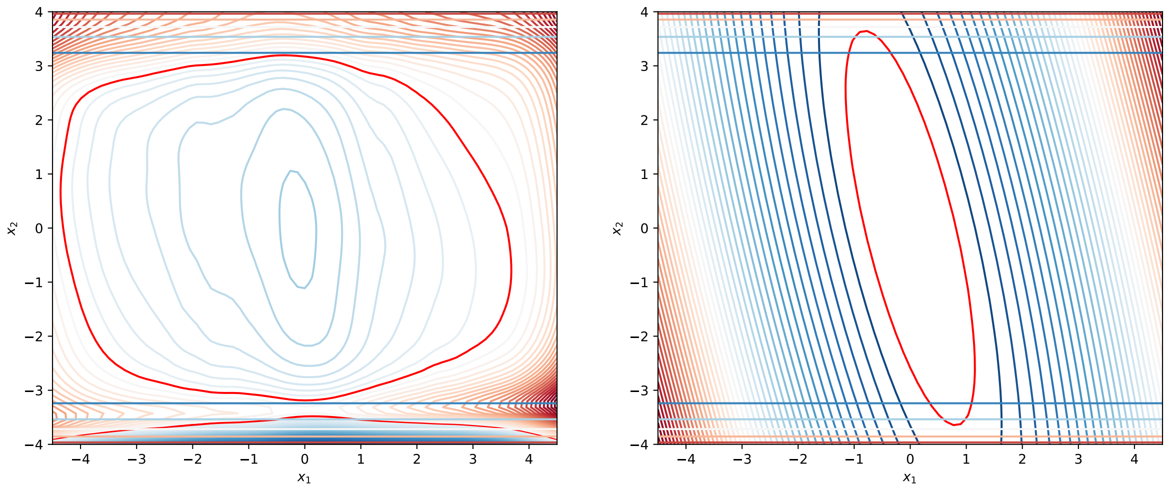


**Figure 4-13:** Left: level sets of the value function for the velocity constraint example, after three iterations of the procedure. Right: level sets of the value function of the linear policy.

### 4-3-4 Conclusion

In this section, we demonstrated the versatility of the method, by showing it is able to enforce input constraints and state constraints in specific cases. Input constraints are realized by appending the policy network with an extra layer with bounded activation function. State constraints can be enforced by designing the cost function in such a way that the value function can serve as a barrier function. However, in the specific region constraint example, we were not able to successfully utilize the value function as a barrier function. More constructive methods to design the cost function are needed in this regard.

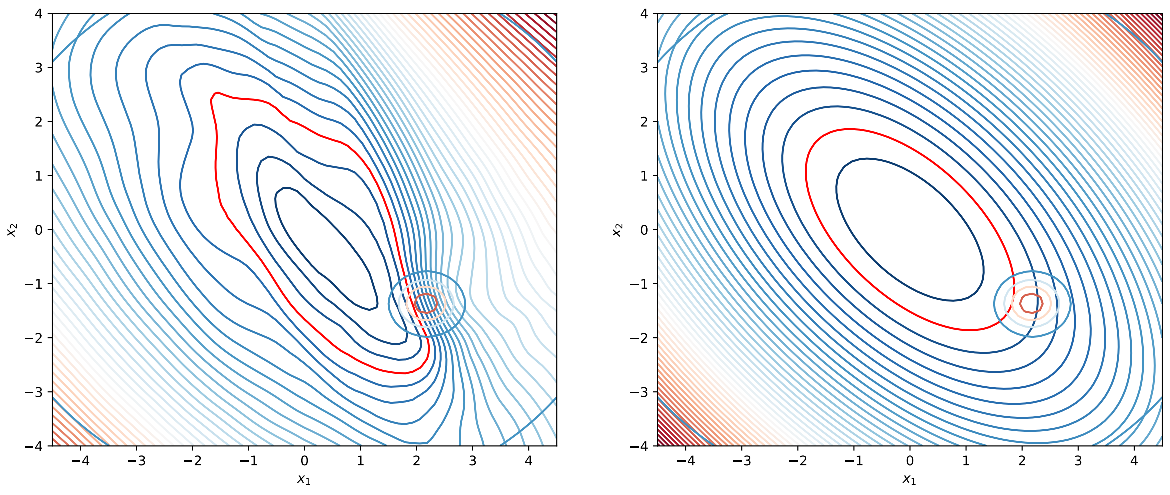


**Figure 4-14:** Left: level sets of the value function for the specific region constraint example, after two iterations of the procedure. Right: level sets of the value function of the linear policy.

## 4-4 Nonlinear systems

In this section we test the method on three nonlinear systems. First, Dubins car from [101] is considered. Next, the Duffing oscillator from [57] is examined. Finally, we test the method on the inverted pendulum from [55].

### 4-4-1 Dubins car

In this example, the control objective is to steer a car with constant velocity to track a reference trajectory, which is a straight line along the x-axis in this case. As depicted in Figure 4-16, the states of the system are the distance error $d_e$, which is the absolute distance between the car and the reference trajectory, and the angle error $\theta_e$ between the heading of the car and the reference trajectory. The equations of motion of Dubins car are described by

$$\begin{bmatrix} \dot{d}_e \\ \dot{\theta}_e \end{bmatrix} = \begin{bmatrix} v \cdot \sin(\theta_e) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} u, \tag{4-13}$$

with velocity $v = 1$. We use a quadratic cost function $\ell(x, u) = x^\top Qx + u^\top Ru$, with $Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and $R = 1$. For the policy and the value function, we use two $2-200-1$ NNs with learning rate $\alpha = 0.01$, SiLU activation functions for the hidden layer and a linear activation function for the last layer. We consider domain of interest $D = 0.2 \leq \sqrt{x_1^2 + x_2^2} \leq 5$

During initialization, the policy network is trained to approximate the LQR solution of the linearized system ($K = \begin{bmatrix} -1 & -\sqrt{3} \end{bmatrix}$), after which two iterations of the procedure occur. Figure 4-15 shows the initial policy and the optimized policies after each iteration.

Figure 4-17 shows several trajectories of the system when it is controlled by the initial policy, the optimized policy after one iteration and the optimized policy after two iterations. Compared to the initial policy, the optimized policy after two iterations attains a lower additive

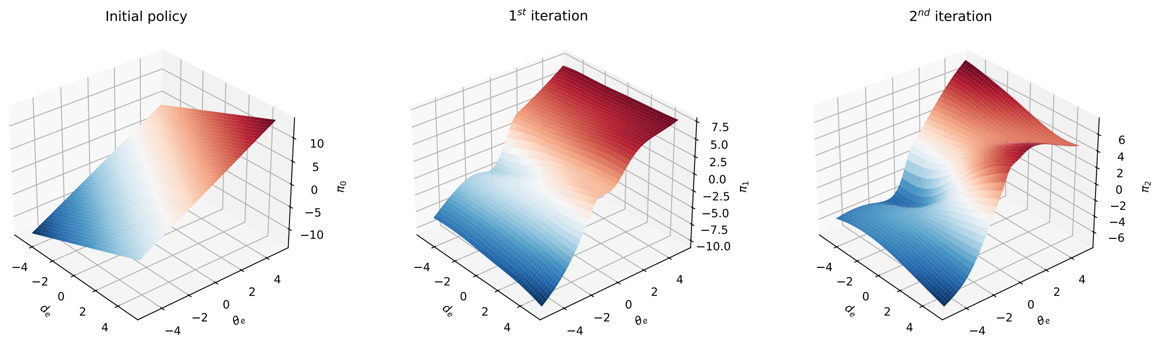


**Figure 4-15:** The policies for Dubins car at initialization, after one iteration and after two iterations of the method.

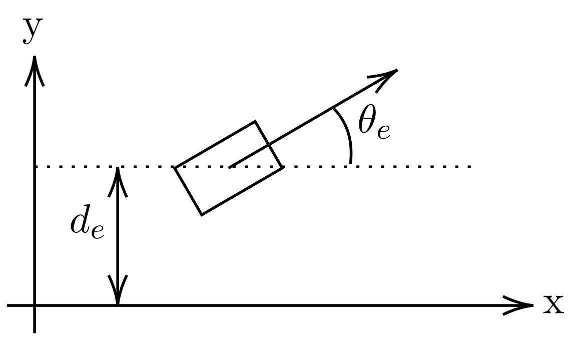


**Figure 4-16:** Schematic view of Dubins car.

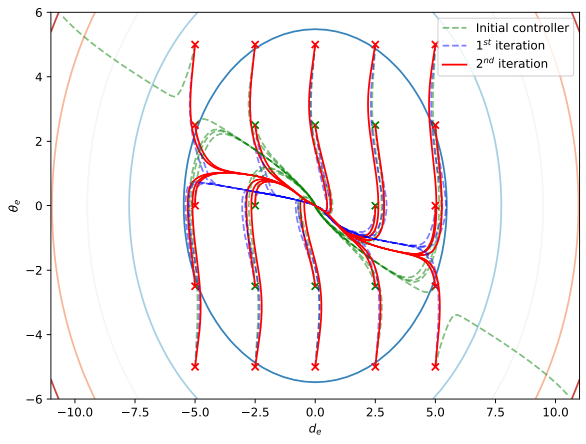


**Figure 4-17:** Trajectories of Dubins car, subject to the initial linear policy, the resulting policy after one iteration and after two iterations. The color of the × marks denotes which policy attains the lowest additive cost for that trajectory

cost for 16 out of 24 trajectories. Furthermore, the optimized policies are able to stabilize the system for all starting conditions, unlike the initial policy. An overview of synthesis times and a detailed cost comparison of the trajectories are given and further discussed in Section 4-4-4.

### 4-4-2 Duffing oscillator

The Duffing oscillator is a nonlinear second-order differential equation, used as an approximate model of many physical systems [102]. In this example, the control objective is to drive the trajectories of the system to the origin. We consider the forced and damped version of the Duffing oscillator, which results in the following equations of motion:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} y \\ -0.6y - x - x^3 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u. \tag{4-14}$$

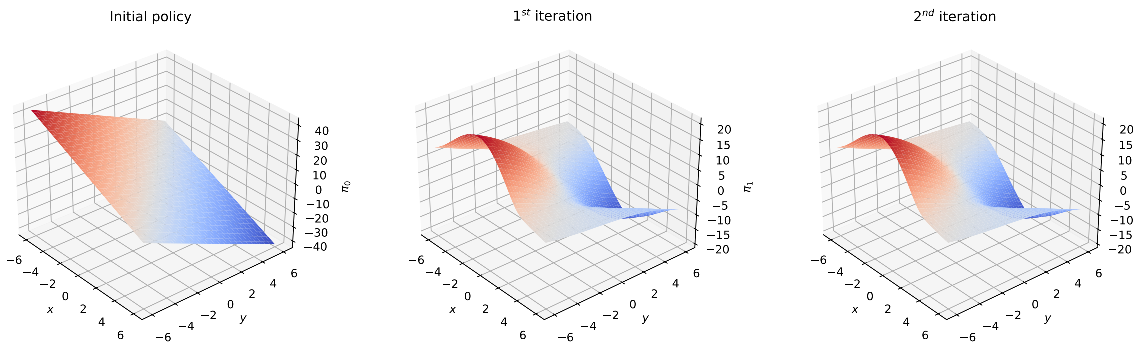


**Figure 4-18:** The policies for the Duffing oscillator at initialization, after one iteration and after two iterations of the method.

We use a quadratic cost function $\ell(x,u) = x^\top Qx + u^\top Ru$, with $Q = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$ and $R = 0.1$. For the policy and the value function, we use two $2 - 300 - 1$ NNs with learning rate $\alpha = 0.01$, SiLU activation functions for the hidden layer and a linear activation function for the last layer. We consider domain of interest $D = 0.2 \leq \sqrt{x_1^2 + x_2^2} \leq 6$.

During initialization, the policy network is trained to approximate the LQR solution of the linearized system ($K = \begin{bmatrix} 3.58 & 3.59 \end{bmatrix}$), after which two iterations of the procedure occur. Figure 4-18 shows the initial policy and the optimized policies after each iteration.

Figure 4-19 shows several trajectories of the system when it is controlled by the initial policy and the optimized policy after two iterations. The optimized policy after two iterations attains a lower additive cost for 10 trajectories, compared to the initial policy, but a higher additive cost for the other 14. An overview of synthesis times and a detailed cost comparison of the trajectories are given and further discussed in Section 4-4-4.

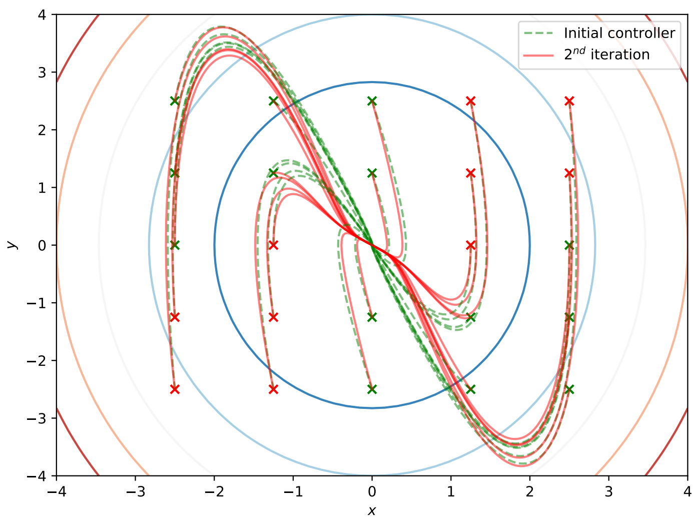


**Figure 4-19:** Trajectories of the Duffing oscillator, subject to the initial linear policy and the resulting policy after two iterations. The color of the $\times$ marks denotes which policy attains the lowest additive cost for that trajectory

### 4-4-3 Inverted pendulum

The inverted pendulum consists of a massless rod that is on one end attached to a pivot point and on the other end attached to a mass. The control input applies a torque at the pivot point. Friction is present in the system, which is proportional to the angular velocity. The control objective is to stabilize the inverted pendulum in the upright position. The equations of motion of the inverted pendulum are described by

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ \dfrac{mLg \cdot \sin(x_1) - bx_2}{mL^2} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u, \tag{4-15}$$

where mass $m = 0.15$, rod length $L = 0.5$, gravitational constant $g = 9.81$ and friction coefficient $b = 0.1$. We use a quadratic cost function $\ell(x,u) = x^\top Q x + u^\top R u$, with $Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and $R = 1$. For the policy and the value function, we use two $2-300-1$ NNs with learning rate $\alpha = 0.01$, SiLU and linear activation for the hidden and final layer, respectively. We consider domain of interest $D = 0.5 \leq \sqrt{x_1^2 + x_2^2} \leq 6$.

During initialization, the policy network is trained to approximate the LQR solution of the linearized system ($K = \begin{bmatrix} 39.27 & 6.64 \end{bmatrix}$), after which three iterations of the procedure occur. Figure 4-20 shows the resulting policies after each iteration.

Figure 4-21 shows several trajectories of the system when it is controlled by the initial policy, the optimized policy after one iteration and the optimized policy after three iterations. In contrast to the previous two examples, here the policy after the first iteration performs better than after two and three iterations. An overview of synthesis times and a detailed cost comparison of the trajectories are given and further discussed in the next section.

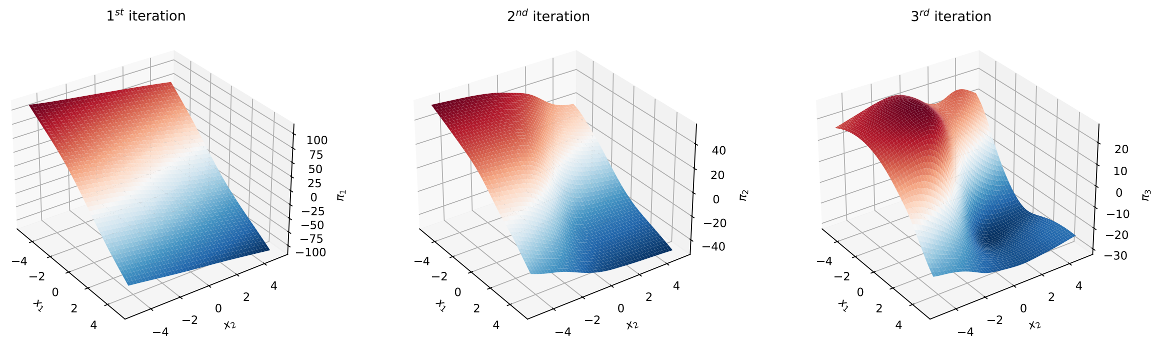


**Figure 4-20:** The resulting policies for the inverted pendulum after one, two and three iterations of the method.

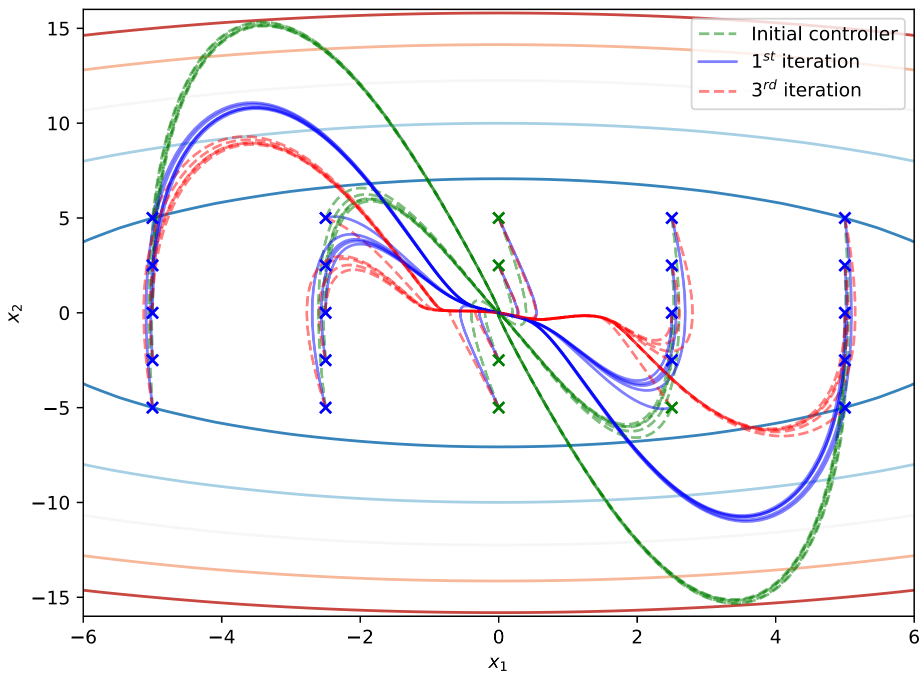


**Figure 4-21:** Trajectories of the inverted pendulum subject to the initial linear policy, the resulting policy after one iteration and after three iterations. The color of the $\times$ marks denotes which policy attains the lowest additive cost for that trajectory

## 4-4-4 Analysis & comparison

We have tested the method on three nonlinear systems. The synthesis times of our method are now compared to synthesis times of related literature. Furthermore, we compare the performance of the controllers synthesized by our method with the initial linear controllers.

**Synthesis times** The durations of the different routines of Formal Neural Policy Iteration (FNPI) are depicted in Table 4-3. Compared to the synthesis times of [55], our method is up to 300 times slower for both synthesis and verification routines. The increase in synthesis time is largely accredited to the way the certificate networks are utilized. In [55], the certificate networks solely function as Lyapunov functions, whereas in our experiments the certificate networks also serve as value functions. The increase in verification time can be explained by the fact that [55] uses significantly smaller networks for the Lyapunov function ($2-6-1$ for the inverted pendulum), and a linear policy.

Other formal neural controller synthesis methods, such as [52] and [57], although they solve slightly different problems, boast up to three orders of magnitude lower synthesis and verification times. Possible explanations are the use of different SMT solvers (Z3 and iSAT3, respectively), and the special structures of the NNs used. The NNs used in [52] have no bias term and quadratic activation functions, thereby restricting the NNs to be polynomial. In [57], a piece-wise linear approximation of the activation functions is used during verification.

**Table 4-3:** Synthesis times and number of iterations for Dubins car, the Duffing oscillator and the inverted pendulum. NPE denotes neural policy evaluation, NPI denotes neural policy iteration

| | Dubins car | | | Duffing oscillator | | | Inverted pendulum | | |
|---|---|---|---|---|---|---|---|---|---|
| | Stage | Sub-iters | Time [s] | Stage | Sub-iters | Time [s] | Stage | Sub-iters | Time [s] |
| Iter = 1 | NPE | 1 | 1416.37 | NPE | 1 | 3563.55 | NPE | 3 | 5392.47 |
| | Verify | 1 | 108.96 | Verify | 1 | 172.77 | Verify | 3 | 2502.28 |
| | NPI | 1 | 112.36 | NPI | 1 | 71.35 | NPI | 1 | 87.40 |
| | Verify | 1 | 147.842 | Verify | 1 | 155.76 | Verify | 1 | 1113.48 |
| 2 | NPE | 1 | 343.10 | NPE | 1 | 895.58 | NPE | 1 | 1073.39 |
| | Verify | 1 | 400.55 | Verify | 1 | 436.10 | Verify | 1 | 124.24 |
| | NPI | 1 | 156.47 | NPI | 1 | 66.46 | NPI | 1 | 81.15 |
| | Verify | 1 | 468.94 | Verify | 1 | 350.38 | Verify | 1 | 197.09 |
| 3 | | | | | | | NPE | 1 | 695.32 |
| | | | | | | | Verify | 1 | 120.26 |
| | | | | | | | NPI | 1 | 79.23 |
| | | | | | | | Verify | 1 | 162.51 |
| Total | | | 3154.60 | | | 5711.95 | | | 11628.82 |

**Cost comparison** For each experiment, we calculated the additive cost for several trajectories of the system when it is controlled by the initial policy and when it is controlled by the optimized policy. The results are shown in Table 4-4. For the Dubins car experiment, 16 trajectories attain a lower additive cost with the optimized policy than with the initial policy. For the remaining 8 trajectories, our controller performs roughly similar to the linear controller, as the costs of the policies are within 5% of each other. Furthermore, trajectories 1 and 24 are, in contrast to the linear policy, stabilized by the optimized policy and therefore show a significantly lower cost.

For the Duffing oscillator, 10 out of 24 trajectories accumulate a lower cost with the optimized policy. On average, the policy obtained by our method performs roughly similar to the linear policy. Although our policies do not convincingly outperform the linear policy in this example, we are able to give stability guarantee for all policies, as the value function was successfully verified as a Lyapunov function after each iteration.

In the inverted pendulum example, the optimized policy after one iteration outperforms the linear policy, but the policies of subsequent iterations perform worse. Therefore, we consider the optimized policy after one iteration in this cost comparison. From Table 4-4 and Figure 4-21 we conclude that our policy outperforms the linear policy where the nonlinearities of the plant are most apparent. Around $x_1 = 0$, where the plant behaves approximately linear, the linear policy outperforms our policy.

We hypothesize that the deterioration of performance of the policy after consecutive iterations is caused by poor approximation of the value function [103]. To prevent this from happening, one would have to deploy a sufficiently large value network, such that is able to adequately approximate the value function. However, in our approach, the topology of the value network can not be arbitrarily large, as this would make the verification step intractable.

**Table 4-4:** Cost comparison between the initial policy and the optimized policy for 24 trajectories of Dubins car, the Duffing oscillator and the inverted pendulum.

| | Dubins car | | | Duffing oscillator | | | Inverted pendulum | | |
|---|---|---|---|---|---|---|---|---|---|
| Policy | Initial | Optimized | Difference | Initial | Optimized | Difference | Initial | Optimized | Difference |
| Trajectory 1 | 848.59 | 94.13 | 754.45 | 34.01 | 37.71 | -3.70 | 1468.61 | 550.55 | 918.05 |
| 2 | 38.88 | 38.06 | 0.81 | 9.34 | 12.48 | -3.14 | 325.71 | 324.71 | 0.99 |
| 3 | 30.09 | 27.51 | 2.57 | 12.86 | 14.74 | -1.88 | 164.47 | 216.71 | -52.23 |
| 4 | 59.36 | 51.50 | 7.86 | 24.52 | 23.05 | 1.46 | 1465.82 | 1001.44 | 464.38 |
| 5 | 144.59 | 117.85 | 26.74 | 52.37 | 44.77 | 7.59 | 3339.86 | 1355.34 | 1984.51 |
| 6 | 93.31 | 57.16 | 36.15 | 30.41 | 33.19 | -2.78 | 1853.72 | 717.50 | 1136.21 |
| 7 | 12.86 | 13.38 | -0.52 | 4.26 | 5.82 | -1.56 | 537.42 | 474.46 | 62.95 |
| 8 | 9.15 | 9.28 | -0.13 | 3.22 | 3.80 | -0.57 | 41.44 | 57.20 | -15.75 |
| 9 | 33.02 | 33.14 | -0.12 | 12.02 | 11.18 | 0.84 | 1109.38 | 819.05 | 290.32 |
| 10 | 112.91 | 94.45 | 18.45 | 39.66 | 37.24 | 2.42 | 2788.21 | 1113.64 | 1674.56 |
| 11 | 70.87 | 63.50 | 7.36 | 32.32 | 32.75 | -0.43 | 2293.24 | 905.06 | 1388.18 |
| 12 | 11.78 | 12.35 | -0.56 | 5.21 | 4.96 | 0.24 | 799.39 | 640.63 | 158.76 |
| 13 | 11.78 | 12.09 | -0.31 | 5.21 | 5.13 | 0.08 | 799.39 | 645.90 | 153.49 |
| 14 | 70.87 | 58.82 | 12.04 | 32.32 | 33.47 | -1.14 | 2293.24 | 900.73 | 1392.50 |
| 15 | 112.91 | 101.27 | 11.63 | 39.66 | 36.24 | 3.42 | 2788.21 | 1120.42 | 1667.78 |
| 16 | 33.02 | 33.83 | -0.81 | 12.02 | 9.98 | 2.03 | 1109.38 | 814.41 | 294.97 |
| 17 | 9.15 | 9.54 | -0.39 | 3.22 | 3.78 | -0.55 | 41.44 | 57.96 | -16.52 |
| 18 | 12.86 | 12.91 | -0.05 | 4.26 | 5.63 | -1.37 | 537.42 | 480.54 | 56.87 |
| 19 | 93.31 | 55.06 | 38.25 | 30.41 | 33.75 | -3.34 | 1853.72 | 712.35 | 1141.36 |
| 20 | 144.59 | 124.22 | 20.37 | 52.37 | 43.51 | 8.85 | 3339.86 | 1366.66 | 1973.20 |
| 21 | 59.36 | 52.44 | 6.91 | 24.52 | 20.44 | 4.08 | 1465.82 | 997.55 | 468.27 |
| 22 | 30.09 | 27.85 | 2.24 | 12.86 | 14.64 | -1.78 | 164.47 | 218.12 | -53.65 |
| 23 | 38.88 | 37.53 | 1.34 | 9.34 | 12.57 | -3.22 | 325.71 | 330.82 | -5.11 |
| 24 | 848.59 | 91.28 | 757.30 | 34.01 | 38.35 | -4.34 | 1468.61 | 547.23 | 921.37 |

### 4-4-5 Conclusion

In this section, we subjected the method to nonlinear systems. In the Dubins car and inverted pendulum examples, the optimized policies were able to outperform the linear policies when the effects of the nonlinearities of the plants are apparent. The optimized policy for the Duffing oscillator performed, on average, similar to the linear policy.

For the inverted pendulum, the optimized policy after one iteration performed better than the policies obtained after multiple iterations. We hypothesize that this deterioration is caused by poor approximation of the value function.

Lastly, the synthesis times of FNPI were compared to synthesis times of methods in related literature. It was concluded that the synthesis times of FNPI are up to three orders of magnitude higher. Both training and verification take longer in our method, which is primarily caused by the deployment of larger networks and that the fact that we use the value network as a Lyapunov candidate.

## 4-5 Summary & discussion

In this section the proposed methodology of Chapter 3 and the results presented in Chapter 4 are discussed and evaluated. The goal of this thesis was to design an automatic controller synthesis method for continuous-time nonlinear systems that produces (near-)optimal controllers with stability and safety guarantees. To this end, we developed a certificate-based synthesis method based on Approximate Dynamic Programming (ADP), using NNs for the policy and the value function, and Satisfiability Modulo Theory (SMT) for formal verification.

To verify the correctness of the synthesis method, we tested the method on linear systems with quadratic cost. There exists an analytical solution of the value function and policy, provided by LQR theory. First, the policy is initialized at a stabilizing solution. Subsequently, seven iterations of the method occurred, after which the method converged to the analytical solution of both the policy and the value function.

To get an adequate approximation of the value function, the value network should be sufficiently large. However, the verification time scales with the size of the policy and value networks. Therefore, in Section 4-2, we ran the procedure for policy and value networks with various topologies. From Figure 4-3 we can conclude verification times for two-dimensional linear systems with single-layer networks scale linearly with the amount of nodes in the layer. Networks with multiple hidden layers, however, quickly become intractable to use as the amount of nodes per layer increases. We therefore propose to use single-layer networks with relatively large amount of nodes in the hidden layer. However, as detailed in Figure 4-6, the verification time for three-dimensional systems becomes intractable, even for single-layer NNs.

In Section 4-3, we show the effectiveness of the method on linear systems with input and state constraints. Input constraints are enforced by appending the policy network with an extra layer with a bounded, Lipschitz continuous activation function. This way, the inputs are kept within the bounds of the activation function of the last layer. Figure 4-7 shows a truncated policy and Figure 4-8 shows its corresponding value function, which is verified as a Lyapunov function by dReal.

To enforce state constraints, we propose to use cost function shaping. By adding an extra penalty to certain regions of the state space, the method will improve the policy such that those regions will be avoided. We provide two examples. In the first example, we put a penalty on states where $|x_2| > 3$. After three iterations of the method, the resulting policy effectively avoids high velocities. Figure 4-13 shows the level set of the value function of the nonlinear policy and of the initial policy. The level set of the optimized policy is significantly larger. The value function can therefore be used as a barrier function, to render the level set invariant and thus safe.

In the second example, we penalise states in a circular region of the state space around $[x_1, x_2]^\top = [2.4, -1.7]^\top$. After two iterations of the method, the policy is able to circumvent the penalised region, as depicted in Figure 4-12. In this experiment, the value function can not be used as a barrier function to certify the policy as safe, as the value function has no level set that solely excludes the penalised region from the rest of the region of interest. More constructive methods to design the cost function are needed in this regard.

In Section 4-4, we subject the method to three nonlinear systems. To test the ability of the method to synthesize optimal controllers, we compared the additive cost of several trajectories when the system was subject to the optimized policies and to the initial linear policies. The optimized policy for the Duffing oscillator performed, on average, similar to the linear policy. In the Dubins car and inverted pendulum examples, the optimized policies were able to outperform the linear policies when the trajectories traversed parts of the state-space where the effects of the nonlinearities of the system are significant. In the parts of the state-space where the systems are approximately linear, the optimized policies performed slightly worse. For the inverted pendulum, the optimized policy after one iteration performed better than the policies obtained after multiple iterations. We hypothesize that this deterioration is caused by poor approximation of the value function.

Lastly, we compared the synthesis times of FNPI to formal synthesis methods of related literature. We conclude that the synthesis times of FNPI are up to three orders of magnitude higher, which is primarily caused by the use of larger networks, which in turn causes the training and verification durations to be longer. This observation highlights the principal bottleneck of our method. With the selection of the network topologies, a trade-off is made between their approximation capabilities and the time it takes to verify the networks. In our method, however, we cannot compromise on the approximation capabilities of the networks. The inverted pendulum example illustrates that poor approximation of the value function causes the deterioration of performance of the policies, ultimately leading to worse performing policies than the initial linear ones.

Chapter 5

# Conclusion

In this chapter, the research goal and problem statements are revisited and answered, final conclusions are drawn, and recommendations for future research are given.

## 5-1 Conclusion

The goal of this thesis was to design an automatic controller synthesis procedure for continuous-time nonlinear systems that produces (near-)optimal controllers with stability and safety guarantees that does not require computationally expensive online optimization.

To this end, we developed a certificate-based controller synthesis method called Formal Neural Policy Iteration (FNPI). FNPI synthesizes closed-form controllers offline, which therefore do not suffer from computationally expensive online optimization. To check whether the proposed method achieves the remaining requirements of the research goal, we defined three sub-problems, which will be discussed hereafter.

1. **Optimal stability:** *Provided with a cost function, synthesize a controller with formal guarantee of stability, that minimizes the the infinite horizon additive cost.*

   To synthesize controllers with optimality and stability guarantees, FNPI uses Approximate Dynamic Programming (ADP) techniques in combination with Lyapunov theory. Specifically, we utilize techniques from Generalized Policy Iteration (GPI) to obtain a controller and value function in the form of Neural Networks (NNs). We exploit the fact that the value function obtained by GPI can, under certain conditions, be used as a Lyapunov function. To assure the controllers are stabilizing, we deploy Satisfiability Modulo Theory (SMT) solver dReal throughout the procedure to formally verify that the value function is a Lyapunov function. In Section 4-1, we show that FNPI effectively combines optimality and stability.

2. **Safe and optimal stability:** *Provided with a cost function, synthesize a controller with formal guarantee of stability and safety, that minimizes the the infinite horizon additive cost.*

In addition to stability and safety, it was investigated in Section 4-3 whether the method is able to synthesize controllers with additional safety guarantee. To this end, we conducted two experiments, in which we used *cost function shaping*. We designed the cost function in such a way that the value network serves as a Lyapunov function for stability guarantee and as a barrier function for safety guarantee. The method was successful for the first experiment, but required careful manual design of the cost function. The method failed to synthesize an adequate certificate function in the second experiment. We therefore conclude that, for cost function shaping, more constructive methods are needed to design the cost function, or other methods need to be considered to combine stability, safety and optimality.

3. **Cost minimization:** *The synthesized controller should attain a lower infinite horizon additive cost than a linear controller, where the linear controller is a Linear-Quadratic Regulator (LQR) controller for the linearized system.*

   From the comparisons in Section 4-4-4, it can be concluded that controllers synthesized by our method are able to outperform the linear controllers. The optimized policies showed significantly lower infinite horizon additive cost when the trajectories traversed parts of the state-space where the effects of the nonlinearities of the system are most apparent. In the parts of the state-space where the systems are approximately linear, the optimized policies performed slightly worse.

We can conclude that the proposed method is able to solve the three sub-problems. FNPI is able to produce optimal controllers with stability guarantee for continuous-time nonlinear systems, which can outperform linear controllers in terms of cost minimization. The method is also capable of synthesizing optimal controllers with safety and stability guarantee for specific problems. However, more constructive methods are needed for the method to be able to synthesize safe controllers for any problem.

In terms of synthesis times, SMT-based verification showed to be the principle bottleneck of the method. This bottleneck prohibits the verification of systems with more than two states. Furthermore, with the selection of the network topologies, a trade-off is made between their approximation capabilities and the time it takes to verify the networks. In our method, however, we cannot compromise on the approximation capabilities of the networks, as poor approximation of the value function causes the deterioration of performance of the policies.

Despite these bottlenecks, FNPI has proven itself to be a flexible framework that adds optimality to the certificate-based formal synthesis paradigm. We are confident the bottlenecks are surmountable and that the method is able to improve with certain adjustments, which we will discuss in the next section.

## 5-2 Future work

In this thesis a novel controller synthesis procedure was presented. In theory, the method is able to produce optimal controllers with stability and safety guarantee for continuous-time nonlinear systems. In practice, however, the method runs into a couple of bottlenecks. Potential research directions to overcome these bottlenecks and directions to further extend the method will be discussed in this section.

**Faster verification** With the selection of the network topologies, a trade-off is made between their approximation capabilities and the time it takes to verify the networks. In our method, however, we cannot compromise on the approximation capabilities of the networks, as poor approximation of the value function causes the deterioration of performance of the policies. Potential ways to improve the verification times are:

1. Decoupling of the Lyapunov function from the value network. One could use a separate, smaller NN for verifiction, which only has to adhere to the Lyapunov conditions. Although this solution might help, it will not be scalable, as this method scales poorly with higher system dimensions, as shown in Section 4-2-4.
2. Sparse networks. By encouraging the weight parameters of the NNs to become exactly zero, the symbolic representation of these networks will become smaller, therefore reducing the verification time. Potential methods to achieve this are $L_0$ regularization [104] or with parameter selection through gate variables [105].
3. Use linear approximations of the activation functions for verification, as done in [57].
4. Use a different function approximation method, e.g. StaF kernels [106] used by [107,108], which allows for efficient local approximation.
5. Utilize the sampling based verification method from [97], which is parallelizable and therefore potentially faster. With a sampling based verification method, the verification step could potentially be merged with the neural policy evaluation and improvement steps, making the synthesis method more concise.

**Barrier functions** In this thesis, we attempted to train a single NN to serve as a Lyapunov and barrier function, by utilizing cost function shaping. The experiments show modest results in this regard. The synthesis of barrier functions could be further improved by the following suggestions:

1. Design a cost function shaping algorithm which transforms the value network into a barrier function. Note that the cost function need not be fixed in advance, but could be adjusted during the synthesis procedure.
2. Decouple the barrier function from the value network. One could look into the density function, which is considered the dual of the value function, and is used for safety [109, 110].

**Initial stabilizing policy** Another potential bottleneck is the necessity of a stabilizing policy at initialization. In this work, we have examined systems that feature a linear stabilizing policy. Yet, for many systems this is not the case [111]. We therefore exhibit related synthesis methods in literature, which do not require an initial stabilizing policy.

1. Approximate value iteration for continuous-time systems [112]. Value iteration is closely related to policy iteration. Value iteration does not feature an explicit policy, however.
2. Actor-critic reinforcement learning [71], utilizes a similar policy-value function structure, but assumes no knowledge of the model and no initial policy.

**Other systems and control specifications** In this thesis, we considered continuous-time nonlinear systems. The systems are deterministic and full knowledge of the dynamics is assumed. Furthermore, we considered the control specifications stability, safety and optimality. This work could be extended, by considering:

1. Synthesis of robust controllers for models with disturbances. This involves a variation of the Hamilton-Jacobi-Bellman (HJB) equation, called the Hamilton-Jacobi-Isaac equation [113] [114].
2. Hybrid systems, which exhibit both continuous and discrete dynamic behavior. An introduction to control and verification of hybrid systems is given in [115].
3. More complex control specifications in the form of Linear Temporal Logic (LTL) or Signal Temporal Logic (STL) [116] [108].

# Appendix A

# Prototype tool FNPI

The methodology discussed in this thesis has been implemented in a prototype tool called Formal Neural Policy Iteration (FNPI). FNPI is available at: github.com/JonathanKLSCH/formal-neural-controller-synthesis.

FNPI utilizes five main classes: `CostFunction`, `Controller`, `Verifier`, `ControlSystem` and `NeuralNetwork`. How these classes interact is depicted in Figure A-1. We will discuss those classes in detail hereafter.

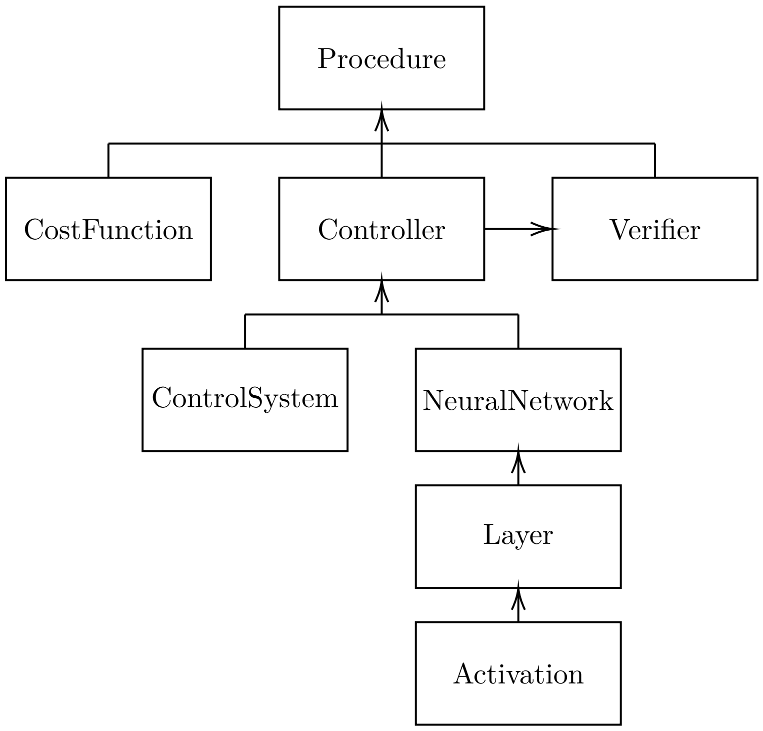


**Figure A-1:** Class diagram of FNPI

**ControlSystem** In the `ControlSystem` class, the dynamics of the control system are described. The class inherits from the `GenericControlSystem` class, which contains a couple of helper functions, such as `euler_step()`, for the visualisation of trajectories, and `lqr_controller(Q,R)`, for initializing the policy network. The `ControlSystem` class acts as a bridge between PyTorch and dReal, as it can represent the control system in both a `Tensor` format to work with PyTorch, and in a `Symbolic` format to work with dReal.

```python
class DubinsCar(GenericControlSystem):
    def __init__(self):
        super().__init__(num_states=2,num_inputs=1)
        self.v = 1

    def f(self, x, u):
        f = [self.v * torch.sin(x[:,:,1]),
             -u[:,:,0]]
        f = torch.stack((f[0].flatten(), f[1].flatten()), 1).unsqueeze(2)
        return f

    def symbolic(self, x1, x2, u):
        expression =    [self.v * dreal.sin(x2),
                         -u]
        return expression
```

**NeuralNetwork** The `NeuralNetwork` class is initialized with a learning rate, the amount of nodes per layer and the activation functions.

```python
policy_network_lr = 1e-3
policy_network_dimensions = [2,200,1]
policy_network_activations = [SiLU(), Identity()]

policy_network = NeuralNetwork(policy_network_lr, policy_network_dimensions,
                                       policy_network_activations)
```

Similar to the `ControlSystems` class, the `NeuralNetwork` class also acts as a bridge between PyTorch and dReal, as it can represent the control system in both a `Tensor` and `Symbolic` format. To realize this, the activation functions are implemented in an `Activation` class, which has a `forward(x)` method for `Tensor` operations and a `symbolic(x)` method for verification.

```python
class SiLU(torch.nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, x):
        return x * torch.sigmoid(x)

    def symbolic(self, x):
        return x/(1+dreal.exp(-x))
```

**Controller** The `Controller` class is initialized with a `ControlSystem` and two neural network topologies. The class will make two instances of the `NeuralNetwork` class, one for the policy and one for the value function. This class is able to calculate the Lie derivative of the value network with respect to the system dynamic in `Tensor` format, which allows for fast evaluation of the value loss and policy loss, and therefore speeds up training significantly.

**Verifier** The `Verifier` class receives the `ControlSystem` and the value and policy network from the `Controller` class and transforms them into symbolic expressions. The class has a `verify_continuous_lyapunov_conditions(area)` method, which returns a counterexample if the conditions are violated in `area`, or returns `None` if the conditions are satisfied.

**CostFunction** The cost function class has an array called `total_cost`, to which cost function elements in the form of lambda functions can be added. The `calculate_total_cost()` function returns the cost in the form of a `Tensor` for each `(X,U)` pair.

```python
class CostFunction(object):
    def __init__(self):
        self.total_cost = []

    def add_cost_term(self, term):
        self.total_cost.append(term)

    def calculate_total_cost(self, X, U):
        cost = 0

        for cost_element in self.total_cost:
            cost += cost_element(X,U)

        return cost
```

# Appendix B

# Verification times

Table B-1 depicts the verification times with various value network topologies for a two-dimensional system. The policy is linear and the gains are initialized at the optimal solution, where after the value network was trained until convergence.

Table B-2 depicts the verification times for various policy network topologies. The experiment is conducted on a two-dimensional linear system with a $2-20-20-1$ value network with quadratic activation function for the hidden layers and linear activation for the final layer.

**Table B-1:** Verification times of neural certificates with various topologies for a linear system with two states. × denotes the network topology was too small to represent a Lyapunov function for the system. − denotes the verification time limit was reached.

| | Quadratic | | | Tanh | | | SiLU | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1-layer | 2-layer | 3-layer | 1-layer | 2-layer | 3-layer | 1-layer | 2-layer | 3-layer |
| nodes = 4 | 0.006 | 0.056 | 0.916 | × | 0.195 | 4.065 | 0.063 | 8.334 | 105.490 |
| 5 | 0.008 | 0.116 | 2.830 | 0.042 | 0.412 | 82.072 | 0.116 | 8.336 | 256.395 |
| 6 | 0.009 | 0.215 | 6.578 | 0.094 | 0.633 | 43.994 | 0.129 | 8.516 | 606.956 |
| 7 | 0.008 | 1.002 | 16.218 | 0.088 | 1.529 | 42.011 | 0.179 | 11.345 | 339.309 |
| 8 | 0.012 | 0.326 | 38.442 | 0.062 | 4.466 | 71.379 | 0.149 | 25.203 | 1032.875 |
| 9 | 0.010 | 0.715 | 51.140 | 0.221 | 2.610 | 125.102 | 0.180 | 25.914 | 732.219 |
| 10 | 0.012 | 0.774 | 8.199 | 0.630 | 11.139 | 196.791 | 0.200 | 27.158 | 527.415 |
| 15 | 0.019 | 1.463 | 43.520 | 0.254 | 14.278 | - | 0.377 | 40.777 | - |
| 20 | 0.029 | 3.008 | 253.507 | 0.540 | 57.067 | - | 0.324 | 62.564 | - |
| 25 | 0.181 | 4.814 | 302.560 | 0.744 | 29.120 | - | 0.367 | 161.854 | - |
| 30 | 0.035 | 8.479 | 761.696 | 0.860 | 44.864 | - | 0.664 | 667.855 | - |
| 35 | 0.040 | 16.253 | 1558.707 | 0.745 | 72.062 | - | 0.610 | 1109.383 | - |
| 40 | 0.048 | 17.617 | - | 1.953 | 13.620 | - | 1.264 | - | - |
| 45 | 0.055 | 31.556 | - | 1.484 | 56.292 | - | 0.821 | - | - |
| 50 | 0.073 | 44.852 | - | 1.117 | 29.934 | - | 0.902 | - | - |
| 60 | 0.081 | 58.925 | - | 1.057 | 364.927 | - | 1.165 | - | - |
| 70 | 0.099 | 89.865 | - | 2.012 | 413.041 | - | 1.525 | - | - |
| 80 | 0.105 | 135.183 | - | 1.377 | - | - | 1.496 | - | - |
| 90 | 0.134 | 162.840 | - | 2.770 | - | - | 2.706 | - | - |
| 100 | 0.144 | 184.360 | - | 3.114 | - | - | 2.253 | - | - |

**Table B-2:** Verification times of neural certificates with neural policies with various topologies for a linear system with two states. − denotes the verification time limit was reached.

| | Quadratic | | | Tanh | | | SiLU | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1-layer | 2-layer | 3-layer | 1-layer | 2-layer | 3-layer | 1-layer | 2-layer | 3-layer |
| nodes = 4 | 2.043 | 2.803 | 5.880 | 0.980 | 1.042 | 1.038 | 2.396 | 1.473 | 2.373 |
| 5 | 2.280 | 2.275 | 11.325 | 1.084 | 0.956 | 1.041 | 2.057 | 1.258 | 2.502 |
| 6 | 2.452 | 4.760 | 81.591 | 1.060 | 0.982 | 1.160 | 2.032 | 1.346 | 3.633 |
| 7 | 2.282 | 8.525 | 79.681 | 0.982 | 0.957 | 1.960 | 2.125 | 1.400 | 4.092 |
| 8 | 2.113 | 3.590 | 37.303 | 0.944 | 0.969 | 1.447 | 2.131 | 1.463 | 7.238 |
| 9 | 2.522 | 4.677 | 137.549 | 1.011 | 1.052 | 1.506 | 2.000 | 1.899 | 7.797 |
| 10 | 2.371 | 5.617 | 44.916 | 0.954 | 1.081 | 1.958 | 2.289 | 1.634 | 9.986 |
| 15 | 2.317 | 6.971 | 191.253 | 1.007 | 1.195 | 4.845 | 2.060 | 2.198 | 29.537 |
| 20 | 2.505 | 14.339 | - | 1.076 | 1.271 | 10.809 | 2.023 | 2.788 | 107.276 |
| 25 | 3.024 | 16.218 | - | 1.054 | 1.599 | 30.792 | 2.365 | 4.247 | 226.941 |
| 30 | 2.999 | 23.232 | - | 1.022 | 2.314 | 57.838 | 2.619 | 4.548 | 479.072 |
| 35 | 2.571 | 21.540 | - | 0.969 | 2.862 | 99.822 | 2.112 | 6.148 | 1045.039 |
| 40 | 2.466 | 28.470 | - | 1.044 | 3.304 | 164.976 | 2.124 | 7.939 | - |
| 45 | 2.711 | 68.623 | - | 1.046 | 3.881 | 346.513 | 2.276 | 11.391 | - |
| 50 | 3.084 | 100.835 | - | 1.043 | 4.684 | 571.932 | 2.354 | 13.630 | - |
| 60 | 3.073 | 47.873 | - | 0.992 | 6.415 | 1122.305 | 2.257 | 21.910 | - |
| 70 | 2.806 | 55.478 | - | 0.991 | 10.276 | - | 2.300 | 29.333 | - |
| 80 | 2.716 | 95.914 | - | 1.070 | 13.748 | - | 2.467 | 49.343 | - |
| 90 | 3.045 | 368.531 | - | 1.002 | 18.175 | - | 2.541 | 68.768 | - |
| 100 | 3.205 | 378.751 | - | 1.095 | 26.680 | - | 2.340 | 80.386 | - |

# Bibliography

[1] C. Belta and S. Sadraddini, “Formal methods for control synthesis: An optimization perspective,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, pp. 115–140, 2019.

[2] R. E. Kalman *et al.*, “Contributions to the theory of optimal control,” *Bol. soc. mat. mexicana*, vol. 5, no. 2, pp. 102–119, 1960.

[3] R. Sepulchre, M. Jankovic, and P. V. Kokotovic, *Constructive nonlinear control.* Springer Science & Business Media, 2012.

[4] J. H. Lee, “Model predictive control: Review of the three decades of development,” *International Journal of Control, Automation and Systems*, vol. 9, no. 3, pp. 415–424, 2011.

[5] D. P. Bertsekas, “Dynamic programming and suboptimal control: A survey from ADP to MPC,” *European Journal of Control*, vol. 11, no. 4-5, pp. 310–334, 2005.

[6] L. Buşoniu, T. de Bruin, D. Tolić, J. Kober, and I. Palunko, “Reinforcement learning for control: Performance, stability, and deep approximators,” *Annual Reviews in Control*, vol. 46, pp. 8–28, 2018.

[7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv preprint arXiv:1606.01540*, 2016.

[8] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, “Learning quadrupedal locomotion over challenging terrain,” *Science robotics*, vol. 5, no. 47, 2020.

[9] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, “Control of a quadrotor with reinforcement learning,” *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[10] M. Reble, *Model predictive control for nonlinear continuous-time systems with and without time-delays.* Logos Verlag Berlin GmbH, 2013.

[11] A. Girard, "Controller synthesis for safety and reachability via approximate bisimulation," *Automatica*, vol. 48, no. 5, pp. 947–953, 2012.

[12] M. Zamani, G. Pola, M. Mazo, and P. Tabuada, "Symbolic models for nonlinear control systems without stability assumptions," *IEEE Transactions on Automatic Control*, vol. 57, no. 7, pp. 1804–1809, 2011.

[13] P. Tabuada, *Verification and control of hybrid systems: a symbolic approach.* Springer Science & Business Media, 2009.

[14] K. Y. Rozier, "Linear temporal logic symbolic model checking," *Computer Science Review*, vol. 5, no. 2, pp. 163–203, 2011.

[15] M. Rungger and M. Zamani, "SCOTS: A tool for the synthesis of symbolic controllers," in *Proceedings of the 19th international conference on hybrid systems: Computation and control*, pp. 99–104, 2016.

[16] M. Mazo, A. Davitian, and P. Tabuada, "PESSOA: A tool for embedded controller synthesis," in *International Conference on Computer Aided Verification*, pp. 566–569, Springer, 2010.

[17] S. Mouelhi, A. Girard, and G. Gössler, "CoSyMa: a tool for controller synthesis using multi-scale abstractions," in *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pp. 83–88, 2013.

[18] K. Hsu, R. Majumdar, K. Mallik, and A.-K. Schmuck, "Multi-layered abstraction-based controller synthesis for continuous-time systems," in *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*, pp. 120–129, 2018.

[19] J. Cámara, A. Girard, and G. Gössler, "Synthesis of switching controllers using approximately bisimilar multiscale abstractions," in *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pp. 191–200, 2011.

[20] Y. Tazaki and J. Imura, "Bisimilar finite abstractions of interconnected systems," in *International Workshop on Hybrid Systems: Computation and Control*, pp. 514–527, Springer, 2008.

[21] E. S. Kim, M. Arcak, and M. Zamani, "Constructing control system abstractions from modular components," in *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*, pp. 137–146, 2018.

[22] M. Zamani, A. Abate, and A. Girard, "Symbolic models for stochastic switched systems: A discretization and a discretization-free approach," *Automatica*, vol. 55, pp. 183–196, 2015.

[23] V. Staudt, "Compact representation of mathematical functions for control applications by piecewise linear approximations," *Electrical Engineering*, vol. 81, no. 3, pp. 129–134, 1998.

[24] I. S. Zapreev, C. Verdier, and M. Mazo Jr, "Optimal symbolic controllers determinization for BDD storage," *IFAC-PapersOnLine*, vol. 51, no. 16, pp. 1–6, 2018.

[25] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pp. 152–166, Springer, 2004.

[26] C. A. Floudas, *Nonlinear and mixed-integer optimization: fundamentals and applications*. Oxford University Press, 1995.

[27] A. Richards and J. P. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, vol. 3, pp. 1936–1941, IEEE, 2002.

[28] Y. Wang, B. De Schutter, T. J. van den Boom, and B. Ning, "Optimal trajectory planning for trains–a pseudospectral method and a mixed integer linear programming approach," *Transportation Research Part C: Emerging Technologies*, vol. 29, pp. 97–114, 2013.

[29] Z. Liu, J. Dai, B. Wu, and H. Lin, "Communication-aware motion planning for multi-agent systems from signal temporal logic specifications," in *2017 American Control Conference (ACC)*, pp. 2516–2521, IEEE, 2017.

[30] S. Karaman and E. Frazzoli, "Linear temporal logic vehicle routing with applications to multi-UAV mission planning," *International Journal of Robust and Nonlinear Control*, vol. 21, no. 12, pp. 1372–1395, 2011.

[31] I. Haghighi, S. Sadraddini, and C. Belta, "Robotic swarm control from spatio-temporal specifications," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 5708–5713, IEEE, 2016.

[32] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *2012 IEEE international conference on robotics and automation*, pp. 477–483, IEEE, 2012.

[33] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *53rd IEEE Conference on Decision and Control*, pp. 81–87, IEEE, 2014.

[34] H. Abbas, A. Winn, G. Fainekos, and A. A. Julius, "Functional gradient descent method for metric temporal logic specifications," in *2014 American Control Conference*, pp. 2312–2317, IEEE, 2014.

[35] Y. V. Pant, H. Abbas, and R. Mangharam, "Smooth operator: Control using the smooth robustness of temporal logic," in *2017 IEEE Conference on Control Technology and Applications (CCTA)*, pp. 1235–1240, IEEE, 2017.

[36] Y. Shoukry, P. Nuzzo, I. Saha, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada, "Scalable lazy SMT-based motion planning," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 6683–6688, IEEE, 2016.

[37] D. Monniaux, "A survey of satisfiability modulo theory," in *International Workshop on Computer Algebra in Scientific Computing*, pp. 401–425, Springer, 2016.

[38] Y. Shoukry, P. Nuzzo, A. Balkan, I. Saha, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada, "Linear temporal logic motion planning for teams of underactuated robots using satisfiability modulo convex programming," in *2017 IEEE 56th annual conference on decision and control (CDC)*, pp. 1132–1137, IEEE, 2017.

[39] S. S. Farahani, V. Raman, and R. M. Murray, "Robust model predictive control for signal temporal logic synthesis," *IFAC-PapersOnLine*, vol. 48, no. 27, pp. 323–328, 2015.

[40] S. Sadraddini and C. Belta, "Robust temporal logic model predictive control," in *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 772–779, IEEE, 2015.

[41] A. M. Lyapunov, "The general problem of the stability of motion," *International journal of control*, vol. 55, no. 3, pp. 531–534, 1992.

[42] F. Blanchini, "Set invariance in control," *Automatica*, vol. 35, no. 11, pp. 1747–1767, 1999.

[43] S. Prajna and A. Jadbabaie, "Safety verification of hybrid systems using barrier certificates," in *International Workshop on Hybrid Systems: Computation and Control*, pp. 477–492, Springer, 2004.

[44] S. Prajna, "Barrier certificates for nonlinear model validation," *Automatica*, vol. 42, no. 1, pp. 117–126, 2006.

[45] S. Prajna, A. Papachristodoulou, and F. Wu, "Nonlinear control synthesis by sum of squares optimization: A lyapunov-based approach," in *2004 5th Asian Control Conference (IEEE Cat. No. 04EX904)*, vol. 1, pp. 157–165, IEEE, 2004.

[46] A. A. Ahmadi, M. Krstic, and P. A. Parrilo, "A globally asymptotically stable polynomial vector field with no polynomial Lyapunov function," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pp. 7579–7580, IEEE, 2011.

[47] J. Lofberg, "Pre-and post-processing sum-of-squares programs in practice," *IEEE transactions on automatic control*, vol. 54, no. 5, pp. 1007–1011, 2009.

[48] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *The International Journal of Robotics Research*, vol. 36, no. 8, pp. 947–982, 2017.

[49] A. A. Ahmadi, G. Hall, A. Papachristodoulou, J. Saunderson, and Y. Zheng, "Improving efficiency and scalability of sum of squares optimization: Recent advances and limitations," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 453–462, IEEE, 2017.

[50] S. Shen and R. Tedrake, "Scalable sampling-based sum-of-squares programs for systems verification,"

[51] A. Solar-Lezama, L. Tancau, R. Bodik, S. Seshia, and V. Saraswat, "Combinatorial sketching for finite programs," in *Proc. of the 12th Int.Conf. on Architectural Support for Programming Languages and Operating Systems*, p. 404–415, ACM, 2006.

[52] A. Abate, D. Ahmed, M. Giacobbe, and A. Peruffo, "Automated formal synthesis of Lyapunov neural networks," *arXiv preprint arXiv:2003.08910*, 2020.

[53] H. Dai, B. Landry, M. Pavone, and R. Tedrake, "Counter-example guided synthesis of neural network Lyapunov functions for piecewise linear systems," in *2020 59th IEEE Conference on Decision and Control (CDC)*, pp. 1274–1281, IEEE, 2020.

[54] A. Peruffo, D. Ahmed, and A. Abate, "Automated formal synthesis of neural barrier certificates for dynamical models," *arXiv preprint arXiv:2007.03251*, 2020.

[55] Y.-C. Chang, N. Roohi, and S. Gao, "Neural Lyapunov control," in *Advances in Neural Information Processing Systems*, pp. 3240–3249, 2019.

[56] W. Jin, Z. Wang, Z. Yang, and S. Mou, "Neural certificates for safe control policies," *arXiv preprint arXiv:2006.08465*, 2020.

[57] H. Zhao, X. Zeng, T. Chen, Z. Liu, and J. Woodcock, "Learning safe neural network controllers with barrier certificates," in *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, pp. 177–185, Springer, 2020.

[58] H. Ravanbakhsh and S. Sankaranarayanan, "Counter-example guided synthesis of control Lyapunov functions for switched systems," in *2015 54th IEEE conference on decision and control (CDC)*, pp. 4232–4239, IEEE, 2015.

[59] C. F. Verdier and M. Mazo, "Formal synthesis of analytic controllers for sampled-data systems via genetic programming," in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 4896–4901, IEEE, 2018.

[60] R. V. Gamkrelidze, "Discovery of the maximum principle," *Journal of dynamical and control systems*, vol. 5, no. 4, pp. 437–451, 1999.

[61] M. Barbero-Liñán and M. C. Muñoz-Lecanda, "Geometric approach to Pontryagin's maximum principle," *Acta applicandae mathematicae*, vol. 108, no. 2, pp. 429–485, 2009.

[62] A. Filippov, "On certain questions in the theory of optimal control," *Journal of the Society for Industrial and Applied Mathematics, Series A: Control*, vol. 1, no. 1, pp. 76–84, 1962.

[63] S. J. Qin and T. A. Badgwell, "An overview of industrial model predictive control technology," in *AIche symposium series*, vol. 93, pp. 232–256, New York, NY: American Institute of Chemical Engineers, 1971-c2002., 1997.

[64] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov, "An integrated system for real-time model predictive control of humanoid robots," in *2013 13th IEEE-RAS International conference on humanoid robots (Humanoids)*, pp. 292–299, IEEE, 2013.

[65] M. Zanon, J. V. Frasch, M. Vukov, S. Sager, and M. Diehl, "Model predictive control of autonomous vehicles," in *Optimization and optimal control in automotive systems*, pp. 41–57, Springer, 2014.

[66] D. P. Bertsekas, *Dynamic programming and optimal control*, vol. 1. Athena scientific Belmont, MA, 1995.

[67] J. Duan, S. E. Li, Z. Liu, M. Bujarbaruah, and B. Cheng, "Generalized policy iteration for optimal control in continuous time," *arXiv preprint arXiv:1909.05402*, 2019.

[68] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," in *Advances in neural information processing systems*, pp. 908–918, 2017.

[69] H. Ferreira, P. Rocha, and R. Sales, "Nonlinear H-infinity control and the Hamilton-Jacobi-Isaac equation," in *17th World Congress, The international Federation of Automatic Control, Seoul, Korea*, pp. 188–193, 2009.

[70] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[71] I. Grondman, L. Busoniu, G. A. 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.

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

[73] H. K. Khalil and J. W. Grizzle, *Nonlinear systems*, vol. 3. Prentice hall Upper Saddle River, NJ, 2002.

[74] M. Z. Romdlony and B. Jayawardhana, "Stabilization with guaranteed safety using control Lyapunov–barrier function," *Automatica*, vol. 66, pp. 39–47, 2016.

[75] K. P. Tee, S. S. Ge, and E. H. Tay, "Barrier Lyapunov functions for the control of output-constrained nonlinear systems," *Automatica*, vol. 45, no. 4, pp. 918–927, 2009.

[76] K. Hornik, M. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural networks*, vol. 3, no. 5, pp. 551–560, 1990.

[77] A. Kratsios, "The universal approximation property," *Annals of Mathematics and Artificial Intelligence*, 2020.

[78] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *Neural Networks*, vol. 107, pp. 3–11, 2018.

[79] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[80] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *Siam Review*, vol. 60, no. 2, pp. 223–311, 2018.

[81] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *Journal of machine learning research*, vol. 18, 2018.

[82] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[83] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.

[84] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, "Adversarial attacks on neural network policies," *arXiv preprint arXiv:1702.02284*, 2017.

[85] C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. J. Kochenderfer, "Algorithms for verifying deep neural networks," *arXiv preprint arXiv:1903.06758*, 2019.

[86] J. P. Vielma, "Mixed integer linear programming formulation techniques," *Siam Review*, vol. 57, no. 1, pp. 3–57, 2015.

[87] M. Akintunde, A. Lomuscio, L. Maganti, and E. Pirovano, "Reachability analysis for neural agent-environment systems," in *Sixteenth International Conference on Principles of Knowledge Representation and Reasoning*, 2018.

[88] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," *arXiv preprint arXiv:1711.07356*, 2017.

[89] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi, "Measuring neural net robustness with constraints," in *Advances in neural information processing systems*, pp. 2613–2621, 2016.

[90] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*, pp. 97–117, Springer, 2017.

[91] S. Dutta, X. Chen, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Sherlock - A tool for verification of neural network feedback systems: demo abstract," in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 262–263, 2019.

[92] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli, "A dual approach to scalable verification of deep networks.," in *UAI*, vol. 1, p. 2, 2018.

[93] E. Wong and Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *International Conference on Machine Learning*, pp. 5286–5295, 2018.

[94] A. Raghunathan, J. Steinhardt, and P. Liang, "Certified defenses against adversarial examples," *arXiv preprint arXiv:1801.09344*, 2018.

[95] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340, Springer, 2008.

[96] S. Gao, S. Kong, and E. M. Clarke, “dReal: An SMT solver for nonlinear theories over the reals,” in *International conference on automated deduction*, pp. 208–214, Springer, 2013.

[97] R. Bobiti and M. Lazar, “Sampling-based verification of Lyapunov’s inequality for piecewise continuous nonlinear systems,” *arXiv preprint arXiv:1609.00302*, 2016.

[98] M. Abu-Khalaf and F. L. Lewis, “Nearly optimal control laws for nonlinear systems with saturating actuators using a neural network HJB approach,” *Automatica*, vol. 41, no. 5, pp. 779–791, 2005.

[99] P. Rutquist, T. Wik, and C. Breitholtz, “Solving the Hamilton-Jacobi-Bellman equation for a stochastic system with state constraints,” in *53rd IEEE Conference on Decision and Control*, pp. 1840–1845, IEEE, 2014.

[100] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “PyTorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[101] J. V. Deshmukh, J. P. Kapinski, T. Yamaguchi, and D. Prokhorov, “Learning deep neural network controllers for dynamical systems with safety guarantees,” in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, IEEE, 2019.

[102] I. Kovacic and M. J. Brennan, *The Duffing equation: nonlinear oscillators and their behaviour.* John Wiley & Sons, 2011.

[103] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, *et al.*, “Policy gradient methods for reinforcement learning with function approximation.,” in *NIPs*, vol. 99, pp. 1057–1063, Citeseer, 1999.

[104] C. Louizos, M. Welling, and D. P. Kingma, “Learning sparse neural networks through $L_0$ regularization,” *arXiv preprint arXiv:1712.01312*, 2017.

[105] S. Srinivas, A. Subramanya, and R. Venkatesh Babu, “Training sparse neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 138–145, 2017.

[106] J. A. Rosenfeld, R. Kamalapurkar, and W. E. Dixon, “The state following approximation method,” *IEEE transactions on neural networks and learning systems*, vol. 30, no. 6, pp. 1716–1730, 2018.

[107] M. H. Cohen and C. Belta, “Approximate optimal control for safety-critical systems with control barrier functions,” in *2020 59th IEEE Conference on Decision and Control (CDC)*, pp. 2062–2067, IEEE, 2020.

[108] M. Cohen and C. Belta, “Model-based reinforcement learning for approximate optimal control with temporal logic specifications,” *arXiv preprint arXiv:2101.07156*, 2021.

[109] Y. Chen and A. D. Ames, “Duality between density function and value function with applications in constrained optimal control and markov decision process,” *arXiv preprint arXiv:1902.09583*, 2019.

[110] Y. Chen, M. Ahmadi, and A. D. Ames, “Optimal safe controller synthesis: A density function approach,” in *2020 American Control Conference (ACC)*, pp. 5407–5412, IEEE, 2020.

[111] C. I. Byrnes and A. Isidori, “New results and examples in nonlinear feedback stabilization,” *Systems & Control Letters*, vol. 12, no. 5, pp. 437–442, 1989.

[112] T. Bian and Z.-P. Jiang, “Value iteration and adaptive dynamic programming for data-driven adaptive optimal control design,” *Automatica*, vol. 71, pp. 348–360, 2016.

[113] T. Dierks and S. Jagannathan, “Optimal control of affine nonlinear continuous-time systems using an online Hamilton-Jacobi-Isaacs formulation,” in *49th IEEE Conference on Decision and Control (CDC)*, pp. 3048–3053, IEEE, 2010.

[114] J. Li, S. E. Li, Y. Guan, J. Duan, W. Li, and Y. Yin, “Ternary policy iteration algorithm for nonlinear robust control,” *arXiv preprint arXiv:2007.06810*, 2020.

[115] H. Lin and P. J. Antsaklis, “Hybrid dynamical systems: An introduction to control and verification,” *Foundations and trends in systems and control*, vol. 1, no. 1, pp. 1–172, 2014.

[116] C. F. Verdier, N. Kochdumper, M. Althoff, and M. Mazo Jr, “Formal synthesis of closed-form sampled-data controllers for nonlinear continuous-time systems under stl specifications,” *arXiv preprint arXiv:2006.04260*, 2020.

# Glossary

## List of Acronyms

| | |
|---|---|
| **ADP** | Approximate Dynamic Programming |
| **CARE** | Continuous Algebraic Riccati Equation |
| **CEGIS** | Counterexample-Guided Inductive Synthesis |
| **FNPI** | Formal Neural Policy Iteration |
| **GPI** | Generalized Policy Iteration |
| **HJB** | Hamilton-Jacobi-Bellman |
| **LP** | Linear Programming |
| **LTL** | Linear Temporal Logic |
| **LQR** | Linear-Quadratic Regulator |
| **MILP** | Mixed-Integer Linear Programming |
| **MPC** | Model Predictive Control |
| **NN** | Neural Network |
| **PMP** | Pontryagin's Maximum Principle |
| **ReLU** | Rectified Linear Unit |
| **RL** | Reinforcement Learning |
| **SDP** | Semidefinite Programming |
| **SiLU** | Sigmoid Linear Unit |
| **SMT** | Satisfiability Modulo Theory |
| **SOS** | Sum-of-Squares |
| **STL** | Signal Temporal Logic |