Learning Stabilizing Controllers for High-dimensional Unknown Systems and Networked Dynamical Systems

by

Songyuan Zhang

B.E., Tsinghua University, 2021

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN AERONAUTICS AND ASTRONAUTICS

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2024

Authored by:	Songyuan Zhang Department of Aeronautics and Astronautics January 19, 2024
Certified by:	Chuchu Fan Assistant Professor of Aeronautics and Astronautics, Thesis Supervisor
Accepted by:	Jonathan P. How R. C. Maclaurin Professor of Aeronautics and Astronautics Chair, Graduate Program Committee

Learning Stabilizing Controllers for High-dimensional Unknown Systems and Networked Dynamical Systems

by

Songyuan Zhang

Submitted to the Department of Aeronautics and Astronautics
on January 19, 2024 in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN AERONAUTICS AND ASTRONAUTICS

ABSTRACT

Designing stabilizing controllers is a fundamental challenge in autonomous systems, particularly for high-dimensional, nonlinear systems that cannot be accurately modeled using differential equations because of the scalability and model transparency, and large-scale networked dynamical systems because of scalability and generalizability. To address the challenge, we develop (1) A Lyapunov-based guided exploration framework to learn stabilizing controllers for high-dimensional unknown systems; (2) A compositional neural certificate based on ISS (Input-to-State Stability) Lyapunov functions for finding decentralized stabilizing controllers in large-scale networked dynamical systems. Comprehensive experiments have shown that the proposed methods outperform the prior work in the case of stability, especially in high-dimensional unknown systems and large-scale networked systems.

Thesis supervisor: Chuchu Fan
Title: Assistant Professor of Aeronautics and Astronautics

Acknowledgments

I would like to start by extending my heartfelt gratitude to my mentor, Prof. Chuchu Fan, for her invaluable guidance and support throughout my master's journey. Chuchu ushered me into the realm of safe autonomy, demonstrating unwavering belief in my capabilities and affording me the freedom to explore various research directions until I discovered the one that resonated with my passion. Her patience was truly commendable, providing extensive advice on various facets, especially in paper writing and presentation preparation. Even in instances where I found myself making recurring mistakes, Chuchu remained patient and offered her assistance. I cannot envision any success in my research endeavors without her indispensable guidance.

I express my sincere appreciation to my exceptional collaborators, Oswin So, Dr. Kunal Garg, Charles Dawson, Prof. Guannan Qu, and Yumeng Xiu, as well as to all my remarkable labmates. Our collaborative efforts involved countless brainstorming sessions, joint debugging endeavors to tackle challenging issues, and shared celebrations upon achieving success. The invaluable contributions of my collaborators and labmates have been instrumental in overcoming numerous hurdles during my research journey. Without their support, my research would have been stuck in many areas.

I also owe many thanks to my friends all over the world for their encouragement and support during my most difficult times. Those late-night chats, trips, hot pots, board games, karaoke, postcards, and sports will always be one of the greatest treasures in my life.

I could never be grateful enough to my family for their unwavering support and boundless

love. Despite the physical distance that separates us, their complete trust and steadfast encouragement in every decision I make have been my pillars of strength. Their support has eliminated any fear of failure in my pursuit, and I wholeheartedly acknowledge that I would not be the person I am today without them.

I am greatly indebted to my girlfriend Tong Xiao. She made the courageous decision to leave China and came to be with me in the US. Thanks to her presence, I never feel alone during my time here, maintaining a positive outlook each day. Her influence has provided clarity regarding the direction of my life, and because of her, I am empowered to be the person I aspire to become.

Last but not least, I would like to acknowledge the funding support from the Singapore Defence Science and Technology Agency, and the National Science Foundation. Their support made my research possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

While autonomous systems continue to assume increasingly significant roles in our daily lives, the crucial attribute of stability often poses a challenge to their application. For instance, the presence of a fragile controller in an autonomous vehicle can lead to severe traffic accidents, and an unstable controller in a nursing robot may pose risks of harm to individuals. The development of stable controllers emerges as a fundamental challenge in the realm of autonomous systems, and this challenge intensifies when dealing with real-world systems. Real-world systems, exemplified by the F-16 fighter jet with its high-dimensional state spaces [1], and networked dynamics such as those in power grids [2] and transportation networks [3], further complicates the design of controllers. In this chapter, we elucidate the problems addressed in this thesis, highlight our contributions, and present an outline of the thesis.

1.1 Stablilizing Controller Design for Real-world Systems

We consider the problem of designing stable and safe controllers for real-world systems, particularly focusing on two challenges: high-dimensional unknown systems, and networked dynamics. The discussion of the problems is mainly from [4], [5].

1.1.1 High-dimensional Unknown Systems

The first challenge addressed in this thesis involves the formulation of stabilizing controllers for high-dimensional unknown systems, for example, fixed-wing aircraft [1], legged robots [6], and autonomous vehicles [7]. Lyapunov-based control design assumes a crucial role in shaping and assessing the stability characteristics of autonomous systems [8]. In recent years, diverse approaches have emerged to devise Lyapunov functions and control Lyapunov functions (CLFs) for systems of varying complexity, spanning both optimization-based and learning-based methodologies [9], [10]. However, existing methods face two significant challenges: *scalability*, referring to their suitability for high-dimensional systems, and *model transparency*, indicating the need for understanding the system dynamics.

Traditional optimization-based control design involves finding controllers and Lyapunov functions by solving a sequence of semi-definite programming (SDP) problems [8], [11], [12]. However, the scalability of such methods to high-dimensional systems is hindered by the exponential growth of the number of decision variables with system dimension [13] as well as numerical problems [14], such as strict feasibility and numerical reliability issues. Although recent advancements have produced scalable SDP solvers [15], they depend on assumptions such as the sparsity of the decision matrix. In recent years, Neural network (NN)-based representations of Lyapunov functions have gained popularity [16]–[19] and can, to some extent, alleviate dimensionality limitations when finding a CLF. However, due to their reliance on state-space sampling, learning techniques still face exponential growth in sample complexity for high-dimensional systems.

Additionally, most of the optimization-based and learning-based methods require system dynamics to be known as ordinary differential equations (ODEs), limiting their applicability and practicality for real-world systems. For instance, the F-16 fighter jet model [1] investigated in this thesis is represented as a combination of look-up tables, block diagrams, and C programs. Accurately describing the complex behavior of such a system using an ODE is

highly challenging, if not impossible. For systems with unknown dynamics, previous works have attempted to first use system identification, (e.g., learn the ODE model with NNs), then find a controller with a CLF [20], [21]. However, using a single NN to fit the dynamics of the entire state space of a high-dimensional system requires a vast number of training samples to cover the entire state space, and these surrogate NN models can exhibit substantial prediction errors in sparsely sampled regions of the state-action space.

To address the challenge, we are motivated by the fact that for high-dimensional systems, collecting data across the entire state space is both infeasible and unnecessary, as only a small subset of the state space is reachable for the agent starting from a set of initial conditions. Therefore, obtaining a model for the entire state space is excessive and potentially impossible. Instead, we can learn a model valid only in the reachable subspace and update the controller according to guidance, e.g., the learned CLF, to continuously expand the subspace towards the goal. Constructing such a subspace is non-trivial, so we assume access to some imperfect and potentially unstable demonstrations as initial guidance for reachable states. Starting from these demonstrations, our objective is to update the controller, guide exploration of necessary regions in the state space, and ultimately stabilize the system at the goal.

1.1.2 Networked Systems

Large-scale networked dynamical systems are another type of system that plays an important role across a wide spectrum of real-world applications, including power grids [2], vehicle platoons [22], drone swarms [23], transportation networks [3], etc. The control, and in particular stabilization, of such networked systems has long been recognized as a challenging problem as the dimension of the state and input spaces of such networked systems is usually very high, and existing methods often suffer from the "curse-of-dimensionality" [24].

Classical approaches to stabilizing dynamical systems include LQR (Linear Quadratic Regulator) for linear systems [25], [26]. For nonlinear systems, certificates like Lyapunov functions can guide the search for a stabilizing controller and certify the stability of the

closed-loop system [27]. However, certificates are usually constructed on a case-by-case basis. While there exist approaches like SOS (Sum-Of-Squares) that can construct certificates for general classes of nonlinear systems [8], they are not scalable to large-scale networked dynamical systems, since the number of polynomial coefficients in an SOS program grows exponentially w.r.t. the dimension of system [8].

A recent line of work parameterizes control certificates (e.g. Lyapunov functions, barrier functions) and controllers as neural networks (NNs) and learns them jointly from data [28]–[30]. They have been successfully applied to nonlinear systems and achieved good performance on complex control tasks [31], [32] thanks to the representation power of NNs. However, the existing works mainly focus on single-agent systems with relatively small state space ($\leq$ 10 dimensions) or multi-agent systems without coupled dynamics [28], [33]. Applying these approaches to large-scale networked systems can be challenging due to the exponential growth of the sample complexity and the hardness of training NNs with large input spaces. Despite the challenge, many networked dynamics often contain sparse network structures that can be exploited to help training, and the question we try to answer in this work is: *can we exploit network structure to learn neural certificates and stabilize large scale networked systems in a scalable and effective manner?*

To answer the question, we view the large networked dynamical system as a group of smaller subsystems interconnected through a graph. Instead of learning a single certificate for the entire system, we find a decentralized ISS (Input-to-State Stability) Lyapunov function [34]–[36] and a decentralized controller for each subsystem. Although ISS Lyapunov theory has been known for a long time, it is not straightforward to adapt them as neural certificates for the stabilization of large networked systems due to the following reasons: 1) Existing ISS Lyapunov theory requires checking a condition involving global information of the networked system [35] and as a result, is not entirely decentralized; 2) Existing ISS Lyapunov theory requires finding different ISS Lyapunov functions for each subsystem and therefore is computationally expensive for systems with many subsystems; 3) Each subsys-

tem, as well as the corresponding ISS Lyapunov function, are intertwined with neighboring subsystems and therefore cannot be learned straightforwardly as Lyapunov functions for a single system such as that in [28]. In this work, we develop a framework for learning ISS Lyapunov functions for networked dynamical systems while solving the aforementioned issues, and learning a decentralized stabilizing controller at the same time.

1.2 Contributions

To address the challenge of developing stabilizing controllers for real-world systems caused by high-dimensional unknown dynamics and networked dynamics, we propose two frameworks, named **LY**apunov-**G**uided **E**xploration (LYGE) and **Neur**al **ISS** Lyapunov functions (NeurISS), respectively.

With high-dimensional unknown dynamics, LYGE [4] jointly learns the system dynamics in the reachable subspace, a controller, and a CLF to guide the exploration of the controller in the system. We iteratively learn the dynamics in the reachable subspace using past experience, update the CLF and the controller, and perform exploration to expand the subspace toward the goal. Upon convergence, we obtain a stabilizing controller for the high-dimensional unknown system. The main **contributions** of this work are:

1. We propose a novel framework, LYGE, to learn stabilizing controllers for high-dimensional unknown systems. Guided by a learned CLF, LYGE explores only the *useful* subspace, thus addressing the scalability and model transparency problems.

2. We show that the proposed algorithm learns a stabilizing controller.

3. We conduct experiments on benchmarks including Inverted Pendulum, Cart Pole, Cart II Pole [37], Neural Lander [38], and the F-16 model [1] with two tasks. Our results show that our learned controller outperforms other reinforcement learning (RL), imitation learning (IL), and neural-certificate-based algorithms in terms of stabilizing the systems while reducing the number of samples by 68% to 95%.

With networked dynamics, We proposed NeurISSn [5] which makes the following contributions:

1. We show that the ISS Lyapunov functions only need to satisfy a *local* condition involving local information from neighboring subsystems to collectively constitute a compositional certificate to certify the stability of the entire dynamical system.

2. We prove that under certain conditions, the compositional certificate for a small networked system can be generalized to be used on a more extensive system that has a similar structure without re-training, which improves the scalability of the proposed approach as one can reduce a large training task to a smaller training task with a smaller network size.

3. We extend the notion of the ISS Lyapunov function to robust ISS Lyapunov function for control-affine systems so that similar subsystems that have different parameters can share the same ISS Lyapunov functions, which not only reduces the number of ISS Lyapunov functions we need to learn for large-scale networked systems but also improves the robustness of the learned results against model uncertainties.

4. We develop a novel approach to encode the ISS logic condition that intertwines neighboring subsystems into the training loss function.

We demonstrate NeurISS using three examples - Power systems, Platoon, and Drone formation control, and show that NeurISS can find certifiably stable controllers for networks of size up to 100 subsystems. Compared with centralized neural certificate approaches, NeurISS reaches similar results in small-scale systems and can generalize to large-scale systems that centralized approaches cannot scale up to. Compared with LQR, NeurISS can deal with strong coupled networked systems like microgrids and reach smaller tracking errors on both small and large-scale systems. Compared with RL (PPO, LYPPO, MAPPO), our algorithm achieves similar or smaller tracking errors in small systems, and can hugely reduce the tracking errors in large systems (up to 75%).

1.3 Thesis Structure

In Chapter 2, we discuss the background and related works of developing stabilizing controllers for the two types of systems studied in this thesis. In Chapter 3 and Chapter 4, we introduce LYGE and NeurISS, respectively, based on their original papers [4] and [5], including the type of system they are designed for, the algorithm structure, and experimental results. Finally, in Chapter 5, we summarize the thesis and discuss possible directions for future work.

Chapter 2

Related Work

In this chapter, we discuss the related works on designing stabilizing controllers for high-dimensional unknown systems and networked dynamical systems, including Lypuanov-based control design, neural certificates, reinforcement learning, and imitation learning. The discussion is based on [4], [5].

2.1 High-dimensional Unknown Systems

Control Lyapunov Functions Our work builds on the widely used Lyapunov theory for designing stabilizing controllers. Classical CLF-based controllers primarily rely on hand-crafted CLFs [39], [40] or Sum-of-Squares (SoS)-based SDPs [8], [11], [12]. However, these approaches require known dynamics and struggle to generalize to high-dimensional systems due to the exponential growth of decision variables and numerical issues [13], [14]. To alleviate these limitations, recent work utilizes neural networks (NNs) to learn Lyapunov functions [31], [41]–[43] and stabilizing controllers [5], [16], [19], [44]–[46]. Most of these works sample states in the entire state space and apply supervised learning to enforce the CLF conditions. They either assume knowledge of the dynamics or attempt to fit the entire state space's dynamics [20], [21], making it difficult to generalize to high-dimensional real-world scenarios where the number of required samples grows exponentially with the dimensions.

In contrast, our algorithm can handle unknown dynamics and does not suffer from the curse of dimensionality caused by randomly sampling states in the entire state space.

Reinforcement Learning (RL) and Optimal Control RL and optimal control have demonstrated strong capabilities on problems without knowledge of the dynamics, particularly in hybrid systems [47]–[50]. However, they struggle to provide results of the closed-loop system's stability. Additionally, hand-crafted reward functions and sample inefficiency impede the generalization of RL algorithms to complex environments. Recent works in the learning for control domain aim to solve this problem by incorporating certificate functions into the RL process [17], [18], [30], [51]–[54]. Nevertheless, they suffer from limitations such as handcrafted certificates [51] and balancing CLF-related losses with RL losses [17], [18]. Unlike these approaches, our algorithm learns the CLF from scratch without prior knowledge of the dynamics or CLF candidates and provides a structured way to design loss functions rather than relying on reward functions. Furthermore, we can demonstrate the stability of the closed-loop system using the learned CLF.

Imitation Learning Imitation learning (IL) is another common tool for such problems. However, classical IL algorithms like behavioral cloning (BC) [55]–[58], inverse reinforcement learning (IRL) [59]–[61], and adversarial learning [62]–[65] primarily focus on recovering the exact policy of the demonstrations, which may result in poor performance when given imperfect demonstrations. A recent line of work on learning from suboptimal demonstrations offers a possible route to learn a policy that outperforms the demonstrations. However, they either require various types of manual supervision, such as rankings [66], [67], weights of demonstrations [68], [69], or have additional requirements concerning the environment [70], [71], the demonstrations [72], [73], or the training process [74]. Moreover, none of them can provide results about the stability of the learned policy. Another line of work learns certificates from demonstrations [75]–[78], but they need additional assumptions such as known dynamics, perfect demonstrations, or the ability to query the demonstrator. In

contrast, our approach leverages the CLF as natural guidance for the exploration process and does not require any additional supervision or assumptions about either the demonstrations or the environment to learn a stabilizing policy.

2.2 Networked Dynamical Systems

Neural Certificates Mostly related is the line of work on learning neural certificates. This line of work focuses on searching for a controller together with a certificate that guarantees the soundness of the controller. Such neural certificates include Lyapunov-like functions for stability guarantees [28], [29], [31], [32], [41], [43], [79], barrier functions for safety guarantees [29], [33], [80]–[82], contraction metrics for tracking guarantees [83], [84], etc. Through learning proof of the correctness of the controllers, these approaches address the concerns about the safety, stability, and reliability of the controllers on a large variety of tasks, including precision quadrotor flight through turbulence [83], walking under model uncertainties [40], tracking with high-dimensional dynamics [84], and safe decentralized control of multi-agent systems [33], [85]. Compared to these works, we learn ISS Lyapunov functions, which are decentralized and scalable to large-scale networked systems.

ISS Lyapunov Function The concept of ISS and ISS Lyapunov function is long established in control theory [34]. ISS Lyapunov function for networked dynamical systems was proposed in [86] for a two-subsystem case and generalized in [35], [87] for multiple subsystems (cf [36] for a review). Compared to these works, our paper builds upon the ISS Lyapunov concept to learn neural certificates for networked systems.

Chapter 3

Learning to Stabilize High-dimensional Unknown Systems

In this chapter, we proposed **LY**apunov-**G**uided **E**xploration (LYGE) to learn stabilizing controllers for high-dimensional unknown systems. The results of this chapter are based on [4].

3.1 Problem Setting and Preliminaries

We consider a discrete-time single-agent unknown dynamical system

$$x(t+1) = h(x(t), u(t)), \tag{3.1}$$

where $x(t) \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$ represents the state at time step t, $u(t) \in \mathcal{U} \subseteq \mathbb{R}^{n_u}$ denotes the control input at time step t, and $h : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is the *unknown* dynamics. We assume the state space $\mathcal{X}$ to be compact and h is Lipschitz continuous in both (x, u) with constant $L_h > 0$ following [51]. Our objective is to find a control policy $u(\cdot) = \pi(x(\cdot))$, where $\pi : \mathcal{X} \to \mathcal{U}$, such that from initial states $x(0) \in \mathcal{X}_0$, under the policy π, the closed-loop system asymptotically stabilizes at a goal $x^{\text{goal}} \in \mathcal{X}$. In other words, $\forall x(t)$ starting from $x(0) \in \mathcal{X}_0$ and satisfying

Equation (3.1) with $u(t) = \pi(x(t))$, we have $\lim_{t\to\infty} \left\|x(t) - x^{\text{goal}}\right\| = 0$.

We assume that we are given a set of N_{D^0} demonstration transitions

$$\mathcal{D}^0 = \{(x_i(t), u_i(t), x_i(t+1))\}_{i=1}^{N_{D^0}} \tag{3.2}$$

generated by a demonstrator policy π_d starting from states $x_i(0) \in \mathcal{X}_0$. In contrast to the assumption of stabilizing demonstrators in classical IL works [62], our demonstrations may not be generated by a stabilizing controller. We also assume an online setting, where we assume access to environmental interaction during training.

Lyapunov theory is widely used to prove the stability of control systems, and CLFs offer further guidance for controller synthesis by defining a set of stabilizing control inputs at a given point in the state space. Following [88], we provide the definition of CLF for discrete system (3.1).

Definition 1. *Consider the system (3.1) and a goal point x^{goal}, and let $\mathcal{G} \subseteq \mathcal{X}$ be a subset of the state space such that $x^{\text{goal}} \in \mathcal{G}$. A function $V : \mathcal{G} \to \mathbb{R}$ is called a CLF on $\mathcal{G}$ if there exists functions $\underline{\alpha}, \bar{\alpha} \in \mathcal{K}_\infty$[1] and a constant $\lambda \in (0, 1)$ such that the following hold:*

$$\underline{\alpha}(\|x - x^{\text{goal}}\|) \leq V(x) \leq \bar{\alpha}(\|x - x^{\text{goal}}\|) \tag{3.3a}$$

$$\inf_{u\in\mathcal{U}} V\left(h(x,u)\right) \leq \lambda V(x) \tag{3.3b}$$

The set of input $\mathcal{K}(x) = \{u \in \mathcal{U} \mid V\left(h(x,u)\right) \leq \lambda V(x)\}$ is called *stablizing control inputs*. It is a standard result that if $\mathcal{G}$ is forward invariant[2] and the goal point $x^{\text{goal}} \in \mathcal{G} \subseteq \mathcal{X}$, then starting from initial set $\mathcal{X}_0 \subseteq \mathcal{G}$, any control input $u \in \mathcal{K}$ will make the closed-loop system asymptotically stable at x^{goal} [88]. The formal proposition is given below:

Proposition 1. *Given a set $\mathcal{G} \subset \mathcal{X}$ such that $x^{\text{goal}} \in \mathcal{G}$. Suppose there exists a CLF V on*

[1] A function $\alpha : [0, +\infty) \to [0, +\infty)$ is said to be class-$\mathcal{K}_\infty$ if α is continuous, strictly increasing with $\alpha(0) = 0$, and $\lim_{s\to+\infty} \alpha(s) = +\infty$.

[2] A set $\mathcal{G}$ is forward invariant if $x(0) \in \mathcal{G} \implies x(t) \in \mathcal{G}$ for all $t > 0$.

$\mathcal{G} \subseteq \mathcal{X}$ *with a constant* $\lambda \in (0,1)$. *If* $\mathcal{G}$ *is forward invariant*[3], *then* x^{goal} *is asymptotically stable for the closed-loop system under* $u \in \mathcal{K}(x) = \{u \mid V(h(x,u)) \leq \lambda V(x)\}$ *starting from initial set* $\mathcal{X}_0 \subseteq \mathcal{G}$.

Proof. The proof follows from Definition 2.18 and Theorem 2.19 in [88]. Condition (3.3a) is the same as condition (i) in [88], and for condition 2, let $u^*(x) = \inf_u V(h(x,u))$, from Condition (3.3b), we have:

$$V(h(x,u^*(x))) \leq \lambda V(x), \quad \forall x \in \mathcal{G}. \tag{3.4}$$

Therefore,

$$V(h(x,u^*(x))) \leq V(x) - (1-\lambda)V(x) \leq V(x) - (1-\lambda)\underline{\alpha}(\|x - x^{\text{goal}}\|), \quad \forall x \in \mathcal{G}, \tag{3.5}$$

where the second equation follows condition (3.3a). Define $g(x) = (1-\lambda)x$. Since $\lambda \in (0,1)$, it follows that $g(x) \in \mathcal{K}_\infty$. Let $\alpha_V(x - x^{\text{goal}}) = g \circ \underline{\alpha}(\|x - x^{\text{goal}}\|)$, we have $\alpha_V \in \mathcal{K}_\infty$ and

$$V(h(x,u^*(x))) \leq V(x) - \alpha_V(\|x - x^{\text{goal}}\|), \quad \forall x \in \mathcal{G}, \tag{3.6}$$

which aligns with condition (ii) in [88]. Note that $\mathcal{G}$ is assumed forward invariant. Therefore, following Theorem 2.19 in [88], the system is asymptotically stable starting from $\mathcal{X}_0 \in \mathcal{G}$. □

3.2 LYGE Algorithm

Notation Let τ be the current iteration step in our algorithm. A *dataset* $\mathcal{D}^\tau$ is a set of N_{D^τ} transitions collected by the current iteration step. Recall that $\mathcal{D}^0$ is the set of given demonstrations. Let $\mathcal{D}_x^\tau \subset \mathcal{X}$ be the projection of $\mathcal{D}^\tau$ on the first state component of $\mathcal{D}^\tau$ defined as $\mathcal{D}_x^\tau = \{x \mid (x,\cdot,\cdot) \in \mathcal{D}^\tau\}$. A *trusted tunnel* $\mathcal{H}^\tau$ is defined as the set of states at

[3] A set $\mathcal{G}$ is forward invariant if $x(0) \in \mathcal{G} \implies x(t) \in \mathcal{G}$ for all $t > 0$.

most $\gamma > 0$ distance away from the dataset $\mathcal{D}_x^\tau$, i.e., $\mathcal{H}^\tau = \{x \mid \exists x_i \in \mathcal{D}_x^\tau, \|x - x_i\| \leq \gamma\}$.

We first outline our algorithm **LY**apunov-**G**uided **E**xploration (LYGE), which learns to stabilize high-dimensional unknown systems, then provide a step-by-step explanation. Given a dataset of imperfect demonstrations $\mathcal{D}^0$ (which may not contain x^{goal}), we firstly employ IL to learn an initial controller π_{init} (which may be an unstable controller). Then, during each iteration τ, we learn a CLF V_θ^τ and the corresponding controller π_ϕ^τ using samples from $\mathcal{D}_x^\tau$, and use the learned controller π_ϕ^τ to generate closed-loop trajectories as additional data added to $\mathcal{D}^\tau$ to obtain $\mathcal{D}^{\tau+1}$. At each iteration, the learned CLF V_θ^τ ensures that the newly collected trajectories get closer to x^{goal}, as each V_θ^τ is designed to reach the global minimum at x^{goal}. In this way, the trusted tunnel $\mathcal{H}^\tau$ grows and includes states closer and closer to x^{goal} with more iterations. Upon convergence, LYGE returns a stabilizing controller π^* that can be trusted within the converged trusted tunnel $\mathcal{H}^*$, where $\mathcal{H}^*$ contains all trajectories starting from $\mathcal{X}_0$.

Learning from Demonstrations At iteration 0, our approach starts with learning an initial policy from imperfect and potentially unstable demonstrations $\mathcal{D}^0$. We use existing IL methods (e.g., BC [56]) to learn the initial policy $\pi_{\text{init}}(\cdot)$. Since the IL algorithms directly recover the behavior of the demonstrations, the initial policy could lead to unstable behaviors.

Learning Local Dynamics At iteration τ, we learn a discrete NN approximation of the dynamics $\hat{h}_\psi^\tau(x, u) : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ parameterized by ψ using the transition dataset $\mathcal{D}^\tau$. We train $\hat{h}_\psi^\tau$ by minimizing the mean square error loss from transitions sampled from the dataset $\mathcal{D}^\tau$ using Adam [89]. Let $\omega > 0$ be the maximum error of the learned dynamics on the training data: $\|\hat{h}_\psi^\tau(x_i(t), u_i(t)) - x_i(t+1)\| \leq \omega$ for all $(x_i(t), u_i(t), x_i(t+1)) \in \mathcal{D}^\tau$.

Learning CLF and Controller After learning the local dynamics, we jointly learn the CLF and the control policy. Let the learned CLF in τ-th iteration be

$$V_\theta^\tau(x) = x^\top S^\top S x + p_{\text{NN}}(x)^\top p_{\text{NN}}(x), \tag{3.7}$$

where $S \in \mathbb{R}^{n_x \times n_x}$ is a matrix of parameters, $p_{\text{NN}} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$ denotes an NN, and θ encompasses all parameters including S and the ones in p_{NN}. Clearly, $V_\theta^\tau(x)$ is positive by construction. The first term in V_θ^τ models a quadratic function, which is commonly used to construct Lyapunov functions for linear systems. Here we use it to introduce a quadratic prior to the learned CLF. The second term models the CLF's non-quadratic residue. We also parameterize the controller with an NN, i.e., $\pi_\phi^\tau : \mathcal{X} \to \mathcal{U}$ with parameters ϕ. The learned controller aims to direct system trajectories toward the goal, guided by the learned CLF. Additionally, since the learned dynamics $\hat{h}_\psi^\tau$ may be invalid outside the trusted tunnel $\mathcal{H}^\tau$, π_ϕ^τ should not drive the system too far from $\mathcal{H}^\tau$ within the simulation horizon. This can be achieved by penalizing the distance of the control inputs in consecutive iterations. Overall, V_θ^τ and π_ϕ^τ can be synthesized by solving the following optimization problem:

$$\min_{\phi} \quad \mathbb{E}_{x \in \mathcal{H}^\tau} \|\pi_\phi^\tau(x) - \pi_\phi^{\tau-1}(x)\| \tag{3.8a}$$

$$\text{s.t.} \quad V_\theta^\tau(x^{\text{goal}}) = 0, \qquad V_\theta^\tau(x) > 0, \quad \forall x \in \mathcal{X} \setminus x^{\text{goal}} \tag{3.8b}$$

$$V_\theta^\tau\left(\hat{h}_\psi^\tau(x, \pi_\phi^\tau(x))\right) \leq \lambda V_\theta^\tau(x), \quad \forall x \in \mathcal{H}^\tau \tag{3.8c}$$

where $\lambda \in (0, 1)$. We approximate the solution of problem (3.8) by self-supervised learning with loss $\mathcal{L}^\tau = \mathcal{L}_{\text{CLF}}^\tau + \eta_{\text{ctrl}} \mathcal{L}_{\text{ctrl}}^\tau$, where $\mathcal{L}_{\text{CLF}}^\tau$ and $\mathcal{L}_{\text{ctrl}}^\tau$ correspond to the constraints and the objective of Problem (3.8), respectively, $\eta_{\text{ctrl}} > 0$ is a hyper-parameter that balances the

weights of two losses, and

$$\mathcal{L}^{\tau}_{\text{CLF}} = V^{\tau}_{\theta}(x^{\text{goal}})^2 + \frac{1}{N}\sum_{x\in\mathcal{X}\setminus x^{\text{goal}}}[\nu - V^{\tau}_{\theta}(x)]^{+} + \frac{\eta_{\text{pos}}}{N}\sum_{x\in\mathcal{D}^{\tau}_{x}}\left[\epsilon + V^{\tau}_{\theta}\left(\hat{h}^{\tau}_{\psi}(x,\pi^{\tau}_{\phi}(x))\right) - \lambda V^{\tau}_{\theta}(x)\right]^{+} \tag{3.9}$$

$$\mathcal{L}^{\tau}_{\text{ctrl}} = \frac{1}{N}\sum_{x\in\mathcal{D}^{\tau}_{x}}\|\pi^{\tau}_{\phi}(x) - \pi^{\tau-1}_{\phi}(x)\|^2, \tag{3.10}$$

where $[\cdot]^{+} = \max(\cdot, 0)$, $\eta_{\text{pos}} > 0$ is a hyper-parameter that balances each term in the loss, and $\epsilon > 0$ is a hyper-parameter that ensures that the learned CLF V^{τ}_{θ} satisfies condition (3.3b) even if the learned dynamics $\hat{h}^{\tau}_{\psi}$ has errors. In practice, it is hard to enforce $V^{\tau}_{\theta}(x^{\text{goal}})$ to be exactly 0 using gradient-based methods. Therefore, in loss (3.9) and (3.10), we instead train the NNs to make $V^{\tau}_{\theta}(x^{\text{goal}}) < \nu$, and $V^{\tau}_{\theta}(x) \geq \nu$, for all $x \in \mathcal{X} \setminus x^{\text{goal}}$, where ν is a small positive number.

Exploration After learning $\hat{h}^{\tau}_{\psi}$, V^{τ}_{θ}, and π^{τ}_{ϕ} in the τ-th iteration, we use the current controller π^{τ}_{ϕ} starting from $x_0 \in \mathcal{H}^{\tau}$ to collect N_D more transitions $\Delta\mathcal{D}^{\tau} = \{(x_i(t), \pi^{\tau}_{\phi}(x_i(t)), x_i(t+1))\}_{i=1}^{N_D}$ and augment the dataset $\mathcal{D}^{\tau+1} = \mathcal{D}^{\tau} \cup \Delta\mathcal{D}^{\tau}$. During the exploration, the controller drives the system to states with lower CLF values. Consequently, by collecting more trajectories with the learned policy π^{τ}_{ϕ}, we expand the trusted tunnel $\mathcal{H}^{\tau}$ to states with lower CLF values. As the global minimum of the CLF is at x^{goal}, the system trajectories used to construct the trusted tunnel keep getting closer to the goal in each iteration.

We illustrate the LYGE process in an inverted pendulum environment in Figure 3.1. In Figure 3.1a, since the demonstrations are imperfect, our initial controller π_{init} cannot reach the goal. Figure 3.1b and Figure 3.1c demonstrate that after several iterations, the trusted tunnel $\mathcal{H}^{\tau}$ (the region around the light orange dots $\mathcal{D}^{\tau}_{x}$) is expanded towards the goal, and the closed-loop system progressively approaches the goal. Upon convergence (Figure 3.1d), our controller stabilizes the system at the goal. The workflow of LYGE is summarized in Algorithm 1.

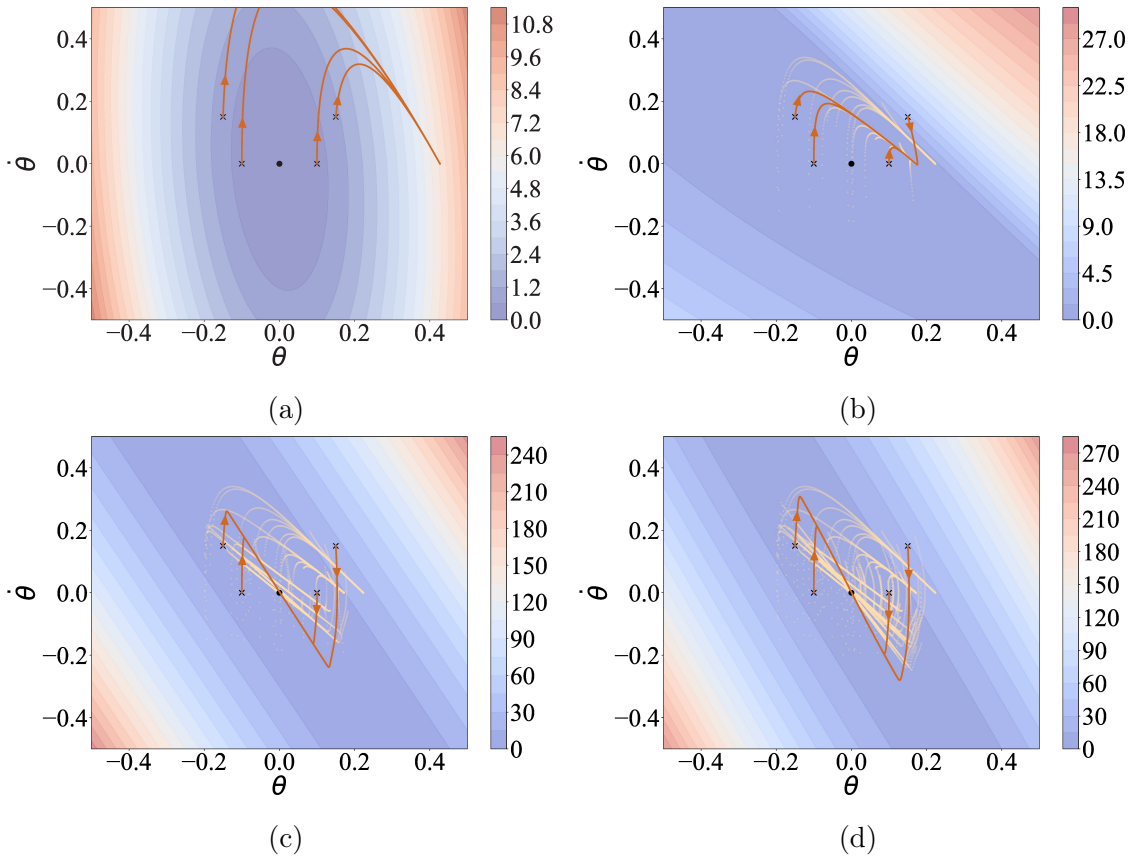


Figure 3.1: Trajectories generated by LYGE in different iterations in inverted pendulum environment. The counters show the learned CLF. The orange trajectories are generated by the learned controller in the current iteration. The light orange dots are the demonstrations generated in previous iterations, which also indicate the trusted tunnel $\mathcal{H}$. The black dot is the goal.

3.3 Analysis of LYGE

In this section, we provide several results to discuss the efficacy of LYGE. We start by reviewing some definitions and results on Lipschitz continuous and Lipschitz smoothness.

Definition 2. *A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is Lipschitz-continuous with constant L if*

$$||f(x) - f(y)|| \leq L\|x - y\|, \quad \forall x, y \in \mathbb{R}^n. \tag{3.11}$$

Definition 3. *A continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}^m$ is Lipschitz-smooth with*

Algorithm 1 LYGE Algorithm

Input: Demonstrations $\mathcal{D}^0$, set of initial states $\mathcal{X}_0$
Learn initial controller π_{init} using IL
Initialize $\hat{h}_\psi^0$ with an NN
Initialize V_θ^0 with the structure (3.7)
Initialize the controller with $\pi_\phi^0 = \pi_{\text{init}}$
for τ in $0, 1, ...$ **do**
  Sample states from $\mathcal{D}_x^\tau$
  Update $\hat{h}_\psi^\tau$ using MSE loss
  Update V_θ^τ and π_ϕ^τ using (3.9) and (3.10)
  Collect more trajectories ΔD^τ by starting from $\mathcal{D}_x^\tau$ and following π_ϕ^τ
  Construct $\mathcal{D}^{\tau+1} = \mathcal{D}^\tau \cup \Delta D^\tau$
end for

constant L if

$$||\nabla f(x) - \nabla f(y)|| \leq L\|x - y\|, \quad \forall x, y \in \mathbb{R}^n. \tag{3.12}$$

Lemma 1. *If a continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}^m$ is Lipschitz-smooth with constant L, then the following inequality holds for all $x, y \in \mathbb{R}^n$:*

$$(\nabla f(x) - \nabla f(y))^T (x - y) \leq L\|x - y\|^2. \tag{3.13}$$

Proof. The proof is straightforward that

$$(\nabla f(x) - \nabla f(y))^T (x - y) \leq \|\nabla f(x) - \nabla f(y)\| \cdot \|x - y\| \leq L\|x - y\|^2, \tag{3.14}$$

where the first equation follows the Cauchy-Schwarz inequality, and the second inequality comes from the definition of Lipschitz-smoothness. □

Lemma 2. *If function $f : \mathbb{R}^n \to \mathbb{R}^m$ is Lipschitz-smooth with constant L, then the following inequality holds for all $x, y \in \mathbb{R}^n$:*

$$f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{L}{2}\|y - x\|^2. \tag{3.15}$$

Proof. Define $g(t) = f(x + t(y - x))$. If f is Lipschitz-smooth with constant L, then from

Lemma 1, we have

$$\begin{aligned} & g'(t) - g'(0) \\ &= \left(\nabla f(x + t(y-x)) - \nabla f(x)\right)^T (y - x) \\ &= \frac{1}{t} \left(\nabla f(x + t(y-x)) - \nabla f(x)\right)^T \left((x + t(y-x)) - x\right) \\ &\leq \frac{L}{t} \|t(y-x)\|^2 = tL\|y - x\|^2. \end{aligned} \tag{3.16}$$

We then integrate this equation from $t = 0$ to $t = 1$:

$$\begin{aligned} f(y) = g(1) &= g(0) + \int_0^1 g'(t)dt \\ &\leq g(0) + \int_0^1 g'(0)dt + \int_0^1 tL\|y - x\|^2 dt \\ &= g(0) + g'(0) + \frac{L}{2}\|y - x\|^2 \\ &= f(x) + \nabla f(x)^T (y - x) + \frac{L}{2}\|y - x\|^2. \end{aligned} \tag{3.17}$$

□

Now, we provide the following theorem to show the convergence of LYGE. We say that the loss $\mathcal{L}^\tau_{\text{CLF}}$ ϵ'-*robustly converges* if (1) $V^\tau_\theta(x^{\text{goal}}) < \nu$; (2) $V^\tau_\theta(x) > \nu$ for all $x \in \mathcal{X} \setminus x^{\text{goal}}$; (3) $\epsilon + V^\tau_\theta(\hat{h}^\tau_\psi(x, \pi^\tau_\phi(x))) - \lambda V^\tau_\theta(x) \leq \epsilon'$ for all $x \in \mathcal{D}^\tau_x$. Here, $\epsilon' > 0$ is an arbitrarily small number.

Theorem 1. *Let L_π, L_V be the Lipschitz constants of the learned controller π^τ_ϕ and the gradient of the learned CLF V^τ_θ, respectively. Furthermore, let $\sigma \geq \|\nabla V^\tau_\theta\|$ be the upper bound of the gradient of V^τ_θ. Choose $\epsilon \geq \omega\sigma + \frac{L_V}{2}\omega^2 + \gamma\lambda(\sigma + \frac{L_V}{2}\gamma) + (1 + L_\pi)\gamma L_h(\sigma + \frac{L_V}{2}(1 + L_\pi)\gamma L_h) + \epsilon'$. If $\mathcal{L}^\tau_{\text{CLF}}$ ϵ'-robustly converges in each iteration, then LYGE converges and returns a stabilizing controller π^* that can be trusted within the converged trusted tunnel $\mathcal{H}^*$, where $\mathcal{H}^*$ contains all closed-loop trajectories starting from $\mathcal{X}_0$.*

Theorem 1 shows that with smooth dynamics and smooth NNs, if the margin ϵ is chosen to be large enough and the training loss $\mathcal{L}^\tau_{\text{CLF}}$ is small, we can conclude that the algorithm

converges and the system is asymptotically stable at x^{goal}. In our implementation, we use spectral normalization to limit the Lipschitz constants of the NNs. We also increase the amount of data collected in the exploration phase and use large NNs to decrease γ and ω. In this way, we can make ϵ a reasonably small value.

Now, we provide the proof of Theorem 1. We start by introducing several lemmas. First, we show that the learned CLF satisfies the CLF conditions (3.3) within the trusted tunnel $\mathcal{H}^\tau$.

Lemma 3. *Under the assumptions of Theorem 1, we have in iteration τ,*

$$V_\theta^\tau(x + h(x, \pi_\phi^\tau(x))) \leq \lambda V_\theta^\tau(x), \forall x \in \mathcal{H}^\tau. \tag{3.18}$$

Proof. For arbitrary $x \in \mathcal{H}^\tau$, let $\bar{x} \in \mathcal{D}_x^\tau$ be the closest point to x in the dataset $\mathcal{D}_x^\tau$, i.e., $\bar{x} = \arg\min_{y \in \mathcal{D}_x^\tau} \|y - x\|$. Using Lemma 2 and the definition of the trusted tunnel, we have:

$$\begin{aligned} V_\theta^\tau(\bar{x}) - V_\theta^\tau(x) = V_\theta^\tau(x + \bar{x} - x) - V_\theta^\tau(x) &\leq (\bar{x} - x)^\top \nabla V_\theta^\tau(x) + \frac{L_V}{2}\|\bar{x} - x\|^2 \\ &\leq \|\bar{x} - x\|\|\nabla V_\theta^\tau(x)\| + \frac{L_V}{2}\|\bar{x} - x\|^2 \leq \gamma\sigma + \frac{L_V}{2}\gamma^2. \end{aligned} \tag{3.19}$$

Using the Lipschitz continuity of the dynamics h and the controller π_ϕ^τ, we have

$$\begin{aligned} \|h(x, \pi_\phi^\tau(x)) - h(\bar{x}, \pi_\phi^\tau(\bar{x})\| &\leq L_h\|(x, \pi_\phi^\tau(x)) - (\bar{x}, \pi_\phi^\tau(\bar{x}))\| = L_h\|(x - \bar{x}, \pi_\phi^\tau(x) - \pi_\phi^\tau(\bar{x}))\| \\ &\leq L_h(\|x - \bar{x}\| + \|\pi_\phi^\tau(x) - \pi_\phi^\tau(\bar{x})\|) \leq (1 + L_\pi)\gamma L_h. \end{aligned} \tag{3.20}$$

Using Lemma 2 and the bounded gradient of V, and applying (3.19) and (3.20), we obtain

that for any $x \in \mathcal{H}^\tau$,

$$
\begin{aligned}
& V_\theta^\tau\left(h(x, \pi_\phi^\tau(x))\right) - \lambda V_\theta^\tau(x) \\
=& V_\theta^\tau\left(h(x, \pi_\phi^\tau(x)) - h(\bar{x}, \pi_\phi^\tau(\bar{x})) + h(\bar{x}, \pi_\phi^\tau(\bar{x}))\right) - \lambda V_\theta^\tau(x) \\
\leq& V_\theta^\tau\left(h(\bar{x}, \pi_\phi^\tau(\bar{x}))\right) + \|h(x, \pi_\phi^\tau(x)) - h(\bar{x}, \pi_\phi^\tau(\bar{x}))\|\sigma \\
& + \frac{L_V}{2}\|h(x, \pi_\phi^\tau(x)) - h(\bar{x}, \pi_\phi^\tau(\bar{x}))\|^2 - \lambda V_\theta^\tau(\bar{x}) + \lambda(V_\theta^\tau(\bar{x}) - V_\theta^\tau(x)) \\
\leq& V_\theta^\tau\left(h(\bar{x}, \pi_\phi^\tau(\bar{x}))\right) - \lambda V_\theta^\tau(\bar{x}) + \gamma\lambda(\sigma + \frac{L_V}{2}\gamma) + (1 + L_\pi)\gamma L_h(\sigma + \frac{L_V}{2}(1 + L_\pi)\gamma L_h).
\end{aligned} \tag{3.21}
$$

Then, we take the error of the learned dynamics into consideration. Using the error bound of the learned dynamics, we have:

$$
\begin{aligned}
& V_\theta^\tau\left(\hat{h}_\psi^\tau(\bar{x}, \pi_\phi^\tau(\bar{x}))\right) - \lambda V_\theta^\tau(\bar{x}) \\
=& V_\theta^\tau\left(h_\psi^\tau(\bar{x}, \pi_\phi^\tau(\bar{x})) - h_\psi^\tau(\bar{x}, \pi_\phi^\tau(\bar{x})) + \hat{h}_\psi^\tau(\bar{x}, \pi_\phi^\tau(\bar{x}))\right) - \lambda V_\theta^\tau(\bar{x}) \\
\geq& V_\theta^\tau\left(h_\psi^\tau(\bar{x}, \pi_\phi^\tau(\bar{x}))\right) - \|\hat{h}_\psi^\tau(\bar{x}, \pi_\phi^\tau(\bar{x})) - h_\psi^\tau(\bar{x}, \pi_\phi^\tau(\bar{x}))\|\sigma \\
& - \frac{L_V}{2}\|\hat{h}_\psi^\tau(\bar{x}, \pi_\phi^\tau(\bar{x})) - h_\psi^\tau(\bar{x}, \pi_\phi^\tau(\bar{x}))\|^2 - \lambda V_\theta^\tau(\bar{x}) \\
\geq& V_\theta^\tau\left(h_\psi^\tau(\bar{x}, \pi_\phi^\tau(\bar{x}))\right) - \lambda V_\theta^\tau(\bar{x}) - \omega\sigma - \frac{L_V}{2}\omega^2.
\end{aligned} \tag{3.22}
$$

Using the assumption that

$$
V_\theta^\tau(\hat{h}_\psi^\tau(\bar{x}, \pi_\phi^\tau(\bar{x}))) - \lambda V_\theta^\tau(\bar{x}) + \epsilon \leq \epsilon', \quad \forall \bar{x} \in \mathcal{D}_x^\tau, \tag{3.23}
$$

we have

$$
\begin{aligned}
& V_\theta^\tau\left(h_\psi^\tau(\bar{x}, \pi_\phi^\tau(\bar{x}))\right)\right) - \lambda V_\theta^\tau(\bar{x}) \\
\leq& V_\theta^\tau\left(\hat{h}_\psi^\tau(\bar{x}, \pi_\phi^\tau(\bar{x}))\right) - \lambda V_\theta^\tau(\bar{x}) + \omega\sigma + \frac{L_V}{2}\omega^2 \\
\leq& \epsilon' - \epsilon + \omega\sigma + \frac{L_V}{2}\omega^2.
\end{aligned} \tag{3.24}
$$

Therefore,

$$\begin{aligned}
&V_\theta^\tau\left(x + h(x, \pi_\phi^\tau(x))\right) - \lambda V_\theta^\tau(x) \\
\leq &\epsilon' - \epsilon + \omega\sigma + \frac{L_V}{2}\omega^2 + \gamma\lambda(\sigma + \frac{L_V}{2}\gamma) + (1 + L_\pi)\gamma L_h(\sigma + \frac{L_V}{2}(1 + L_\pi)\gamma L_h) \qquad (3.25)\\
\leq &0.
\end{aligned}$$

□

Lemma 3 suggests that under the assumptions of Theorem 1, the learned CLF satisfies the CLF condition (3.3b) in the trusted tunnel $\mathcal{H}^\tau$.

Next, we discuss the *growth* of the trusted tunnel $\mathcal{H}^\tau$ in each iteration.

Lemma 4. *Let T be the simulation horizon, and $x^\tau \in \mathcal{H}^\tau$ be the state that has the minimum value of the CLF at iteration τ, i.e.,*

$$x^\tau = \arg\min_{x \in \mathcal{H}^\tau} V_\theta^\tau(x). \qquad (3.26)$$

If $x^{\text{goal}} \notin \mathcal{H}^\tau$ and at least one of the sampled state x_d^τ satisfies

$$\|x_d^\tau - x^\tau\| < \frac{-\sigma + \sqrt{\sigma^2 + 2L_V\left(\frac{1}{\lambda^T} - 1\right)V_\theta^\tau(x^\tau)}}{L_V}, \qquad (3.27)$$

then during the exploration period, we have $\mathcal{H}^\tau \subsetneqq \mathcal{H}^{\tau+1}$.

Proof. Using the definition of the trusted tunnel $\mathcal{H}^\tau$ and the fact that $\mathcal{D}^\tau \subset \mathcal{D}^{\tau+1}$, we have $\mathcal{H}^\tau \subseteq \mathcal{H}^{\tau+1}$. Let $x^\tau \in \mathcal{H}^\tau$ be the state that has the minimum value of the CLF, i.e.,

$$x^\tau = \arg\min_{x \in \mathcal{H}^\tau} V_\theta^\tau(x). \qquad (3.28)$$

During the exploration process, let x_d^τ be a sampled initial state. Then, it follows from

Lemma 7 that

$$V_\theta^\tau(x_d^\tau) \leq V_\theta^\tau(x^\tau) + (x_d^\tau - x^\tau)^\top \nabla V_\theta^\tau(x^\tau) + \frac{L_V}{2}\|x_d^\tau - x^\tau\|^2, \tag{3.29}$$

If the trajectory leaves the trusted tunnel $\mathcal{H}^\tau$, then the claim is true. Otherwise, the trajectory stays in $\mathcal{H}^\tau$ in the simulation horizon T. Then, using Lemma 3, we have

$$V_\theta^\tau(x_d^\tau(T)) \leq \lambda V_\theta^\tau(x_d^\tau(T-1)) \leq \cdots \leq \lambda^T V_\theta^\tau(x_d^\tau(0)) = \lambda^T V_\theta^\tau(x_d^\tau). \tag{3.30}$$

Additionally, since the trajectory stays in $\mathcal{H}^\tau$, we have

$$V_\theta^\tau(x_d^\tau(T)) \geq V_\theta^\tau(x^\tau). \tag{3.31}$$

Using (3.29), (3.30), and (3.31), we obtain

$$\lambda^T \left(V_\theta^\tau(x^\tau) + (x_d^\tau - x^\tau)^\top \nabla V_\theta^\tau(x^\tau) + \frac{L_V}{2}\|x_d^\tau - x^\tau\|^2 \right) \geq V_\theta^\tau(x^\tau). \tag{3.32}$$

Note that V_θ^τ has bounded gradients σ. Therefore,

$$\frac{L_V}{2}\|x_d^\tau - x^\tau\|^2 + \sigma\|x_d^\tau - x^\tau\| - \left(\frac{1}{\lambda^T} - 1\right) V_\theta^\tau(x^\tau) \geq 0, \tag{3.33}$$

which implies

$$\|x_d^\tau - x^\tau\| \geq \frac{-\sigma + \sqrt{\sigma^2 + 2L_V\left(\frac{1}{\lambda^T} - 1\right) V_\theta^\tau(x^\tau)}}{L_V}. \tag{3.34}$$

This is the necessary condition for the trajectory to stay in $\mathcal{H}$. Otherwise, we have $V_\theta^\tau(x_d^\tau(T)) < V_\theta^\tau(x^\tau)$, which violates the assumption that x^τ is the minima of V_θ^τ in $\mathcal{H}^\tau$. Therefore, if we sample an initial state x_d^τ with

$$\|x_d^\tau - x^\tau\| < \frac{-\sigma + \sqrt{\sigma^2 + 2L_V\left(\frac{1}{\lambda^T} - 1\right) V_\theta^\tau(x^\tau)}}{L_V}, \tag{3.35}$$

the trajectory will leave $\mathcal{H}^\tau$, which implies $\mathcal{H}^\tau \subsetneqq \mathcal{H}^{\tau+1}$.

□

Note that the RHS of inequality (3.27) grows to infinity as T grows. Lemma 4 shows that the trusted tunnel $\mathcal{H}^\tau$ continues to grow when x^{goal} is not inside. Also, it shows that the trusted tunnel $\mathcal{H}^\tau$ cannot converge to some $\mathcal{H}^*$ such that $x^{\text{goal}} \notin \mathcal{H}^*$.

Now, we are ready to provide the proof of Theorem 1.

Proof. First, using Lemma 4, we know that the size of $\mathcal{H}^\tau$ increases monotonically. Since the size of $\mathcal{H}^\tau$ is upper-bounded by the compact state space, using Monotone Convergence Theorem, we know that $\mathcal{H}^\tau$ will converge to some set $\mathcal{H}^*$. Then, using Lemma 3, we know that the CLF conditions (3.3) are satisfied in $\mathcal{H}^*$[4]. Using Lemma 4, we know that $x^{\text{goal}} \in \mathcal{H}^*$. In addition, as $\mathcal{H}^\tau$ converges to $\mathcal{H}^*$, we know that starting from any initial states in $\mathcal{H}^*$, the agent cannot leave $\mathcal{H}^*$. Therefore, using the stabilizing result of CLF, the system is asymptotically stable at x^{goal}. □

3.4 Experiments

We conduct experiments in six environments including Inverted Pendulum, Cart Pole, Cart II Pole, Neural Lander [38], and the F-16 jet [1] with two tasks: ground collision avoidance (GCA) and tracking. To simulate the imperfect demonstrations, We collect imperfect and potentially unstable demonstrations for each environment using nominal controllers such as LQR [90] for Inverted Pendulum, PID [91] for Neural Lander and the F-16, and RL controllers for Cart Pole and Cart II Pole. In the first four environments, we collect 20 trajectories as demonstrations, and in the two F-16 environments, we collect 40 trajectories. We aim to answer the following questions in the experiments:

[4]Note that although $V_\theta^\tau(x^{\text{goal}})$ is not exactly zero, x^{goal} is still the global minimum of V_θ^τ, and therefore, the closed-loop system will still converge to x^{goal}.

1. How does LYGE compare with other algorithms for the case of stabilizing the system at goal?

2. What is the sampling efficiency of LYGE as compared to other baseline methods?

3. Can LYGE be used for systems with high dimensions?

The experiments are run on a 64-core AMD 3990X CPU @ 3.60GHz and four NVIDIA RTX A4000 GPUs (one GPU for each training job)

3.4.1 Baselines

We compare LYGE with the most relevant works in our problem setting including RL algorithm PPO [48], standard IL algorithm AIRL [64], and algorithms of IL from suboptimal demonstrations D-REX [70] and SSRR [71]. For PPO, we hand-craft reward functions following standard practices in reward function design [37] for stabilization problems. For AIRL, D-REX, and SSRR, we let them learn directly from the demonstrations. Other relevant approaches including 2IWIL [68], IC-GAIL [68], CAIL [67], and DPS [74] need different kinds of additional supervision, such as rankings or confidence labels, which are not implementable under the assumptions in our setting, so we do not include them. For a fair comparison, we initialize all the algorithms with the BC policy. Compared with these baselines, our algorithm has one additional assumption that we know the desired goal point. However, we believe that the comparison is still fair because we do not need the reward function or optimal demonstrations. While LYGE needs more information than D-REX and SSRR, the performance increment from LYGE is large enough that it is worth the additional information.

We also design two other baselines, namely, CLF-sparse and CLF-dense, to show the efficacy of the Lyapunov-guided exploration compared with learning the dynamics model from random samples [20], [21]. These methods require the stronger assumption of being able to sample from arbitrary states in the state space, which is unrealistic when performing

experiments outside of simulations. For both algorithms, we follow the same training process as LYGE, but instead of collecting samples of transitions by applying Lyapunov-guided exploration, we directly sample states and actions from the entire state-action space to obtain the training set of the dynamics. We note that CLF-sparse uses the same number of samples as LYGE, while CLF-dense uses the same number of samples as the RL and IL algorithms, which is much more than LYGE (see Table 3.1).

3.4.2 Implementation Details

Implementation of LYGE Our framework contains three models: the dynamics model $\hat{h}^\tau_\psi(x,u)$, the CLF $V^\tau_\theta(x) = x^\top S^\top S x + p_{\text{NN}}(x)^\top p_{\text{NN}}(x)$, and the controller $\pi^\tau_\phi(x)$. $\hat{h}^\tau_\psi(x,u)$, $p_{\text{NN}}(x)$, and $\pi^\tau_\phi(x)$ are all neural networks with two hidden layers with size 128 and Tanh as the activation function. $S \in \mathbb{R}^{n_x \times n_x}$ is a matrix of parameters. To limit the Lipschitz constant of the learned models L_V and L_π, we add spectral normalization [92] to each layer in the neural networks. We implement our algorithm in the PyTorch framework [93] based on the rCLBF repository[5] [19]. During training, we use ADAM [89] as the optimizer to optimize the parameters of the neural networks. The loss function used in training the controller and the CLF is

$$\mathcal{L}^\tau = \eta_{\text{goal}}\mathcal{L}^\tau_{\text{goal}} + \eta_{\text{pos}}\mathcal{L}^\tau_{\text{pos}} + \eta_{\text{ctrl}}\mathcal{L}^\tau_{\text{ctrl}}, \tag{3.36}$$

where η_{goal}, η_{pos}, η_{ctrl} are hyper-parameters, which we will further introduce in Section 3.4.4, and

$$\begin{aligned}
\mathcal{L}^\tau_{\text{goal}} &= V^\tau_\theta(x^{\text{goal}})^2, \\
\mathcal{L}^\tau_{\text{pos}} &= \frac{1}{N}\sum_{x\in\mathcal{D}^\tau_x} \max\left[\epsilon + V^\tau_\theta\left(\hat{h}^\tau_\psi(x,\pi^\tau_\phi(x))\right) - \lambda V^\tau_\theta(x), 0\right], \\
\mathcal{L}^\tau_{\text{ctrl}} &= \frac{1}{N}\sum_{x\in\mathcal{D}^\tau_x} \left\|\pi^\tau_\phi(x) - \pi_{\text{init}}(x)\right\|^2 + \|\pi^\tau_\phi(x) - \pi^{\tau-1}_\phi(x)\|^2.
\end{aligned} \tag{3.37}$$

[5] https://github.com/MIT-REALM/neural_clbf (BSD-3-Clause license)

Note that there is another term in the loss: $\frac{1}{N}\sum_{x\in\mathcal{X}\setminus x^{\text{goal}}}\max\left[\nu - V_\theta^\tau(x), 0\right]$. However, this loss term is often 0 in the training so we omit the discussion of this term here. In our implementation, we add the term $\|\pi_\phi^\tau(x) - \pi_{\text{init}}(x)\|^2$ to further limit the exploration of π_ϕ^τ, and also make the training more stable, as there is a non-changing reference for π_ϕ^τ.

In the exploration period, starting from some states in the trusted tunnel $x(0) \in \mathcal{H}^\tau$, following the newly updated controller in this iteration π_ϕ^τ, the trajectories collected are $\{x(0), \pi_\phi^\tau(x(0)), x(1), \pi_\phi^\tau(x(1)), x(2), \pi_\phi^\tau(x(2)), \dots\}$, where $x(t+1) = h(x(t), \pi_\phi^\tau(x(t)))$. The trajectory ends when the maximum simulation time step T is reached or the states are no longer safe.

Implementation of the baselines We implement PPO based on the open-source python package stablebaselines3[6] [94], AIRL based on the open-source python package Imitation[7] [95], and D-REX and SSRR based on their official implementations [8,9], with some adjustments based on the CAIL repository [10] [67]. All the neural networks in the baselines, including the actor, the critic, the discriminator, and the reward module, have two hidden layers with size 128 and Tanh as the activation function. We use ADAM [89] as the optimizer with a learning rate 3×10^{-4} to optimize the parameters of the neural networks. For CLF-sparse and CLF-dense, we use the same NN structures as LYGE, but pre-train the dynamics $\hat{h}_\psi^\tau(x, u)$ from state-action-state transitions randomly sampled from the state-action space. Other implementation details are the same as LYGE.

3.4.3 Environments

Inverted Pendulum Inverted Pendulum (Inv Pendulum) is a standard benchmark for testing control algorithms. The state of the inverted pendulum is $x = [\theta, \dot{\theta}]^\top$, where θ is the

[6] https://github.com/DLR-RM/stable-baselines3 (MIT license)
[7] https://github.com/HumanCompatibleAI/imitation (MIT license)
[8] https://github.com/dsbrown1331/CoRL2019-DREX (MIT license)
[9] https://github.com/CORE-Robotics-Lab/SSRR
[10] https://github.com/Stanford-ILIAD/Confidence-Aware-Imitation-Learning (MIT license)

angle of the pendulum to the straight-up location, and the control input is the torque. The dynamics is given by $\dot{x} = f(x) + g(x)u$, with

$$f(x) = \begin{bmatrix} \dot{\theta} \\ \frac{g\theta}{L} - \frac{b\dot{\theta}}{mL^2} \end{bmatrix} \qquad (3.38)$$
$$g(x) = \begin{bmatrix} 1 \\ \frac{}{mL^2} \end{bmatrix}$$

where $g = 9.80665$ is the gravitational acceleration, $m = 1$ is the mass, $L = 1$ is the length, and $b = 0.01$ is the damping. We define the goal point at $x^{\text{goal}} = [0, 0]^\top$. We let the discrete-time dynamics be $x(t+1) = x(t) + \dot{x}(t)\delta t$, where the simulating time step $\delta t = 0.01$. To simulate imperfect demonstrations, the demonstration data is collected by an LQR controller with noise that leads to oscillation of the pendulum around a point away from the goal. We use the reward function $r(x) = 2.0 - |\theta|$ to train RL algorithms.

We set the initial state with $\theta \in [-0.2, 0.2]$ and $\dot{\theta} \in [-0.2, 0.2]$. For the demonstrations, we solve the LQR controller with $Q = I_2$ and $R = 1$, where I_n is the n-dimensional identity matrix, and add standard deviation 0.1 and bias 4.0 to the solution to make it unstable. We collect 20 trajectories for the demonstrations, where each trajectory has 1000 time steps. For hyper-parameters in the loss function (3.36), we use $\eta_{\text{goal}} = 10.0$, $\eta_{\text{pos}} = 10.0$, $\eta_{\text{ctrl}} = 1.0$.

Cart Pole The Cart Pole environment we use is a modification of the InvertedPendulum environment introduced in OpenAI Gym [37][11]. However, the original reward function is not suitable for the stabilization task because there is only one term: “alive bonus" in the original reward function. Therefore, we change the reward function to be $r = 1 - \|x\|$ where x is the current state.

We collect demonstrations using an RL policy that has not fully converged. We collect 20 trajectories for the demonstrations, where each trajectory has 1000 time steps. For hyper-parameters in the loss function (3.36), we use $\eta_{\text{goal}} = 1000.0$, $\eta_{\text{pos}} = 10.0$, $\eta_{\text{ctrl}} = 1000.0$.

[11] The original name is InvertedPendulum

Note that η_{ctrl} is large because the control actions in this environment are often tiny.

Cart II Pole The Cart II Pole environment we use is a modification of the InvertedDoublePendulum environment introduced in OpenAI Gym [37][12]. To provide more signal for the stabilization task, we change the reward function to be $r = r_{\text{origin}} - \|q\|$, where r_{origin} is the original reward function and q is the current position of the cart.

We collect demonstrations using an RL policy that has not fully converged. We collect 20 trajectories for the demonstrations, where each trajectory has 1000 time steps. For hyper-parameters in the loss function (3.36), we use $\eta_{\text{goal}} = 100000.0$, $\eta_{\text{pos}} = 5.0$, $\eta_{\text{ctrl}} = 3000.0$. Note that η_{ctrl} is large because the control actions in this environment are often very small.

Neural Lander Neural lander [38] is a widely used benchmark for systems with unknown disturbance. The state of the Neural Lander is $x = [p_x, p_y, p_z, v_x, v_y, v_z]^\top$, with control input $u = [f_x, f_y, f_z]^\top$. p_x, p_y, p_z are the 3D displacements, v_x, v_y, v_z are the 3D velocities, and f_x, f_y, f_z are the 3D forces. The dynamics is given by $\dot{x} = f(x) + g(x)u$, with

$$
\begin{aligned}
f(x) &= \left[v_x, v_y, v_z, \frac{F_{a1}}{m}, \frac{F_{a2}}{m}, \frac{F_{a3}}{m} - g' \right]^\top \\
g(x) &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1/m & 0 & 0 \\ 0 & 1/m & 0 \\ 0 & 0 & 1/m \end{bmatrix}
\end{aligned}
\tag{3.39}
$$

where $g' = 9.81$ is the gravitational acceleration, $m = 1.47$ is the mass, and F_a is the learned dynamics of the ground effect, represented as a 4-layer neural network. The goal is to stabilize the neural lander at a user-defined point near the ground. We define the goal point

[12]The original name is InvertedDoublePendulum

at $x^{\text{goal}} = [0, 0, 0.5, 0, 0, 0]^\top$. We let the discrete-time dynamics be $x(t+1) = x(t) + \dot{x}(t)\delta t$, where the simulating time step $\delta t = 0.01$. For the reward function, we use $r(x) = 10 - \|x\|$.

We set the initial state with $p_x, p_y \in [-2, 2]$, $p_z \in [1, 2]$, and $v_x, v_y, v_z = 0$. For the demonstrations, we use a PD controller

$$u = \begin{bmatrix} -8p_x - v_x \\ -8p_y - v_y \\ -8p_z - v_z + mg' \end{bmatrix}, \tag{3.40}$$

which makes the neural lander oscillate and cannot reach the goal point because of the strong ground effect. We collect 20 trajectories for the demonstrations, where each trajectory has 1000 time steps. For hyper-parameters in the loss function (3.36), we use $\eta_{\text{goal}} = 100.0$, $\eta_{\text{pos}} = 50.0$, $\eta_{\text{ctrl}} = 1.0$.

F-16 Ground Collision Avoidance (GCA) F-16 [1][13] is a fixed-wing fighter model. Its state space is 16D including air speed v, angle of attack α, angle of sideslip β, roll angle ϕ, pitch angle θ, yaw angle ψ, roll rate P, pitch rate Q, yaw rate R, northward horizontal displacement p_n, eastward horizontal displacement p_e, altitude h, engine thrust dynamics lag pow, and three internal integrator states. The control input is 4D including acceleration at z direction, stability roll rate, side acceleration + raw rate, and the throttle command. The dynamics are complex and cannot be described as ODEs, so the authors of the F-16 model provide look-up tables to describe the aerodynamics. The lookup tables describe an approximation of the Lipschitz real dynamics, and also since we simulate the system in a discrete way in the experiments, the look-up table does not violate our assumptions about the real dynamics. In GCA, the F-16 starts at an initial condition with the head pointing at the ground. The goal is to pull up the aircraft as soon as possible, avoid colliding with the ground, and fly at a height between 800 ft and 1200 ft. We define the goal point at

[13] https://github.com/stanleybak/AeroBenchVVPython (GPL-3.0 license)

$h = 1000$. The simulating time step is 0.02.

We set the initial state with $v \in [520, 560]$, $\alpha = 0.037$, $\beta = 0$, $\phi = 0$, $\theta = -1.4\pi$, $\psi = 0.8\pi$, $P \in [-5, 5]$, $Q \in [-1, 1]$, $R \in [-1, 1]$, $p_n = 0$, $p_e = 0$, $h \in [2600, 3000]$, $pow \in [4, 5]$. For the demonstrations, we use the controller provided with the model. We collect 40 trajectories for the demonstrations, where each trajectory has 500 time steps. For hyper-parameters in the loss function (3.36), we use $\eta_{\text{goal}} = 100.0$, $\eta_{\text{pos}} = 50.0$, $\eta_{\text{ctrl}} = 50.0$.

F-16 Tracking The F-16 Tracking environment uses the same model as the F-16 GCA environment. We define the goal point at $[p_n, p_e, h] = [7500, 5000, 1500]$, and $\psi = \arctan \frac{p_n}{p_e}$. The simulating time step is 0.02.

We set the initial state with $v \in [520, 560]$, $\alpha = 0.037$, $\beta = 0$, $\phi \in [-0.1, 0.1]$, $\theta \in [-0.1, 0.1]$, $\psi \in [-0.1, 0.1]$, $P \in [-0.5, 0.5]$, $Q \in [-0.5, 0.5]$, $R \in [-0.5, 0.5]$, $p_n = 0$, $p_e = 0$, $h = 1500$, $pow \in [4, 5]$. For the demonstrations, we use the controller provided with the model. However, the original PID controller cannot pull up the aircraft early enough or cannot track the waypoint precisely. We collect 40 trajectories for the demonstrations, where each trajectory has 500 time steps. For hyper-parameters in the loss function (3.36), we use $\eta_{\text{goal}} = 100.0$, $\eta_{\text{pos}} = 50.0$, $\eta_{\text{ctrl}} = 1000.0$.

3.4.4 Choice of Hyper-parameters

In our framework, the hyper-parameters include the Lyapunov convergence rate λ, the robust parameter ϵ, the weights of the losses η_{goal}, η_{pos}, η_{ctrl}, and the parameters used in training including the learning rate. λ controls the convergence rate of the learned controller. Larger λ enables the controller to reach the goal faster, but it also makes the training harder. In our implementation, we choose $\lambda = 1 - \delta t$, where δt is the simulation time step. ϵ controls the robustness of the learned CLF w.r.t. the Lipschitz constant of the environment, the radius of the trusted tunnel, and the error of the learned dynamics. It should be large enough to satisfy Theorem 1, but large ϵ also makes the training harder. We choose $\epsilon = 1.0$

for Inverted Pendulum, Cart Pole, Cart II Pole, and Neural Lander, and $\epsilon = 2.0$ for the F-16 environments. The weights of the losses control the importance of each loss term. Generally, in a simple environment, we tend to use large η_{goal} and η_{pos} with small η_{ctrl}, so that the radius of the trusted tunnel can be large and the controller can explore more regions in each iteration, which makes the convergence of our algorithm faster. In a complex environment, however, we tend to use small η_{goal} and η_{pos} with large η_{ctrl}. This will limit the divergence between the updated controller and the reference controllers (initial controller and the controller learned in the last iteration) so that the radius of the trusted tunnel won't be so large that the learned CLF is no longer valid. The learning rate controls the convergence rate of the training. Large learning rates can make the training faster, but it may also make the training unstable and miss the minimum. We let the learning rate be 3×10^{-4}.

3.4.5 Results and Discussions

We train each algorithm in each environment 5 times with different random seeds and test the converged controllers 20 times each. In Figure 3.2 we show the distance to the goal w.r.t. the simulation time steps. We can observe that LYGE achieves comparable or better results in terms of stabilizing the systems in all environments, especially in high-dimensional complex systems like Neural Lander and F-16. In Neural Lander, none of the baselines can reach the goal as they prioritize flying at a higher altitude to avoid collisions. In F-16 environments, the baselines either pull up the aircraft too late or have large tracking errors. LYGE, however, can finish these tasks perfectly.

Specifically, compared with PPO, LYGE achieves comparable results in simpler environments like Cart Pole and Cart II Pole and behaves much better in complex environments like Neural Lander and F-16. This is because PPO is a policy gradient method that approximates the solution of the Bellman equation, but getting an accurate approximation is hard for high-dimensional systems. In PPO the reward function can only describe "where the goal is", but our learned CLF can explicitly tell the system "how to reach the goal". Compared with AIRL,

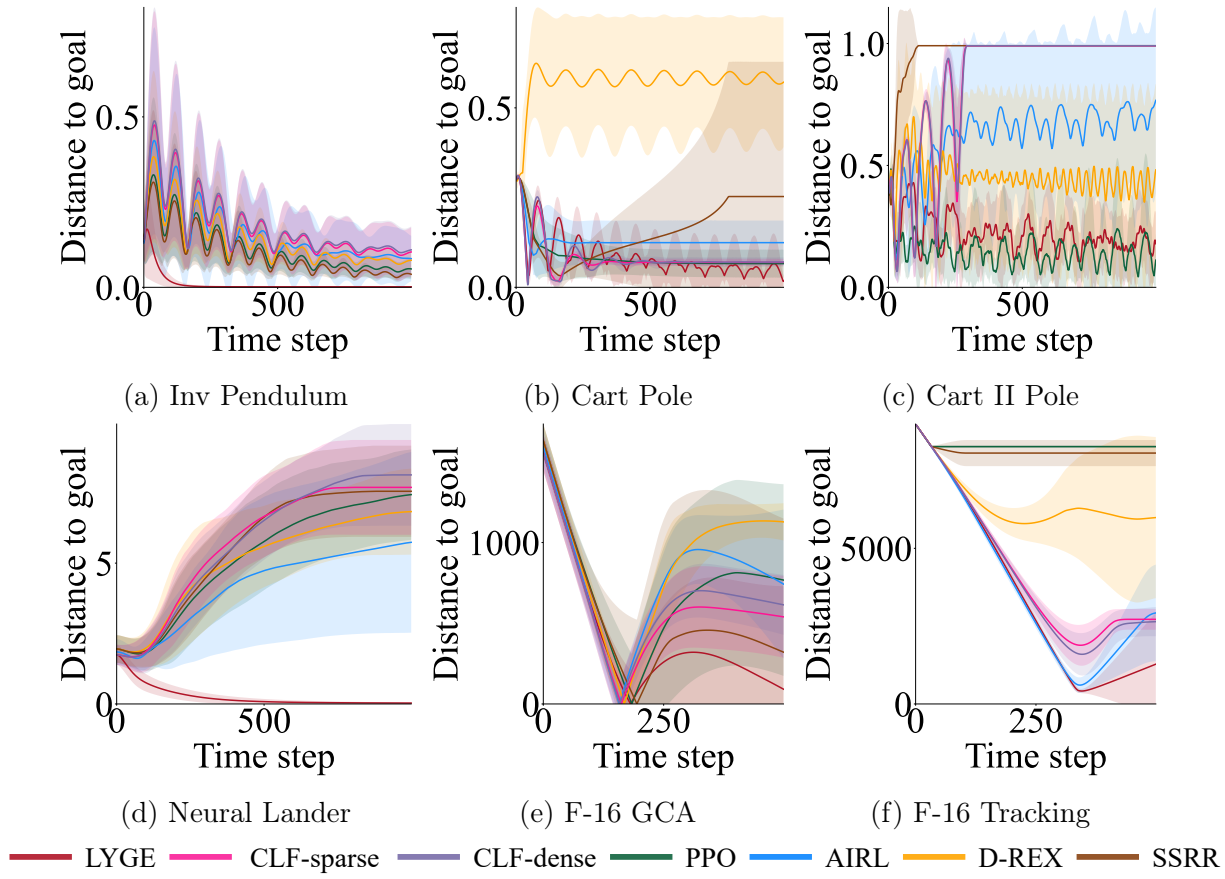


Figure 3.2: The distance to the goal w.r.t. time step of LYGE and the baselines. The solid lines show the mean distance while the shaded regions show the standard deviation. Note that the curve corresponding to CLF-sparse almost overlaps with the curve corresponding to CLF-dense, so the curve for CLF-sparse might not be visible in some of the plots.

which learns the same policy as the demonstrations and cannot make improvements, LYGE learns a policy that is much better than the demonstrations. Compared with ranking-guided algorithms D-REX and SSRR, LYGE behaves better because ranking guidances provide less information than our CLF guidance, and are not designed to explicitly encode the objective of reaching the goal. Compared with CLF-sparse and CLF-dense, LYGE outperforms them because Lyapunov-guided exploration provides a more effective way to sample in the state space to learn the dynamics rather than random sampling. Although CLF-dense uses many more samples than CLF-sparse and LYGE, its performance does not significantly improve. This shows that naïvely increasing the number of samples without guidance does little to improve the accuracy of the learned dynamics in high-dimensional spaces.

Table 3.1: Number of samples (k) used for training in different environments.

Algorithm	Inv Pendulum	Cart Pole	Cart II Pole	Neural Lander	F-16 GCA	F-16 Tracking
LYGE	160	160	480	240	480	560
Baselines	2000	500	5000	5000	2000	10000

In Table 3.1, we show the number of samples used in the training. It is shown that LYGE needs 68% to 95% fewer samples than other algorithms. This indicates that our Lyapunov-guided exploration explores only the necessary regions in the state space thus improving the sample efficiency.

Figure 3.3 shows the expected rewards and standard deviations of different algorithms with respect to the number of samples used in the training process. We do not show the training processes of CLF-sparse and CLF-dense because they use all the samples from the beginning of the training. It is demonstrated that LYGE achieves similar rewards in relatively low dimensional systems Inverted Pendulum, Cart Pole, and Cart II Pole, and significantly higher rewards in relatively high dimensional systems Neural Lander, F-16 GCA, and F-16 Tracking. We can also observe that LYGE converges very fast, using about one order of magnitude fewer samples than the baselines. This supports our argument that the Lyapunov-guided exploration explores only the necessary regions in the state space thus improving the sample efficiency. Note that the reward of LYGE decreases a bit before rising up again in the F-16 environments. This is because the reward can only measure the cumulative error w.r.t. the goal. For LYGE, before convergence, the goal is not inside the trusted tunnel $\mathcal{H}^\tau$. Therefore, during the exploration stage, the learned controller guides the agent to get out of the trusted tunnel $\mathcal{H}^\tau$. When this happens, there is no guarantee of the agent's behavior, so the cumulative error can increase. However, after convergence, LYGE is able to stabilize the system. The cumulative error is small and the reward is high.

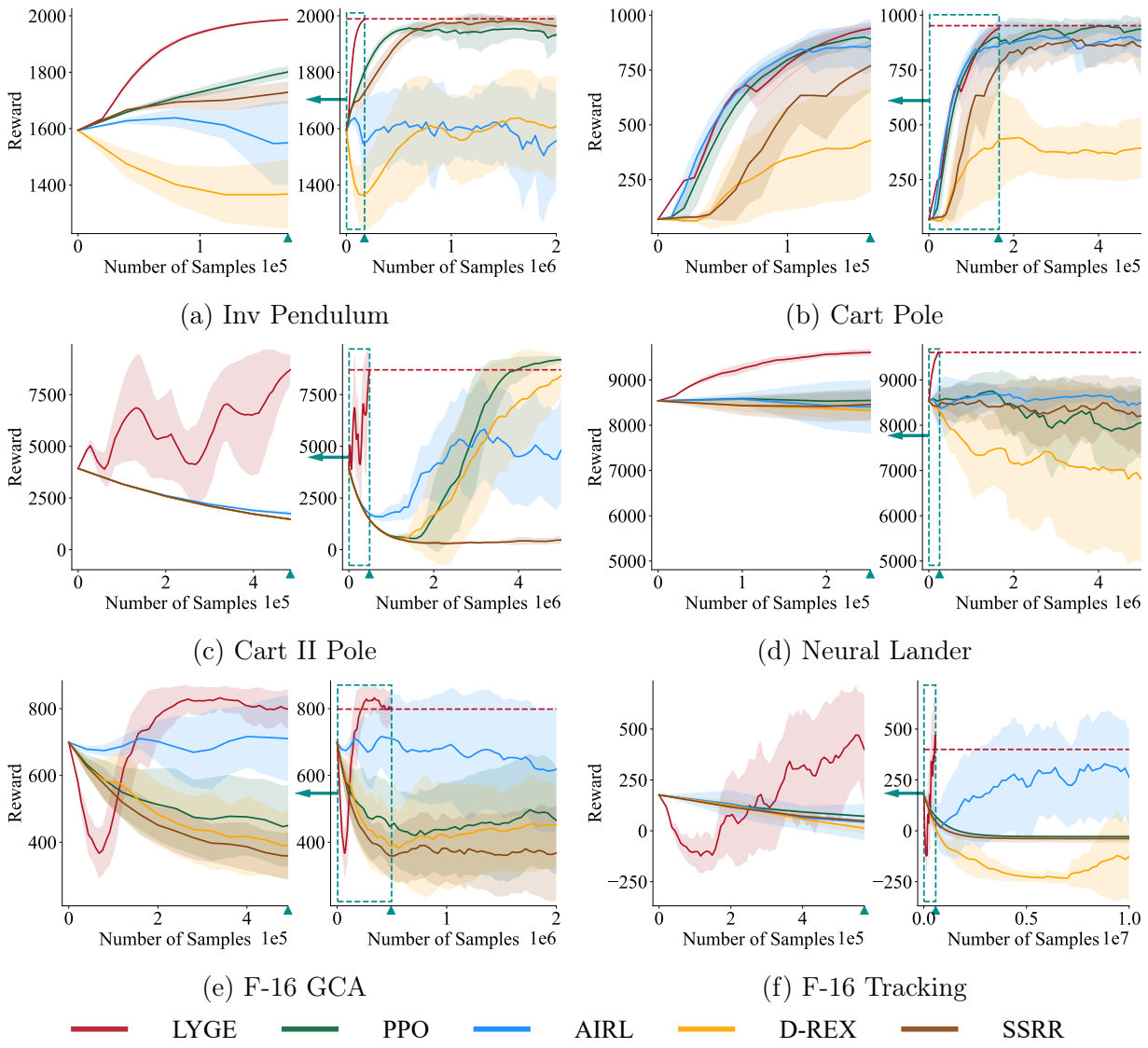


Figure 3.3: The expected return of LYGE and the baselines with respect to the number of samples. The dashed red line shows the converged reward of LYGE. In the right of each subplot, we show the whole curve of the expected return w.r.t. the number of samples, and since our LYGE converges too fast compared with the baselines, we zoom in the region inside the dashed rectangle and show this region in the left. The triangle on the x-axis shows the number of samples needed by LYGE to converge. The reward at 0 number of samples is the reward of demonstrations.

3.4.6 Ablation Studies

We first show the influence of the optimality of the demonstrations. We use the Inverted Pendulum environment and the Neural Lander environment and collect demonstrations with

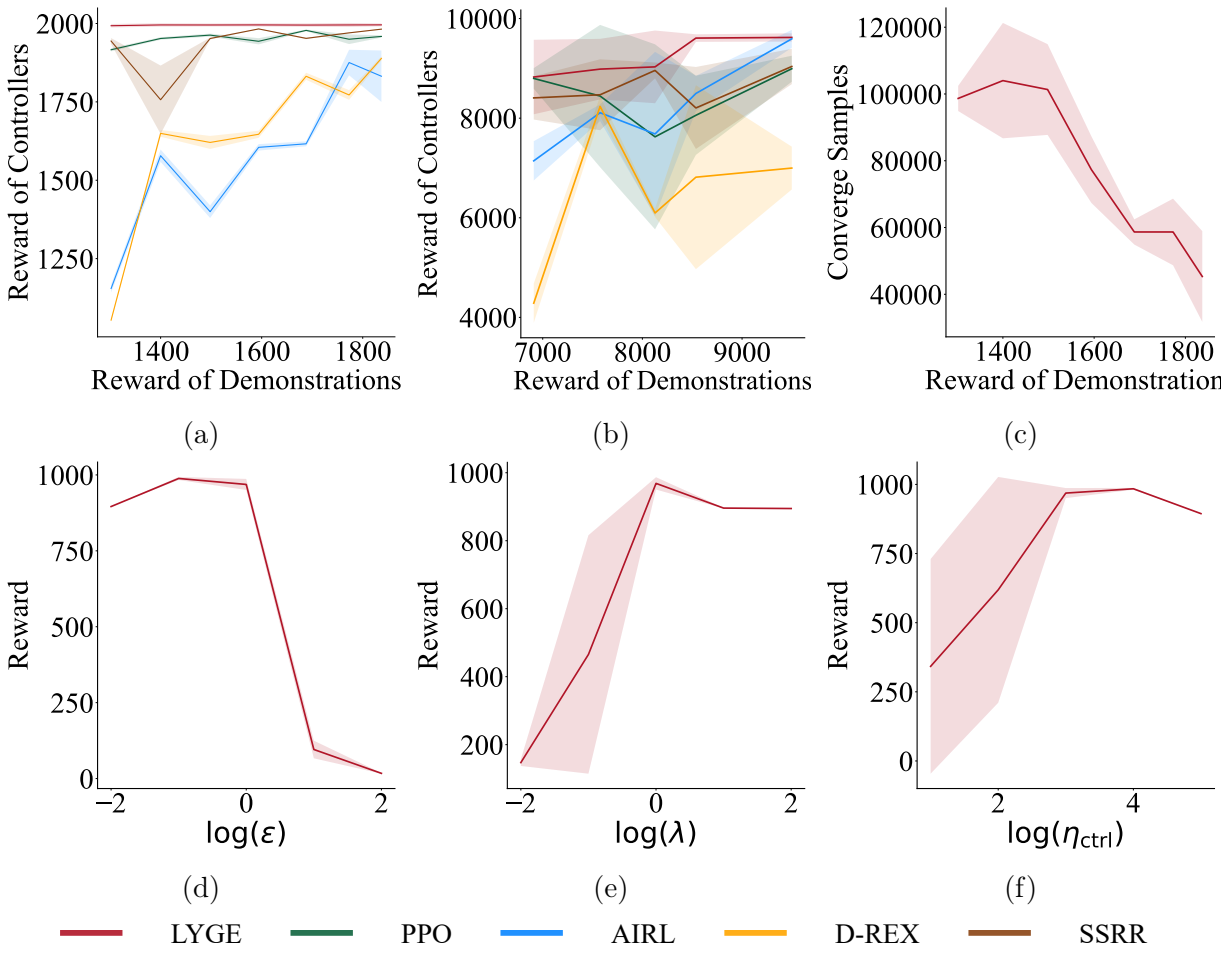


Figure 3.4: Ablation studies. (a) The converged reward w.r.t. demonstration rewards in the Inv Pendulum environment. (b) The converged reward w.r.t. demonstration rewards in the Neural Lander environment. (c) The number of samples used before LYGE converges w.r.t. demonstration rewards. (d) The converged reward w.r.t. ϵ. (e) The converged reward w.r.t. λ. (f) The converged reward w.r.t. η_{ctrl}.

different levels of optimality by varying the distance between the actual goal state and the target state of the demonstrator controller. For each optimality level, we train each algorithm 3 times with different random seeds and test each converged controller 20 times. We omit the experiments on CLF-sparse and CLF-dense since their performances are not related to the demonstrations. The results are shown in Figure 3.4a and Figure 3.4b. We observe that LYGE outperforms other algorithms with demonstrations at different levels of optimality. In addition, we do not observe a significant performance drop of LYGE as the demonstrations become worse. This is because the quality of the demonstrations only influences the con-

vergence speed of LYGE, instead of the controller. PPO's behavior is also consistent since the reward function remains unchanged, but it consistently performs worse than LYGE. IL algorithms, however, have a significant performance drop as the demonstrations get worse because they all depend on the quality of the demonstrations.

The quality of the demonstrations can also influence the convergence speed of LYGE. In the Inv Pendulum environment, we define the algorithm converges when the reward is larger than 1980. We plot the number of samples used for convergence w.r.t. the reward of demonstrations in Figure 3.4c. It is shown that the better the given demonstrations, the fewer samples LYGE needs for convergence.

We also do ablations to investigate the influence of the hyperparameter ϵ. We test LYGE in the Cart Pole environment and change ϵ from 0.01 to 100. The results are shown in Figure 3.4d, which demonstrates that LYGE works well when ϵ is large enough to make the system robust and small enough that it does not make the training very hard.

The λ in Equation (3.9) is another hyperparameter that controls the convergence rate of the learned policy. We test LYGE in the Cart Pole environment and change λ from 0.01 to 100. The results are shown in Figure 3.4e, demonstrating that LYGE works well with λ in some range. If λ is too small, the convergence rate is too small and the system cannot be stabilized within the simulation time steps. If λ is too large, loss (3.9) becomes too hard to converge, so the controller cannot stabilize the system.

During training, η_{ctrl} controls the expansion of the trusted tunnel. We do ablations for η_{ctrl} in the Cart Pole environment and change η_{ctrl} from 10 to 10^5. The results are shown in Figure 3.4f, which suggests that LYGE can work well when the value of η_{ctrl} is within a certain range. If η_{ctrl} is too small, the controller leaves the trusted tunnel too early and enters the space with unknown dynamics instead of smoothly expanding the trusted tunnel. If η_{ctrl} is too large, exploration is strongly discouraged and the trusted tunnel will expand too slowly.

3.5 Discussions

3.5.1 Verification of the Learned CLFs

The focus of this paper is to use the learned CLF to guide the exploration and synthesize feedback controllers for high-dimensional unknown systems. The experimental results show that the learned controllers are goal-reaching, and we find that the learned CLF satisfies the conditions in the majority of the trusted tunnel, and the learned CLF can successfully guide the controller to explore the useful subspace. However, we do not claim that our learned CLF is formally verified. Although the theories we provide suggest that the learned CLF satisfies the CLF conditions (3.3) under certain assumptions, these assumptions may not be satisfied in experiments. For example, we cannot theoretically guarantee that the loss always ϵ'-robustly converges on any system. If we want to verify the learned CLF is valid in the whole trusted tunnel, additional verification tools are needed, including SMT solvers [16], [96], Lipschitz-informed sampling methods [97], etc. However, these verification tools are known to have poor scalability and thus generally used in systems with dimensions less than 6 [16], [21]. Scalable verification for the learned CLFs remains an open problem.

3.5.2 Possible Future Directions

Our algorithm can also benefit from the verification tools. For instance, SMT solvers allow us to find counterexamples to augment the training data to make our learned CLF converge faster. In addition, *almost Lyapunov functions* [98] show that the system can still be stable even if the Lyapunov conditions do not hold everywhere. We are excited to explore these possible improvements in our future work.

3.6 Conclusion

We propose a general learning framework, **LY**apnov **G**uided **E**xploration (LYGE), for learning stabilizing controllers in high-dimensional environments with unknown dynamics. LYGE iteratively fits the local dynamics to form a trusted tunnel, and learns a control policy with a CLF to guide the exploration and expand the trusted tunnel toward the goal. Upon convergence, the learned controller stabilizes the closed-loop system at the goal point. We show the stabilizing ability of LYGE, and provide experimental results to demonstrate that LYGE performs comparably or better than the baseline RL, IL, and neural certificate methods. We also demonstrate that the same framework can be applied to learn other certificates in environments with unknown dynamics.

Our framework has a few **limitations**: we require a set of demonstrations for initialization in high-dimensional systems, although they can be potentially imperfect. Without them, LYGE may take a long time to expand the trusted tunnel to the goal. In addition, we need Lipschitz assumptions for the dynamics to derive the theoretical results. If the dynamics do not satisfy the Lipschitz assumptions, the learned CLF might be invalid even inside the trusted tunnel. Moreover, although we observe the convergence of the loss terms in training on all our case studies, it is hard to guarantee that the loss always converges on any system. Finally, if we desire a fully validated Lyapunov function, we need to employ formal verification tools, and we have provided a detailed discussion in Section 3.5.

Chapter 4

Compositional Neural Certificates for Networked Dynamical Systems

In this chapter, we proposed **Neur**al **ISS** Lyapunov functions (NeurISS) to learn stabilizing controllers for networked dynamical systems. The results of this chapter are based on [5].

4.1 Problem Setting

In this chapter, we consider the following networked dynamical system involving n subsystems $\mathcal{N} = \{1, 2, \ldots, n\}$. The dynamics of each subsystem are given by

$$\dot{x}_i = f_i(x_i, x_{i_1}, x_{i_2}, \ldots, x_{i_{n_i}}, u_i) = f_i(x_i, x_{\mathcal{N}_i}, u_i) \tag{4.1}$$

where $x_i \in \mathbb{R}^{d_i}$, $u_i \in \mathbb{R}^{p_i}$ is the state and control inputs of each subsystem i, and $x_{\mathcal{N}_i} = (x_{i_1}, x_{i_2}, \ldots, x_{i_{n_i}})$ is used to denote the states of the neighbors of subsystem i, $\mathcal{N}_i = \{i_1, i_2, \ldots, i_{n_i}\}$ (not including i itself) which affect the dynamics of the subsystem i. We use $x = (x_1, \ldots, x_n)$, $u = (u_1, \ldots, u_n)$, and $f = (f_1, \ldots, f_n)$ to denote the vector of states, actions, and dynamics across all subsystems (i.e. the overall system), respectively. We also denote $d = \sum_{i=1}^{n} d_i$ and $p = \sum_{i=1}^{n} p_i$ to the dimension of x and u, respectively. Our goal is to design a controller

$u = \pi(x)$ such that the closed-loop system is asymptotically stable around a goal set x^{goal}, formally defined as follows.

Definition 4. *Consider a goal set* $x^{\text{goal}} := x_1^{\text{goal}} \times \cdots \times x_n^{\text{goal}}$ *where each* x_i^{goal} *is a closed convex subset of* $\mathbb{R}^{d_i}$. *The closed-loop system is globally asymptotically stable about* x_i^{goal} *if for any initial state* $x(0)$, *the trajectory* $x(t)$ *satisfies* $\lim_{t\to\infty} \text{dist}(x(t), x^{\text{goal}}) = 0$, *where* $\text{dist}(x, x^{\text{goal}}) := \inf_{y\in x^{\text{goal}}} \|x - y\|$ *is the distance between the point* x *and the set* x^{goal}.

Example 1. *A simple networked system is the truck Platoon system with* $n + 2$ *trucks, where the* 0*-th (leading) truck can drive freely within the speed and acceleration limits, and the* $(n + 1)$*-th (last) truck will be driven in a way so that the total length of the platoon is roughly kept as a pre-defined constant. We assume other trucks are controllable but can only measure the distance to the two trucks directly in front of and behind themselves. We want to control these trucks so that the trucks in the whole platoon are spread evenly. Specifically, for truck* $i \in \{1, 2, ..., n\}$, *the states are given as* $x_i = [p_i^f, p_i^b, v_i]^\top$ *and the neighboring subsystems are trucks* $\mathcal{N}_i = \{i-1, i+1\}$, *where* p_i^f *is the distance between the* i*-th truck and the* $(i-1)$*-th truck,* p_i^b *is the distance between the* i*-th truck and the* $(i+1)$*-th truck, and* v_i *is the velocity of the* i*-th truck. The control input is the acceleration of the* i*-th truck. Therefore, the dynamics of truck* i *is* $\dot{x}_i = [v_{i-1} - v_i, v_i - v_{i+1}, a_i]^\top$. *The goal set for each truck* i *is uniquely defined as the set of states satisfying* $p_i^f = p_i^b$.

Notations Function $\alpha : [0, \infty) \to [0, \infty)$ is said to be class-$\mathcal{K}$ if α is continuous, strictly increasing, and $\alpha(0) = 0$. Class-$\mathcal{K}$ function α is said to be class-$\mathcal{K}_\infty$ if $\lim_{a\to+\infty} \alpha(a) = +\infty$.

4.2 Compositional Neural Certificates

4.2.1 Decentralized Controller and ISS Lyapunov Functions

Lyapunov functions are widely used to guarantee the stability of dynamical systems (see Section 3.1 for an introduction). A common paradigm for stabilizing a dynamical system

is to jointly search for a controller $u = \pi(x)$ and a Lyapunov function $V(x)$. However, this approach is not scalable for large-scale networked dynamical systems due to the sheer dimension of the state space, and the controller of the form $u = \pi(x)$, which requires global information of the entire network.

To address the issues, our framework NeurISS includes two key components: decentralized controllers and compositional certificates. We consider the class of *decentralized controllers* $u_i = \pi_i(x_i)$, which only needs local information within the small subsystem. Further, we consider *compositional certificates*, that is, instead of finding a single Lyapunov function $V(x)$ for the whole system, we find one Lyapunov function $V_i(x_i)$ for each subsystem i. The individual Lyapunov functions only depend on the subsystem state, which is much smaller in dimension. Further, based on [35], we provide the following Lemma 5 which shows that when the individual Lyapunov functions V_i satisfy an ISS-style condition, they will certify the stability of the entire dynamical system. Since it is the collection of the ISS Lyapunov functions $\{V_i\}_{i=1}^n$ that certify the stability of the entire dynamics, we also call such ISS Lyapunov functions $\{V_i\}_{i=1}^n$ as a "compositional" certificate to distinguish them from typical certificates that only contain one Lyapunov function for the entire system.

Lemma 5. *Suppose each subsystem has a decentralized controller $u_i = \pi_i(x_i)$ and a continuously differentiable function $V_i(x_i)$. Suppose: (1) For each i, there exists $\mathcal{K}_\infty$ functions $\underline{\alpha}_i, \bar{\alpha}_i$ such that $\underline{\alpha}_i(\text{dist}(x_i, \mathcal{X}_i^{\text{goal}})) \le V_i(x_i) \le \bar{\alpha}_i(\text{dist}(x_i, \mathcal{X}_i^{\text{goal}}))$; (2) For each i, there exists $\alpha_i > 0$ and class-$\mathcal{K}$ functions $\chi_{ij}, j \in \mathcal{N}_i$ satisfying $\chi_{ij}(a) < a, \forall a > 0$, such that $\forall x_i, x_{\mathcal{N}_i}$,*

$$V_i(x_i) \ge \max_{j \in \mathcal{N}_i} \chi_{ij}(V_j(x_j)) \quad \Rightarrow \quad [\nabla V_i(x_i)]^\top f_i(x_i, x_{\mathcal{N}_i}, \pi_i(x_i)) \le -\alpha_i \, V_i(x_i). \tag{4.2}$$

Then, the closed-loop system under controllers $\pi_1, \ldots, \pi_n$ is globally asymptotically stable around x^{goal}. Such functions $V_i(x_i), i = 1, \ldots, n$ are called ISS Lyapunov functions.

Proof. The proof follows similar steps as that in [35] and we modify it to accommodate the network structure in our setting. Recall that we use $f(x, u)$ to denote the vector of the indi-

vidual dynamical functions f_i. Further, we use $\pi(x)$ to denote $\pi(x) = [\pi_1(x_1), \ldots, \pi_n(x_n)]^\top$. With this notation, the closed-loop dynamical system can be written as

$$\dot{x} = f(x, \pi(x)).$$

Now consider the following function

$$V(x) = \max_{i \in \mathcal{N}} V_i(x_i).$$

Case 1. Suppose for a given x, $\max_{i\in\mathcal{N}} V_i(x_i)$ is uniquely achieved at $i^* \in \mathcal{N}$. Due to the continuous differentiability of the V_i's, we have $V(x') = V_{i^*}(x'_{i^*})$ for x' in a neighborhood of x, and as a result, $V(x)$ is continuously differentiable at x. Further, note that clearly $V_{i^*}(x_{i^*}) > \max_{j\in\mathcal{N}_{i^*}} V_j(x_j) \geq \max_{j\in\mathcal{N}_{i^*}} \chi_{i^*j}(V_j(x_j))$. Using the imply condition (4.2), we have,

$$\begin{aligned}
&[\nabla V(x)]^\top f(x, \pi(x)) \\
&= [\nabla V_{i^*}(x_{i^*})]^\top f_{i^*}(x_{i^*}, x_{\mathcal{N}_{i^*}}, \pi_{i^*}(x_{i^*})) \\
&\leq -\alpha_{i^*} V_{i^*}(x_{i^*}) = -\alpha_{i^*} V(x).
\end{aligned} \tag{4.3}$$

Case 2. Suppose for a given x, $\max_{i\in\mathcal{N}} V_i(x_i)$ is achieved at a set of multiple indices $\mathcal{I}$. Then, following a similar argument as Case 1 and using a continuity argument, there must exist a neighborhood $\mathcal{A}$ around x and constant $\lambda > 0$ such that for all $j \in \mathcal{I}$ and for all $y \in \mathcal{A}$,

$$[\nabla V_j(y_j)]^\top f_j(y_j, y_{\mathcal{N}_j}, \pi_j(y_j)) \leq -\lambda V_j(x_j), \tag{4.4}$$

and further, $\max_{i\in\mathcal{N}} V_i(y_i)$ must be achieved within $\mathcal{I}$.

Consider a trajectory of the system $\dot{y} = f(y, \pi(y))$ starting at $y(0) = x$. There must

exist $\delta > 0$ s.t. $y(t) \in \mathcal{A}, \forall t \in [0, \delta]$. For any t within range $[0, \delta]$, suppose $\max_{i \in \mathcal{N}} V_i(y_i(t))$ is achieved at a certain $j \in \mathcal{I}$. Then, we have,

$$\begin{aligned}
&V(y(t)) - V(x) \\
&= V_j(y_j(t)) - V_j(y_j(0)) \\
&= \int_{\tau=0}^{t} \frac{d}{d\tau} V_j(y_j(\tau)) d\tau \\
&= \int_{\tau=0}^{t} [\nabla V_j(y_j(\tau))]^\top f_j(y_j(\tau), y_{\mathcal{N}_j}(\tau), \pi_j(y_j(\tau))) d\tau \\
&\leq -\lambda \int_{\tau=0}^{t} V_j(x_j) d\tau \\
&= -\lambda V_j(x_j) t = -\lambda V(x) t
\end{aligned}$$

where in the last inequality we have used (4.4). Therefore, whenever V is differentiable at x, we can divide the above by t and let $t \to 0$ to get

$$[\nabla V(x)]^\top f(x, \pi(x)) \leq -\lambda V(x). \tag{4.5}$$

Combining the above and (4.3), we have there exists a constant $\lambda' > 0$ s.t. $[\nabla V(x)]^\top f(x, \pi(x)) \leq -\lambda' V(x)$ whenever V is differentiable. Also, it is easy to check that V is continuously differentiable almost everywhere, and satisfies

$$\underline{\alpha}(\text{dist}(x, x^{\text{goal}})) \leq V(x) \leq \bar{\alpha}(\text{dist}(x, x^{\text{goal}})),$$

for $\mathcal{K}_\infty$ functions $\underline{\alpha}$ and $\bar{\alpha}$. Lastly, we can use the same argument as in the proof of [86, Thm 3.1] to show that the trajectory of the dynamical system must converge to the goal set. □

We note that Lemma 5 is a variant of the result in [35], in that we explicitly consider the network structure in the dynamics. As a result, in the ISS implication condition (4.2), we need to test $V_i(x_i)$ versus the max of $V_j(x_j)$ over only the neighbors $\mathcal{N}_i$, as opposed to

the entire network as in [35]. This effectively makes (4.2) a condition that can be checked locally at each subsystem. One benefit of the local structure in the implication condition (4.2) is that it allows us to use certificates from smaller networks to compose certificates for larger networks that consist of blocks of the smaller networks. We will discuss this in detail in Section 4.2.2. Moreover, our results can also be robustified so a single ISS Lyapunov function can be used across different subsystems and handle uncertain parameters in the dynamics. We will explain this in detail in Section 4.2.3.

4.2.2 Network Generalizability

As discussed in Section 4.2.1, the condition (4.2) only involves f_i, V_i, and the Lyapunov functions of neighbors $\{V_j\}_{j\in\mathcal{N}_i}$. With such a local architecture, we present the following Lemma 6 that shows the decentralized controllers π_i and ISS Lyapunov functions V_i for a small system can be "ported over" to a larger dynamical system that has a similar symmetric structure to the smaller dynamical system.

Lemma 6. *Consider a networked dynamical system with node set $\mathcal{N}$, neighborhood sets $\mathcal{N}_i$, and dynamics functions f_i, and suppose there exist decentralized controllers π_i such that the closed-loop dynamical system admits a compositional certificate V_i that satisfies the conditions in Lemma 5 with parameters χ_{ij}, α_i. Suppose there is another dynamical system with node set $\tilde{\mathcal{N}}$, neighborhood sets $\tilde{\mathcal{N}}_j$, and dynamics functions $\tilde{f}_i$. Suppose for each $j \in \tilde{\mathcal{N}}$, there exists a one-to-one map $\tau_j : \{j\} \cup \tilde{\mathcal{N}}_j \to \mathcal{N}$ such that $\tau_j(\tilde{\mathcal{N}}_j) = \mathcal{N}_{\tau_j(j)}$, and $\tilde{f}_j = f_{\tau_j(j)}$. Further, suppose $\forall j, j' \in \tilde{\mathcal{N}}$, $\forall \ell \in \tilde{\mathcal{N}}_j \cap \tilde{\mathcal{N}}_{j'}$, we have $V_{\tau_j(\ell)} = V_{\tau_{j'}(\ell)}$. Then, $\tilde{\pi}_j = \pi_{\tau_j(j)}$ is a stabilizing controller for the new system with compositional certificate $\tilde{V}_j = V_{\tau_j(j)}$.*

Proof. Fix a $j \in \tilde{\mathcal{N}}$. For $j' \in \tilde{\mathcal{N}}_j$, we have

$$\tilde{V}_{j'} = V_{\tau_{j'}(j')} = V_{\tau_j(j')}. \tag{4.6}$$

Define $\tilde{\chi}_{jj'} = \chi_{\tau_j(j)\tau_j(j')}$ and $\tilde{\alpha}_j = \alpha_{\tau_j(j)}$. Suppose x_j and $x_{\tilde{\mathcal{N}}_j}$ are such that

$$\tilde{V}_j(x_j) \geq \max_{j' \in \tilde{\mathcal{N}}_j} \tilde{\chi}_{jj'}(\tilde{V}_{j'}(x_{j'})) \tag{4.7}$$

Using (4.6), we have (4.7) is equivalent to

$$V_{\tau_j(j)}(x_j) \geq \max_{j' \in \tilde{\mathcal{N}}_j} \chi_{\tau_j(j)\tau_j(j')}(V_{\tau_j(j')}(x_{j'})).$$

By (4.2), we have the above implies,

$$[\nabla V_{\tau_j(j)}(x_j)]^\top f_{\tau_j(j)}(x_j, x_{\tilde{N}_j}, \pi_{\tau_j(j)}(x_j)) \leq -\alpha_{\tau_j(j)} V_{\tau_j(j)}(x_j),$$

which is equivalent to

$$[\nabla \tilde{V}_j(x_j)]^\top \tilde{f}_j(x_j, x_{\tilde{N}_j}, \tilde{\pi}_j(x_j)) \leq -\tilde{\alpha}_j \tilde{V}_j(x_j). \tag{4.8}$$

This shows that (4.7) can imply (4.8). As such, $\tilde{V}_j$ is a valid compositional certificate for the new system. □

Example 2. *For the Platoon system, using Lemma 6, we can prove that a stabilizing controller for a 5-truck system $\pi_i, i = 1, \ldots, 5$ can be generalized to any system with $n > 5$. Let $\tilde{\mathcal{N}}$, $\tilde{\mathcal{N}}_i$, $\tilde{f}_i$ be the subsystems, neighborhood, and dynamics functions of the Platoon system with n trucks, and $\mathcal{N}$, $\mathcal{N}_i$, f_i be those of the system with 5 trucks. Note that in the Platoon system we have $\mathcal{N}_j = \{j-1, j+1\}$ and the same for $\tilde{\mathcal{N}}_j$. We define the one-to-one mapping τ_j in the following way: For the first truck, let $\tau_1(0) = 0, \tau_1(1) = 1, \tau_1(2) = 2$. For the last truck, let $\tau_n(n-1) = 4, \tau_n(n) = 5, \tau_n(n+1) = 6$. For other trucks $j = 2, 3, \ldots, n-1$, let $\tau_j(j-1) = 2, \tau_j(j) = 3, \tau_j(j+1) = 4$. Further, let $V_2 = V_3 = V_4$ and $\pi_2 = \pi_3 = \pi_4$ in the system with 5 trucks. In this way, we can check that $\forall j, j' \in \tilde{\mathcal{N}}$, $\forall \ell \in \tilde{\mathcal{N}}_j \cap \tilde{\mathcal{N}}_{j'}$, we have $V_{\tau_j(\ell)} = V_{\tau_{j'}(\ell)}$. Then following Lemma 6, we can conclude that $\tilde{\pi}_1 = \pi_1$, $\tilde{\pi}_n = \pi_5$,*

$\tilde{\pi}_j = \pi_2 = \pi_3 = \pi_4, j = 2, 3, \ldots, n-1$ *are stabilizing controllers for the new system with certificates* $\tilde{V}_1 = V_1, \tilde{V}_n = V_5$, *and* $\tilde{V}_j = V_2 = V_3 = V_4, j = 2, 3, \ldots, n-1$.

4.2.3 Robust ISS Lyapunov Functions

Many subsystems in a networked system are very similar in terms of dynamics and network structure but may have different parameters. Furthermore, system dynamics may have model uncertainties and unknown parameters [79]. For instance, in the Platoon example, trucks may have different weights, and the leading truck can have unknown velocity and acceleration, but the platoon system should be stabilized for any driving style of the leading truck. Having a robust version of ISS Lyapunov functions that can work for a set of different subsystems with different parameters can significantly reduce the number of ISS Lyapunov functions we need to find for a large networked system and also improve the robustness of the resulting controller. To tackle this, we show that the robust ISS Lyapunov functions can be established for control-affine systems taking the form: $\dot{x}_i = h_i(x_i, x_{\mathcal{N}_i}; \beta) + g_i(x_i, x_{\mathcal{N}_i}; \beta)u_i$, where $\beta \in \mathcal{B}$ is the parameter of the dynamics that models uncertainties. Such an assumption is not restrictive and can cover a large range of physical systems, e.g., systems following the manipulator function [23]. Under this assumption, we further introduce robust ISS Lyapunov functions to guarantee the global asymptotic stability of systems with uncertainties.

Lemma 7. *Given a networked dynamical system with control-affine dynamics with bounded parametric uncertainty* $\beta \in \mathcal{B}$, *where* $\mathcal{B}$ *is the convex hull of parameters* $\beta_1, \beta_2, \ldots, \beta_{n_\beta}$. *If there exists ISS Lyapunov functions* V_i *satisfying the conditions in Lemma 5 for each* $\beta_j, j \in \{1, 2, \ldots, n_\beta\}$, *the dynamics* h_i *and* g_i *are affine with respect to* β, *then the closed-loop system is globally asymptotically stable with any* $\beta \in \mathcal{B}$.

Proof. Using the control-affine dynamics, we have

$$\begin{aligned}
&[\nabla V_i(x_i)]^\top f_i(x_i, x_{\mathcal{N}_i}, \pi_i(x_i)) \\
&\quad = [\nabla V_i(x_i)]^\top \left[h_i(x_i, x_{\mathcal{N}_i}; \beta) + g_i(x_i, x_{\mathcal{N}_i}; \beta)\pi(x_i)\right] \\
&\quad = L_{h_i(\beta)} V_i(x_i) + L_{g_i(\beta)} V_i(x_i)\pi(x_i),
\end{aligned} \tag{4.9}$$

where we denote

$$\begin{aligned}
L_{h_i(\beta)} V_i(x_i) &= [\nabla V_i(x_i)]^\top h_i(x_i, x_{\mathcal{N}_i}; \beta) \\
L_{g_i(\beta)} V_i(x_i) &= [\nabla V_i(x_i)]^\top g_i(x_i, x_{\mathcal{N}_i}; \beta)
\end{aligned} \tag{4.10}$$

as the Lie derivatives of V_i along $h_i(x_i, x_{\mathcal{N}_i}; \beta)$ and $g_i(x_i, x_{\mathcal{N}_i}; \beta)$. By assumption, we have h_i and g_i are affine in β. In addition, the Lie derivatives $L_{h_i(\beta)} V_i$ and $L_{g_i(\beta)} V_i$ are affine in h_i and g_i, and Equation (4.9) is affine in $L_{h_i(\beta)} V_i$ and $L_{g_i(\beta)} V_i$. Therefore, the mapping from $\mathcal{B}$ to Equation (4.9) is affine which maps the convex hull of $\beta_1, \beta_2, \ldots, \beta_{n_\beta}$ to the convex hull of $L_{h_i(\beta_1)} V_i + L_{g_i(\beta_1)} V_i \pi_i, \ldots, L_{h_i(\beta_{n_\beta})} V_i + L_{g_i(\beta_{n_\beta})} V_i \pi_i$. As a result, if the conditions in Lemma 5 are satisfied for $\beta_i, i = 1, 2, \ldots, n_\beta$, then the conditions are satisfied for any $\beta \in \mathcal{B}$. Using Lemma 5, we can conclude that the closed-loop system is globally asymptotically stable with any $\beta \in \mathcal{B}$. □

Example 3. *The robust ISS Lyapunov functions can be directly used to the Platoon system. $v_0 = v_{n+1}$ is a parameter of this system, which is bounded between $v_{\min}$ and $v_{\max}$ by assumption. Following Lemma 7, if we can find ISS Lyapunov functions for both $v_{\min}$ and $v_{\max}$, we can ensure our system is stable with any velocity of the leading truck.*

4.3 Learning Compositional Certificates and Controllers

Based on the compositional certificate developed in Section 4.2, we now focus on jointly learning the individual ISS Lyapunov functions and the decentralized controllers. We note that while there are many existing approaches to learning neural certificates [43], [79], they can not be directly applied here because we have a unique imply condition (4.2) that can not

be handled by the existing approaches. We will introduce a novel approach to incorporate the imply condition as specially designed loss terms. To proceed, we start with formally defining the parameterization of the decentralized controllers and the ISS Lyapunov functions.

Controllers and ISS Lyapunov Functions Parameterization We focus on decentralized controllers, in which the control u_i of subsystem i only depends on the subsystem state x_i, i.e. $u_i = \pi_i(x_i; \theta_i)$. Here π_i is an NN with θ_i as the parameters. We parameterize Lyapunov function

$$V_i(x_i; S_i, \omega_i, \nu_i) = x_i^\top S_i^\top S_i x_i + p_i(x_i; \omega_i)^\top p_i(x_i; \omega_i) + q_i(x_i; \nu_i). \tag{4.11}$$

where $S_i \in \mathbb{R}^{d_i \times d_i}$ is a matrix of parameters, $p_i(x_i; \omega_i)$ is an NN with weights ω_i, and $q_i(x_i; \nu_i)$ is another NN with weights ν_i and ReLU as the output activation function, which is only applied to the output of the NN. The first term in $V_i(x_i)$ is a quadratic term to capture the linear part of the non-linear dynamics. The second term is a sum-of-squares term to capture the polynomial part of the dynamics. Finally, the third term is used to model the residues. Using this form, the ISS Lyapunov function satisfies $V_i(x_i) \geq 0$ by construction.

Sharing ISS Lyapunov Across Subsystems Following Lemma 6 and Lemma 7, to reduce the number of neural networks and to make the ISS Lyapunov functions learned in small-scale networked system generalizable to large-scale systems, we use the following weight sharing technique. We let the subsystems share the same ISS Lyapunov functions if their dynamics are similar. In addition, we can also let similar subsystems share the same controller. In this way, the number of trainable parameters can be reduced, and we can easily apply the controllers trained in small-scale systems to large-scale systems. For example, in the previous Platoon system, we can let the j-th truck, $j = 2, 3, \ldots, n-1$, share the same ISS Lyapunov function and the same controller.

Gain Function Parameterization We use linear functions to model the gain functions χ_{ij} in Equation (4.2). We let $\chi_{ij}(x) = \chi_i(x; k_i) = \text{Sigmoid}(k_i)x, \forall j$, where k_i is a trainable parameter, x is the scalar input of the gain functions, which is always the output of the ISS Lyapunov functions. In this way, the condition $\chi_{ij}(x) < x, \forall x > 0$ is satisfied by construction.

Loss Functions A key challenge in learning ISS Lyapunov functions is how to ensure condition (4.2) is satisfied. We now propose a methodology that promotes (4.2). Let Boolean $A_i, B_i \in \{0, 1\}$ be[1]

$$A_i = V_i(x_i) \geq \max_{j \in \mathcal{N}_i} \chi_i(V_j(x_j)), \quad B_i = [\nabla V_i(x_i)]^\top f_i(x_i, x_{\mathcal{N}_i}, \pi_i(x_i)) \leq -\alpha_i V_i(x_i). \tag{4.12}$$

Then condition (4.2) can be written as $A_i \Rightarrow B_i$, which is the same as $\neg A_i \vee B_i$, or $\max\{\neg A_i, B_i\}$. However, this kind of formulation is not trainable for neural networks because Boolean variables are not differentiable. To settle this problem, we introduce the following losses:

$$\mathcal{L}_{A_i} = \text{ReLU}\left(V_i(x_i) - \max_{j \in \mathcal{N}_i} \chi_i(V_j(x_j)) + \epsilon_A\right), \tag{4.13}$$

$$\mathcal{L}_{B_i} = \text{ReLU}\Big([\nabla V_i(x_i)]^\top f_i(x_i, x_{\mathcal{N}_i}, \pi_i(x_i)) + \alpha_i V_i(x_i) + \epsilon_B\Big), \tag{4.14}$$

where ϵ_A and ϵ_B are small parameters that encourages strict satisfactions and generalization abilities [79]. $\mathcal{L}_{A_i}$ and $\mathcal{L}_{B_i}$ can address the problem introduced by Boolean variables A_i and B_i, but they introduce a new problem that $\max\{\neg A_i, B_i\}$ cannot be written as $\max\{\mathcal{L}_{A_i}, \mathcal{L}_{B_i}\}$ since the two losses are not comparable. To address this issue, we minimize the loss $\mu_{A_i}\mathcal{L}_{A_i} + \mu_{B_i}\mathcal{L}_{B_i}$ instead, where μ_{A_i} and μ_{B_i} are two hyper-parameters for balancing the two losses. Note that in practice, since we often discretely simulate the dynamical systems, we can use two ways to calculate $\nabla V_i(x_i)$ in $\mathcal{L}_{B_i}$. First, we can directly calculate the gradient of

[1]For notational simplicity, from now on we omit all the notations of parameters in the function approximators.

$V_i(x_i)$ w.r.t. x_i. For the second method, we can just do a one-step simulation $x_i \to x_i^{\text{next}}$, and approximate $[\nabla V_i(x_i)]^\top f_i(x_i, x_{\mathcal{N}_i}, \pi_i(x_i))$ with $(V_i(x_i^{\text{next}}) - V_i(x_i))/\Delta t$, where Δt is the simulation time step.

To ensure the condition $V_i(x_i) = 0$ for $x_i \in \mathcal{X}_i^{\text{goal}}$ is satisfied, we introduce another loss term $\frac{1}{|\widehat{\mathcal{X}}_i^{\text{goal}}|} \sum_{x_i^{\text{goal}} \in \widehat{\mathcal{X}}_i^{\text{goal}}} |V_i(x_i^{\text{goal}})|$, where $\widehat{\mathcal{X}}_i^{\text{goal}}$ is a randomly sampled set of states from $\mathcal{X}_i^{\text{goal}}$. In addition, we add $\|\pi_i(x_i) - u_i^{\text{nominal}}\|^2$ to the loss, where u_i^{nominal} is the control signal calculated by some nominal controller. We use the Droop controller and LQR controller in our experiments. We add the nominal controller so that the learned controller can explore the "informed region" near the nominal control signal rather than randomly, to accelerate the training. We do not need the nominal controller to be stable or optimal, and the learned controller behaves much better than the nominal controller as shown in Section 4.4. The final loss function used in training is

$$\mathcal{L} = \sum_{i=1}^{n} \left[\frac{1}{|\widehat{\mathcal{X}}_i^{\text{goal}}|} \sum_{x_i^{\text{goal}} \in \widehat{\mathcal{X}}_i^{\text{goal}}} |V_i(x_i^{\text{goal}})| + \mu_{A_i} \mathcal{L}_{A_i} + \mu_{B_i} \mathcal{L}_{B_i} + \mu_{\text{ctrl}} \|\pi_i(x_i) - u_i^{\text{nominal}}\|^2 \right], \tag{4.15}$$

where μ_{ctrl} is a tuning parameter, and the training parameters are $\theta, S, \omega, \nu, k$.

Training Procedure During training, we draw samples by randomly sampling states in the state space and in the goal set. We first initialize the controller by minimizing the loss $\|\pi_i(x_i) - u_i^{\text{nominal}}\|^2$, and then fix the controller to initialize the ISS Lyapunov function by minimizing the loss $\mathcal{L}_{B_i}$. After the initialization, we minimize loss (4.15) to train the controllers, the ISS Lyapunov functions, and the gain functions jointly.

4.4 Experiments

We demonstrate NeurISS in 3 environments including Power system, Platoon, and Drone, aiming to answer the following questions:

1. How does NeurISS compare with other algorithms in the case of stabilizing networked systems?

2. Can NeurISS perform similarly or surpass the centralized controllers in small-scale networked systems?

3. Can NeurISS scale up to large-scale networked systems?

Our experiments are run on a 64-core AMD 3990X CPU @ 3.60GHz and four NVIDIA RTX A4000 GPUs (one GPU for each training job).

Baselines We compare NeurISS with both centralized and decentralized baselines. For centralized ones, we compare with the state-of-the-art RL algorithm PPO [48], the RL-with-Lyapunov-critic algorithm LYPPO [18], and the centralized Neural CLF controller (NCLF) [79]. For decentralized ones, we compare with the classical LQR [99] controller and the multi-agent RL algorithm MAPPO [100]. We hand-craft reward functions based on the common way of designing reward functions for tracking problems for the RL algorithms. For LQR, since the agents only have local observations, we calculate the goal point for the LQR controller based on local observations in each time step.

4.4.1 Environment Descriptions

Networked Microgrid We consider the networked microgrid system introduced by [101], which is shown in Figure 4.1. In this environment, a power distribution network is divided into 5 regions (MG1, MG2, ..., MG5 in Figure 4.1). Each region i functions as a microgrid and two microgrids are neighbors when their corresponding regions are connected by a power line. Each microgrid i has two states $x_i = (\delta_i, E_i)$ where δ_i means the voltage phase angle

and E_i is the voltage magnitude. The dynamics of microgrid i is given by

$$M_{a,i}\dot{\delta}_i + (\delta_i - \delta_i^{\text{ref}}) = u_i^P + D_{a,i}(P_i^{\text{ref}} - G_{ii}E_i^2 - \sum_{j\in\mathcal{N}_i} E_i E_j Y_{ji}\cos(\delta_j - \delta_i - \sigma_{ji})), \tag{4.16a}$$

$$M_{v,i}\dot{E}_i + (E_i - E_i^{\text{ref}}) = u_i^Q + D_{v,i}(Q_i^{\text{ref}} + B_{ii}E_i^2 - \sum_{j\in\mathcal{N}_i} E_i E_j Y_{ji}\sin(\delta_j - \delta_i - \sigma_{ji})), \tag{4.16b}$$

where $M_{a,i}$ and $M_{v,i}$ are innertia coefficients, $D_{a,i}$ and $D_{v,i}$ are droop coefficients to provide internal droop controls, $\delta_i^{\text{ref}}, E_i^{\text{ref}}, P_i^{\text{ref}}, Q_i^{\text{ref}}$ are precomputed reference values, $B_{ii}, G_{ii}, Y_{ji}, \sigma_{ji}$ are coefficients from the network admittance matrix. For a detailed description of the model, see [101]. Note that compared to [101], we also introduce the control input $u_i = (u_i^P, u_i^Q)$ which represents the active and reactive power produced by the secondary control, and the goal is to design secondary controllers so that δ_i, E_i can converge to their reference values $\delta_i^{\text{ref}}, E_i^{\text{ref}}$.

During training, the training data are sampled from $\delta_i \in [-3, 3]$ and $E_i \in [-3, 3]$, and the data in the goal region are sampled from the region where $\delta_i = \delta_i^{\text{ref}}$ and $E_i = E_i^{\text{ref}}$. During testing, we set the simulation time interval $\Delta t = 0.01$, and randomly sample the initial states of the microgrids in $\delta_i \in [-2, 2]$ and $E_i \in [-3, 3]$. The number of simulation time steps is 500.

Distribution grid voltage control We consider a power system voltage control problem given in [102], which we name GridVoltage8. There is a power distribution network as a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, which consists of a set of nodes $\mathcal{N} = \{1, \ldots, n\}$ and edges $\mathcal{E}$. Each node $i \in \mathcal{N}$ is associated with a reactive power injection q_i, and a voltage magnitude v_i. We use

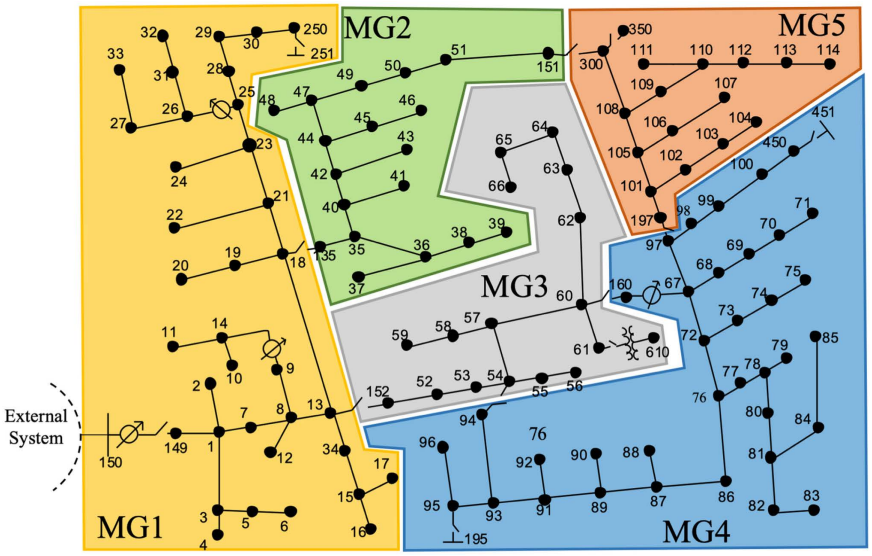


Figure 4.1: IEEE 123-node Test Feeder [101]

q and v to denote the q_i, v_i stacked into a vector. The system dynamics are given as follows

$$\dot{q}(t) = u(t) = \pi(v(t)) \tag{4.17}$$

$$v(t) = v(q(t)) \tag{4.18}$$

where the state is the reactive power $q(t)$, and the control action is the change rate of $q(t)$. Critically, the voltage $v(t)$ is a function of the reactive power $q(t)$, denoted as $v(t) = v(q(t))$, and this function is defined implicitly via the solution of a nonlinear algebraic equation system known as the power-flow equation [103]. In our experiment, we use PandaPower [104] as the powerflow solver. The goal is to design a controller $u(t) = \pi(v(t))$ where the control action $u(t)$ depends on the voltage $v(t)$ such that in the close loop system, the voltage $v(t)$ across all nodes will converge to the nominal value (which is 1.0). In other words, the goal points of the problem consist of the set of reactive power such that the voltage is 1.0.

In our experiments, we consider a power distribution system that consists of 8 buses, see Figure 4.2. The buses are arranged in a line, each one is connected to a static generator. The nominal controller is similar to the droop control and is a proportional controller on the voltage deviation, $u_i(t) = c_i(v_i(t) - 1)$ (where c_i is a constant). This proportion controller is

a standard controller used in practice.

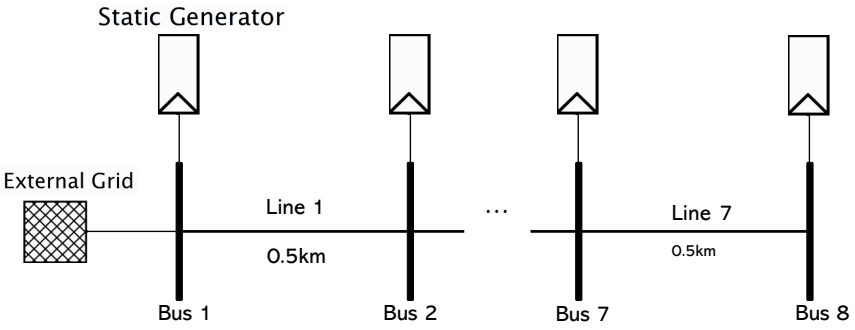


Figure 4.2: GridVoltage8

During training, the training data are sampled from $q_i \in [-0.001, 0.001]$, since Pandapower will give no solution when the variance of the training data is too large, in this case, we manually set the output voltage $v_i(t)$ to 0.9 for negative states $q_i(t)$ and 1.1 for positive ones. During testing, we set the simulation time interval $\Delta t = 0.01$, and sample the initial states of the power system from states of all 0. The number of simulation time steps is 500.

Platoon A platoon system is shown in Figure 4.3, which contains a list of trucks. The green truck is the leading truck (the 0-th truck), which can drive freely, and the orange truck is the last truck (the $(n+1)$-th truck). We want to control the trucks in the middle (the black trucks) so that the trucks in the whole truck platoon system are spread evenly. As mentioned in the main pages, the state of the i-th truck is defined as $x_i = [p_i^f, p_i^b, v_i]^\top$, where p_i^f is the distance between the i-th truck and the $(i-1)$-th truck, p_i^b is the distance between the i-th truck and the $(i+1)$-th truck, and v_i is the velocity of the i-th truck. The state of the i-th truck is shown in Figure 4.3. For each relative position of the leading and last trucks, the goal set for each truck $i \in \{1, 2, ..., n\}$ is uniquely defined as the set of states satisfying $p_i^f = p_i^b$, which means that each middle truck aims to keep the distance to the front and behind trucks the same.

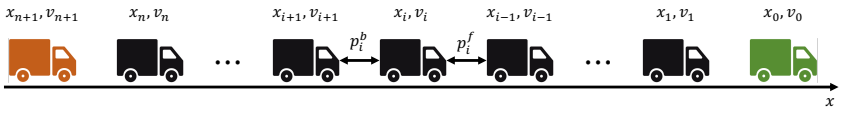


Figure 4.3: Platoon

During training, the training data are sampled from $p_i^f \in [0, 2]$, $p_i^b \in [0, 2]$, $v_i \in [0, 4]$, and the data in the goal region are sampled from the region where $p_i^f = p_i^b$. During testing, the leading truck follows a trajectory with initial velocity 2.0 and acceleration $\sin(5t\Delta t)$, where t is the simulation time step and $\Delta t = 0.01$ is the simulation time interval. Therefore, the velocity of the leading truck follows the profile shown in Figure 4.4. We randomly sample the controlled trucks' initial states in $p_i^f \in [0.6, 1.4]$, $p_i^b \in [0.6, 1.4]$, and $v_i \in [1.0, 1.2]$, so that the trucks need to speed up first to catch the leading truck. The number of simulation time steps is 500.

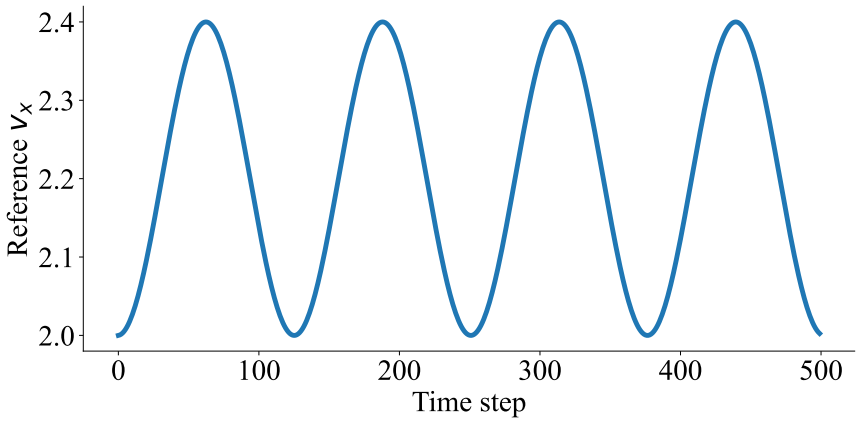


Figure 4.4: The velocity profile of the leading truck of the platoon system

Planar Drone Formation Control A Planar Drone is shown in Figure 4.5 [23]. For a single drone, the state is given by $x = [p_x, p_y, \theta, v_x, v_y, \omega]\top$, where (p_x, p_y) is the position, (v_x, v_y) is the velocity, θ is shown in Figure 4.5, and ω is the changing rate of θ. The control inputs are forces generated by the two propellers u_1, u_2 shown in Figure 4.5. The dynamics

of the drone is given by $\dot{x} = h(x) + g(x)u$, where

$$h(x) = \begin{bmatrix} v_x \\ v_y \\ \omega \\ 0 \\ -g \\ 0 \end{bmatrix}, \tag{4.19}$$

and

$$g(x) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -\frac{\sin\theta}{m} & -\frac{\sin\theta}{m} \\ \frac{\cos\theta}{m} & \frac{\cos\theta}{m} \\ \frac{r}{I} & -\frac{r}{I} \end{bmatrix}, \tag{4.20}$$

where m, I, r is the mass, moment of inertia, and the distance from the center to the base of the propeller, respectively.

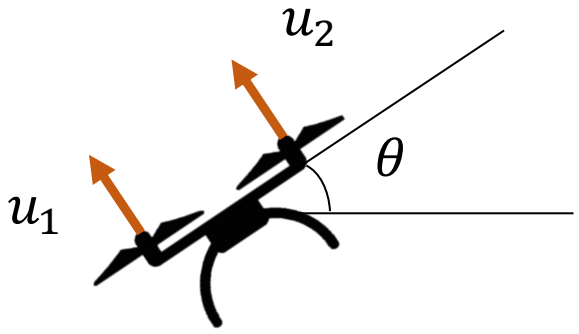


Figure 4.5: Planar Drone Dynamics [23]

For the formation control task, we design the states of the drones to be a 2-D platoon-like system, given by $x_i = [p_i^l, p_i^r, p_i^u, p_i^d, \theta_i, v_i^x, v_i^y, \omega_i]^\top$, where $p_i^l, p_i^r, p_i^u, p_i^d$ are the distances from drone i to the left, right, up, down drones. Other dimensions of the state are not coupled

and are the same as the single drone system. The control inputs of each drone are the same as the single drone system.

During training, the training data are sampled from $p_i^l, p_i^r, p_i^u, p_i^d \in [0, 5]$, $\theta_i \in [-\pi/2, \pi/2]$, $v_i^x \in [-7, 7]$, $v_i^y \in [-5, 5]$, $\omega_i \in [-\pi/2, \pi/2]$, and the data in the goal region are sampled from the region where $p_i^l = p_i^r, p_i^u = p_i^d$, and $\theta_i = \omega_i = 0$. During testing, the trajectory that we want the drones to track follows the profile with initial velocity 1.0, and acceleration $0.5\sin(t\Delta t) - 0.25$, and the velocity is clipped below by 0.5. Here t is the simulation time step, and $\Delta t = 0.03$ is the simulation time interval. Therefore, the velocity of the tracking trajectory follows the profile shown in Figure 4.6. We randomly sample the drones' initial states in $p_i^l, p_i^r \in [0.8, 1.2]$, $p_i^u, p_i^d \in [0.09, 0.11]$, $v_i^x \in [0.85, 1.15]$, $v_i^y \in [-0.15, 0.15]$, $\theta_i \in [-0.05, 0.05]$, $\omega_i \in [-0.05, 0.05]$, so the drones need to first rise up and then follow the desired trajectory. The number of simulation time steps is 500.

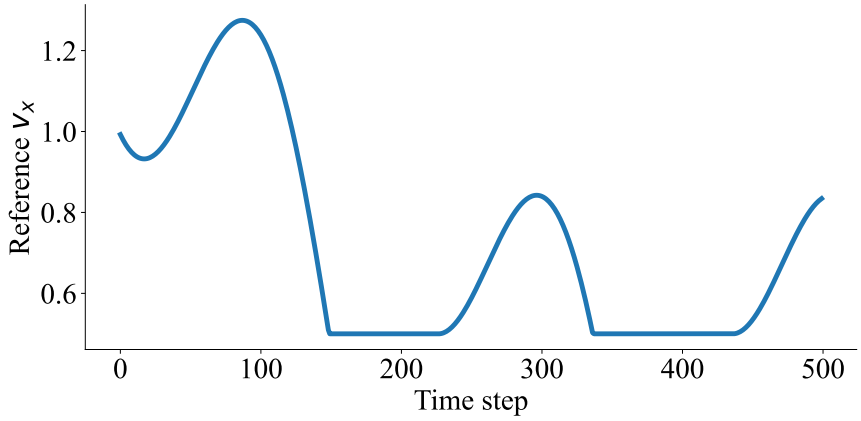


Figure 4.6: The velocity profile of the tracking trajectory of the planar drone formation control system

4.4.2 Implementation Details

Implementation of NeurISS

In our framework, there are two models to be trained: the neural ISS Lyapunov functions $V_i(x_i; S_i, \omega_i, \nu_i)$ and the controllers $\pi_i(x_i; \theta_i)$, where

$$\begin{aligned} V_i(x_i; S_i, \omega_i, \nu_i) = x_i^\top S_i^\top S_i x_i \\ + p_i(x_i; \omega_i)^\top p_i(x_i; \omega_i) + q_i(x_i; \nu_i), \end{aligned} \tag{4.21}$$

and $\pi_i(x_i; \theta_i)$ is a fully-connected multi-layer perceptron (MLP). $S_i \in \mathbb{R}^{d_i \times d_i}$ is a matrix of trainable parameters. The neural networks $p_i(x_i; \omega_i)$ and $\pi_i(x_i; \theta_i)$ are MLPs with two hidden layers with size 64 and Tanh as the hidden activation function. $q_i(x_i; \nu_i)$ is an MLP with two hidden layers with size 64, Tanh as the hidden activation function, and ReLU as the output activation function. To control the Lipschitz of the neural networks, we add the spectral normalization mechanism [92] on every hidden layer of the neural networks. Our framework is implemented in PyTorch [93] framework with ADAM [89] as the optimizer.

In our framework, we include the hyper-parameters α_i, ϵ_A, ϵ_B, μ_{A_i}, μ_{B_i}, μ_{ctrl}. For simplicity, we omit the coefficient of the first term μ_{goal} in loss (4.15) in the main pages. We also include this hyper-parameter here. Note that omitting this coefficient in the main pages does not cause any problem because the coefficients μ_{goal}, μ_{A_i}, μ_{B_i}, μ_{ctrl} only controls the weight of each loss. If we set $\mu_{\text{goal}} = 1$ manually and divide other coefficients and the learning rate with μ_{goal}, we can get the same results. The exact values of the hyper-parameters are included in Table 4.1.

We further discuss the function of each hyper-parameter. α_i is the convergence rate of the ISS Lyapunov function. Larger α_i can make the closed-loop system converge faster to the goal, but it also makes the training harder. ϵ_A and ϵ_B are used to encourage strict satisfactions of loss (4.13) and (4.14), and to encourage the generalization abilities [79]. Larger ϵ_A and ϵ_B make the learned ISS Lyapunov functions have better generalization abilities but

Table 4.1: Hyper-parameters used during training NeurISS

Hyper-parameter	α_i	ϵ_A	ϵ_B	μ_{goal}	μ_{A_i}	μ_{B_i}	μ_{ctrl}
Microgrid	0.5	1.0	1.0	10	0.1	50.0	0.0
GridVoltage8	0.5	1.0	1.0	100	0.01	50.0	1.0
Platoon	1.0	1.0	1.0	100	0.1	50	0.001
PlanarDrone	0.2	1.0	1.0	100	0.01	3.0	0.2

also make the training harder. μ_{goal}, μ_{A_i}, μ_{B_i} are weights of different terms in the total loss (4.15). We choose them by balancing the value of each term to be at similar order of magnitudes. μ_{ctrl} controls the strength of the additional training signal of $\pi_i(x_i;\theta_i)$. If the nominal controller is good, we can use large μ_{ctrl} to accelerate the training, while if the nominal controller behaves badly, we use smaller μ_{ctrl} or even set $\mu_{\text{ctrl}} = 0$ to make sure that the nominal controller will not affect $\pi_i(x_i;\theta_i)$ too much.

During training, we set the batch size to be 2048, except for GridVoltage8 where the batch size is 1024, and train NeurISS for 10000 iterations. We set the learning rate to be 3×10^{-4} for the ISS Lyapunov functions $V_i(x_i; S_i, \omega_i, \nu_i$, 5×10^{-4} for the controllers $\pi(x_i;\theta_i)$, and 10^{-3} for the coefficients k_i for $\chi_i(x_i;k_i)$. To prevent overfitting, we add the weight decay mechanism with coefficient 10^{-3} for the ISS Lyapunov functions and the controllers. We train NeurISS and the baselines 4 times with random seeds $0, 1, 2, 3$.

Implementation of the Baselines

Our baselines include the droop controller, LQR [99], PPO [48], LYPPO [18], MAPPO [100], and neural CLF [79]. The droop controller is only used in Microgrid5 and GridVoltage8. There is an internal droop controller given with the IEEE 123-node test feeder [101] so we directly use that controller as one of the baselines. The droop controller used in GridVoltage8 is similar to a proportion controller on the voltage deviation, $u_i(t) = c_i(v_i(t) - 1)$ (where c_i is a constant), this proportion controller is a standard controller used in practice. The LQR controller is used in the truck platoon system and the planar drone formation control system because these two systems are separable. We linearize the dynamics of the single truck and

Environment	Microgrid5	GridVoltage8	Platoon5	Drone2x2
NeurISS	2027.25 ± 18.15	$\mathbf{2483.70} \pm 1.68$	$\mathbf{2054.89} \pm 95.84$	$\mathbf{1713.73} \pm 13.35$
LQR	——	——	1894.64 ± 5.20	1209.15 ± 5.27
Droop	1431.30 ± 55.03	2251.70 ± 0.00	——	——
PPO	1970.89 ± 43.97	1086.72 ± 1.13	1489.32 ± 125.24	1489.32 ± 125.24
LYPPO	$\mathbf{2234.58} \pm 34.92$	1086.27 ± 2.12	707.08 ± 300.61	41.18 ± 6.05
MAPPO	1934.72 ± 149.66	1086.06 ± 0.27	1880.80 ± 155.49	1549.06 ± 271.32
NCLF	1887.36 ± 26.98	1078.19 ± 660.76	1979.02 ± 8.03	871.87 ± 46.02

Table 4.2: The expected reward of NeurISS and the baselines in the small-scale environments

the single planar drone to calculate the LQR controller for them. PPO is implemented based on the open-source python package stablebaselines[2] [94]. LYPPO is implemented based on the official implementation. MAPPO is implemented based on the official implementation[3] [100]. Neural CLF is implemented based on the official implementation[4] [79].

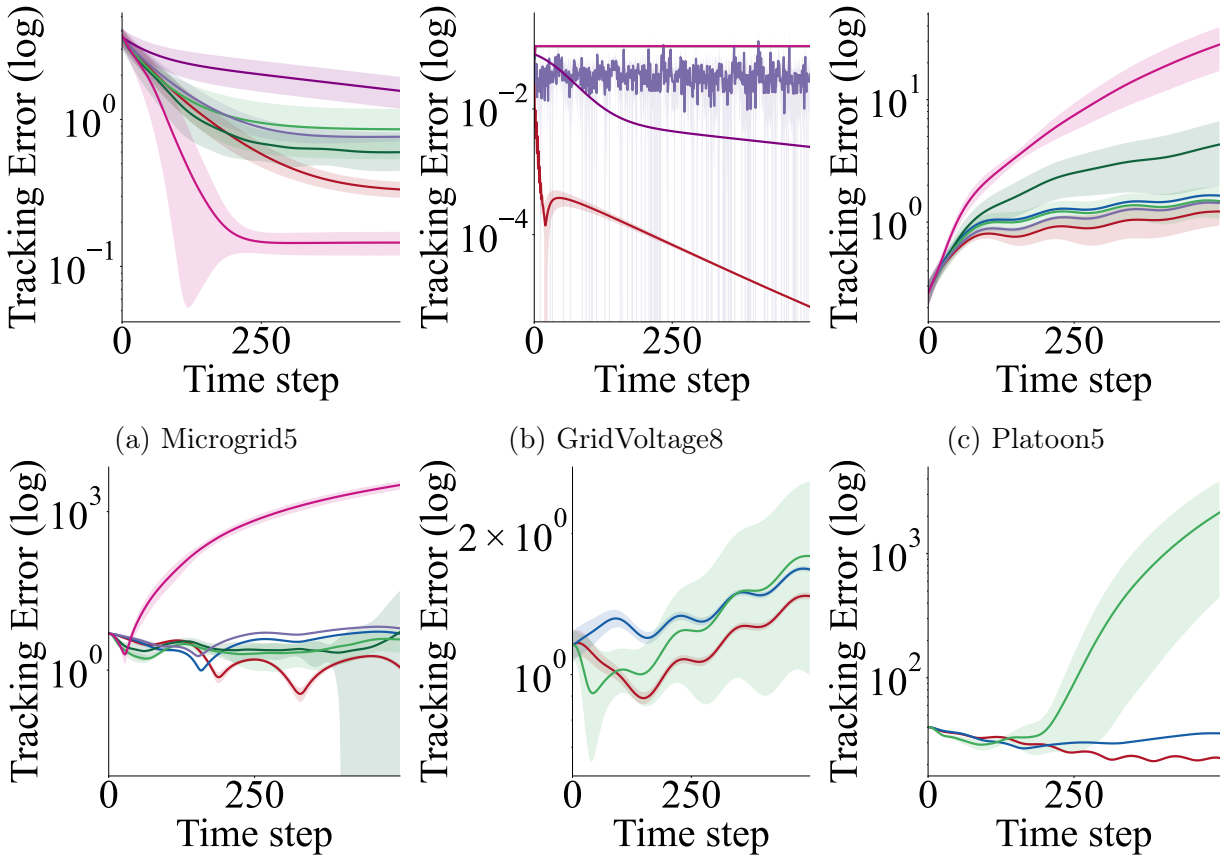
4.4.3 Results

The results show that compared with the baselines, NeurISS can achieve comparable or better rewards and tracking errors in small-scale environments, and significantly higher rewards and lower tracking errors in large-scale environments, which demonstrates its efficacy and generalizability.

Small-scale Experiments

In Table 4.2, we show the expected rewards and standard deviations of NeurISS and the baselines, and in Figure 4.7 (a-d) we show the tracking error w.r.t. the simulation time steps. We can observe that in the small-scale system Microgrid5 (10 dimensions), NeurISS achieves the second highest expected reward, and second lowest tracking error, while LYPPO behaves the best. In GridVoltage8 (8 dimensions) and larger systems Platoon5 and Drone2x2 (15 and 24 dimensions), NeurISS achieves the highest expected rewards and the lowest tracking

[2]https://github.com/DLR-RM/stable-baselines3
[3]https://github.com/marlbenchmark/on-policy
[4]https://github.com/MIT-REALM/neural_clbf



(a) Microgrid5 (b) GridVoltage8 (c) Platoon5

(d) Drone2x2 (e) Platoon100 (f) Drone10x10

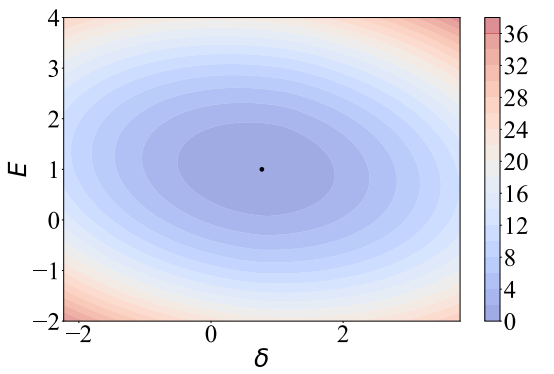
Figure 4.7: The tracking error in log scale w.r.t. time step of NeurISS and the baselines in the small-scale environments (*a-d*) and the large-scale environments (*e-f*). The line shows the mean tracking error while the shaded region shows the standard deviation.

error. Note that in Platoon5 and Drone2x2, we do not have full knowledge of the tracking trajectory, and the trajectory changes fast, so the tracking error cannot converge to 0. NeurISS has this performance because of its ability to learn decentralized controllers *jointly* with the ISS Lyapunov functions as certificates. Compared with NCLF, NeurISS performs better because it is hard to find a global CLF for networked systems. PPO and MAPPO achieve lower rewards than NeurISS.[5] They are policy gradient methods to approximate the solution of the Bellman equation, so there is no certificate of their stability. LYPPO, although

[5]For GridVoltage8, PPO, MAPPO, and LYPPO have almost identically bad rewards and tracking errors. This is because we use a professional solver PandaPower [104] to simulate the underlying distribution grid. All three RL methods return controllers that quickly drive the system to an unsafe state, making the solver fail to solve the underlying model. In such cases, we simply set the reward and the tracking error assuming a 10% voltage deviation, which is a typical safety limit for real-world distribution grids [102].

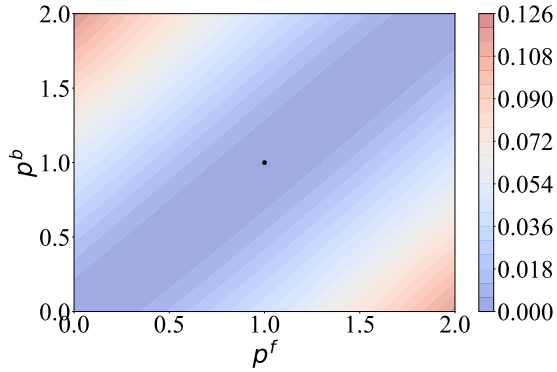
outperforms NeurISS in very small systems (Microgrid), its performance drops a lot in larger systems (Platoon and Drone). This is because in small-scale systems, with the guidance of CLF, RL can achieve the goal very quickly to maximize the cumulative reward, but NeurISS only seeks to reach the goal without targeting the convergence speed. Therefore, NeurISS convergences slower than LYPPO. However, in larger scale systems, because of the hardness of finding a correct global CLF, LYPPO receives the wrong guidance by the wrong CLF and thus behaves much worse than NeurISS and even PPO and MAPPO. For the nominal controllers, LQR is designed for linear systems, and the Droop controllers are hand-tuned. Therefore, their performance is hard to guarantee in complex nonlinear networked systems.

We provide the contour plots of the learned ISS Lyapunov functions in Figure 4.8. In the networked microgrid environments, we train only one ISS Lyapunov function for all the subsystems because of its robustness. The contour plot of the learned ISS Lyapunov function is shown in Figure 4.8a. We can observe that the learned ISS Lyapunov functions are in ellipse-like shapes indicating that the learned controller can go downhill w.r.t. the learned ISS Lyapunov functions and converge to the goal points (the black dots). In the networked GridVoltage8 environments, we train 8 ISS Lyapunov functions, one for each subsystem. The curve plots of the learned ISS Lyapunov function are shown in Figure 4.8b. Since we have only one input state for each subsystem, the learned ISS Lyapunov functions are in quadratic-like shapes, and the learned controller can go downwards w.r.t. the learned ISS Lyapunov functions and converge to the goal points (the black dots). In the truck platoon environment, we train two ISS Lyapunov functions. One for the first and the last controllable truck ($i = 1$ and $i = n$), and another one for all other trucks ($i = 2, 3, \ldots, n-1$). The contour plots of the learned ISS Lyapunov functions are shown in Figure 4.8c and Figure 4.8d. For trucks with $i = 1, n$, the learned ISS Lyapunov functions will make them converge to the line where $p^f = p^b$. For trucks with $i = 2, 3, \ldots, n-1$, the learned ISS Lyapunov functions are sheer under the line $p^f = p^b$ and flat above that line. This is because the environment is parameterized by the velocity of the leading truck v_0, which is large compared with the

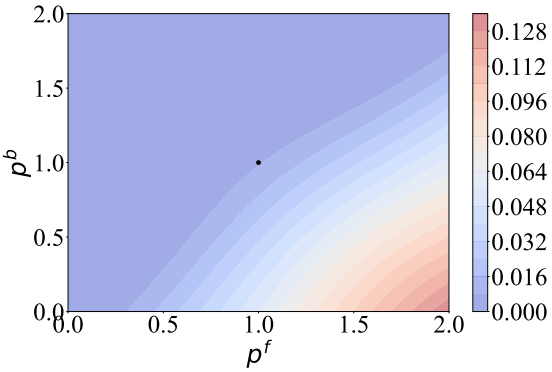


(a) Microgrid5

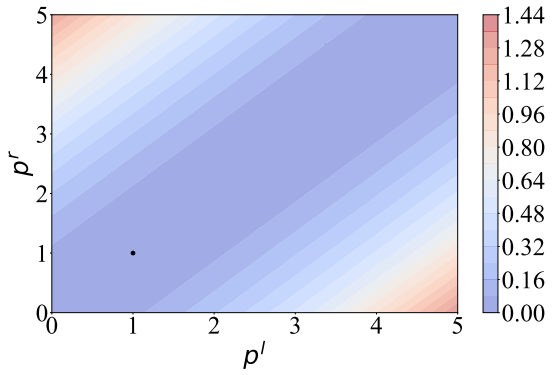
(b) GridVoltage8,$i = 1, ..., 8$



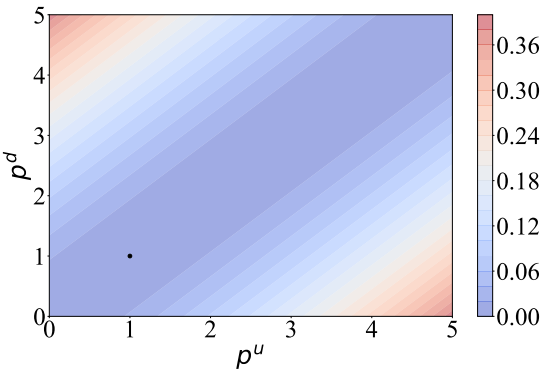
(c) Platoon5, $i = 1, 5$



(d) Platoon5, $i = 2, 3, 4$



(e) PlanarDrone2x2 (p_l v.s. p_r)



(f) PlanarDrone2x2 (p_u v.s. p_d)

Figure 4.8: Contour plots of the learned ISS Lyapunov functions, where the black dots show the goal point

initial velocity of other trucks. This causes that for the trucks in the middle, it is more often for them to reach the states where $p^f > p^b$, the region under $p^f = p^b$. Therefore, when the trucks are in states $p^f > p^b$, the sheer ISS Lyapunov functions will give them a strong signal to return to the line $p^f = p^b$. However when the trucks are in states $p^f < p^b$, they do not have to return to the $p^f = p^b$ quickly because the large v_0 can make them return automatically. In the drone formation control environment, we still train only one ISS Lyapunov function for all the subsystems because of its robustness. The learned ISS Lyapunov function is shown in Figure 4.8e and Figure 4.8f, which can make the drone converge to the states where $p^l = p^r$ and $p^u = p^d$.

Large-scale Experiments

One key advantage of the proposed framework is network generalizability (Section 4.2.2), where the decentralized controllers and ISS Lyapunov functions trained in small networked systems can be directly applied to large networked systems without further training, while the centralized controllers need a really long time to be trained on large-scale systems (Approximately 250 hours for Drone10x10). We test the 3 decentralized approaches, NeurISS, MAPPO, and LQR in large-scale Platoon and Drone systems, Platoon100 and Drone10x10, with 100 trucks and $10 \times 10 = 100$ drones. We show the tracking errors w.r.t. the simulation time steps in Figure 4.7 (e-f). We observe that NeurISS has the smallest tracking errors in both environments, with large gaps to others, which shows that NeurISS has the strongest scalability.

4.5 Conclusion

In this chapter, we propose a neural compositional certificate framework for stabilizing large-scale networked dynamical systems. Limitations of the approach include: 1) the approach requires the knowledge of the dynamical system functions f_i; 2) the network generalizability

result, Lemma 6, requires a strong symmetric condition; 3) the robust ISS Lyapunov result Lemma 7 assumes the dynamical system is control affine; 4) the approach only learns a compositional certificate using finite samples but does not verify it, so in some sense, the ISS Lyapunov functions we learn are only candidate ISS Lyapunov functions. These limitations are all interesting future directions.

Chapter 5

Conclusion and Future Work

In this thesis, we study developing stabilizing controllers for real-world systems, particularly focusing on two types of challenging systems: high-dimensional unknown systems and networked dynamical systems. For the first type of system, we present the LYGE algorithm in Chapter 3, which uses the learned CLF to guide us to explore the high-dimensional state space, fit the local dynamics, and learn a stabilizing controller iteratively. Upon convergence, the learned controller stabilizes the closed-loop system at the goal point. For the second type of system, we present the NeurISS algorithm in Chapter 4, which jointly learns an ISS Lyapunov function and decentralized controllers for subsystems, and can share controllers between systems if the systems have structural symmetry. This allows us to control each subsystem in a fully decentralized way while having global stability. We provide extensive experimental results to demonstrate the proposed algorithm outperforms other state-of-the-art baselines in the particular type of systems they are designed for.

Apart from stability, safety is another crucial desired property for real-world controllers. In future work, we would like to explore developing safe controllers for real-world systems, especially for large-scale multi-agent systems. We would like to explore the utilization of the Control Barrier Function (CBF), an extension of CLF, and design algorithms to find safe controllers based on the guidance of CBF. In addition, RL has been shown to be powerful

in solving multi-agent control problems. However, the reward function design and safety requirements are challenging for RL algorithms. In future work, we would like to explore the possibility of combining certificate functions (CBF, CLF) with RL to address the reward function design and safety problem.

References

[1] P. Heidlauf, A. Collins, M. Bolender, and S. Bak, "Verification challenges in f-16 ground collision avoidance and other automated maneuvers.," in *ARCH@ ADHS*, 2018, pp. 208–217.

[2] C. Zhao, U. Topcu, N. Li, and S. Low, "Design and stability of load-side primary frequency control in power systems," *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1177–1189, 2014.

[3] P. Varaiya, "Max pressure control of a network of signalized intersections," *Transportation Research Part C: Emerging Technologies*, vol. 36, pp. 177–195, 2013.

[4] S. Zhang and C. Fan, "Learning to stabilize high-dimensional unknown systems using lyapunov-guided exploration," *arXiv preprint arXiv:2306.08722*, 2023.

[5] S. Zhang, Y. Xiu, G. Qu, and C. Fan, "Compositional neural certificates for networked dynamical systems," in *Learning for Dynamics and Control Conference*, PMLR, 2023, pp. 272–285.

[6] E. R. Westervelt, J. W. Grizzle, C. Chevallereau, J. H. Choi, and B. Morris, *Feedback control of dynamic bipedal robot locomotion*. CRC press, 2018.

[7] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE access*, vol. 8, pp. 58 443–58 469, 2020.

[8] P. A. Parrilo, *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. California Institute of Technology, 2000.

[9] P. Giesl and S. Hafstein, "Review on computational methods for lyapunov functions," *Discrete & Continuous Dynamical Systems-B*, vol. 20, no. 8, p. 2291, 2015.

[10] C. Dawson, S. Gao, and C. Fan, "Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods," *arXiv preprint arXiv:2202.11762*, 2022.

[11] A. Majumdar, A. A. Ahmadi, and R. Tedrake, "Control design along trajectories with sums of squares programming," in *2013 IEEE International Conference on Robotics and Automation*, IEEE, 2013, pp. 4054–4061.

[12] A. A. Ahmadi and A. Majumdar, "Some applications of polynomial optimization in operations research and real-time decision making," *Optimization Letters*, vol. 10, no. 4, pp. 709–729, 2016.

[13] 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.

[14] F. Permenter and P. Parrilo, "Partial facial reduction: Simplified, equivalent sdps via approximations of the psd cone," *Mathematical Programming*, vol. 171, no. 1, pp. 1–54, 2018.

[15] A. Yurtsever, J. A. Tropp, O. Fercoq, M. Udell, and V. Cevher, "Scalable semidefinite programming," *SIAM Journal on Mathematics of Data Science*, vol. 3, no. 1, pp. 171–200, 2021.

[16] Y.-C. Chang, N. Roohi, and S. Gao, "Neural lyapunov control," *Advances in neural information processing systems*, vol. 32, 2019.

[17] M. Han, L. Zhang, J. Wang, and W. Pan, "Actor-critic reinforcement learning for control with stability guarantee," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6217–6224, 2020.

[18] Y.-C. Chang and S. Gao, "Stabilizing neural control using self-learned almost lyapunov critics," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 1803–1809.

[19] C. Dawson, Z. Qin, S. Gao, and C. Fan, "Safe nonlinear control using robust neural lyapunov-barrier functions," in *Conference on Robot Learning*, PMLR, 2022, pp. 1724–1735.

[20] H. Dai, B. Landry, L. Yang, M. Pavone, and R. Tedrake, "Lyapunov-stable neural-network control," *arXiv preprint arXiv:2109.14152*, 2021.

[21] R. Zhou, T. Quartz, H. De Sterck, and J. Liu, "Neural lyapunov control of unknown nonlinear systems with stability guarantees," *arXiv preprint arXiv:2206.01913*, 2022.

[22] S. S. Stankovic, M. J. Stanojevic, and D. D. Siljak, "Decentralized overlapping control of a platoon of vehicles," *IEEE Transactions on Control Systems Technology*, vol. 8, no. 5, pp. 816–832, 2000.

[23] R. Tedrake, *Underactuated Robotics, Algorithms for Walking, Running, Swimming, Flying, and Manipulation*. 2022. [Online]. Available: http://underactuated.mit.edu.

[24] W. B. Powell, *Approximate Dynamic Programming: Solving the curses of dimensionality*. John Wiley & Sons, 2007, vol. 703.

[25] I. Khalil, J. Doyle, and K. Glover, *Robust and optimal control*. Prentice hall, 1996.

[26] G. E. Dullerud and F. Paganini, *A course in robust control theory: a convex approach*. Springer Science & Business Media, 2013, vol. 36.

[27] J.-J. E. Slotine and W. Li, *Applied nonlinear control*. Prentice Hall, 1991.

[28] Y.-C. Chang, N. Roohi, and S. Gao, "Neural Lyapunov Control," in *Advances in Neural Information Processing Systems*, vol. 32, 2019, pp. 3245–3254.

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

[30] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh, "A lyapunov-based approach to safe reinforcement learning," *Advances in neural information processing systems*, 2018.

[31] S. M. Richards, F. Berkenkamp, and A. Krause, "The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems," in *Conference on Robot Learning*, PMLR, 2018, pp. 466–476.

[32] G. Manek and J. Z. Kolter, "Learning stable deep dynamics models," *Advances in neural information processing systems*, 2019.

[33] Z. Qin, K. Zhang, Y. Chen, J. Chen, and C. Fan, "Learning safe multi-agent control with decentralized neural barrier certificates," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=P6_q1BRxY8Q.

[34] E. D. Sontag, *Mathematical control theory: deterministic finite dimensional systems*. Springer Science & Business Media, 2013, vol. 6.

[35] T. Liu, D. J. Hill, and Z.-P. Jiang, "Lyapunov formulation of iss cyclic-small-gain in continuous-time dynamical networks," *Automatica*, vol. 47, no. 9, pp. 2088–2093, 2011.

[36] Z.-P. Jiang and T. Liu, "Small-gain theory for stability and control of dynamical networks: A survey," *Annual Reviews in Control*, vol. 46, pp. 58–79, 2018.

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

[38] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural lander: Stable drone landing control using learned dynamics," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 9784–9790.

[39] J. Choi, F. Castañeda, C. J. Tomlin, and K. Sreenath, "Reinforcement learning for safety-critical control under model uncertainty, using control lyapunov functions and control barrier functions," in *Robotics: Science and Systems (RSS)*, 2020.

[40] F. Castañeda, J. J. Choi, B. Zhang, C. J. Tomlin, and K. Sreenath, "Gaussian process-based min-norm stabilizing controller for control-affine systems with uncertain input effects and dynamics," in *2021 American Control Conference (ACC)*, IEEE, 2021, pp. 3683–3690.

[41] A. Abate, D. Ahmed, M. Giacobbe, and A. Peruffo, "Formal synthesis of lyapunov neural networks," *IEEE Control Systems Letters*, vol. 5, no. 3, pp. 773–778, 2020.

[42] A. Abate, D. Ahmed, A. Edwards, M. Giacobbe, and A. Peruffo, "Fossil: A software tool for the formal synthesis of lyapunov functions and barrier certificates using neural networks," in *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, 2021, pp. 1–11.

[43] N. Gaby, F. Zhang, and X. Ye, "Lyapunov-net: A deep neural network architecture for lyapunov function approximation," *arXiv preprint arXiv:2109.13359*, 2021.

[44] A. Mehrjou, M. Ghavamzadeh, and B. Schölkopf, "Neural lyapunov redesign," *Proceedings of Machine Learning Research vol*, vol. 144, pp. 1–24, 2021.

[45] V. Zinage and E. Bakolas, "Neural koopman lyapunov control," *Neurocomputing*, 2023.

[46] L. Wang, S. Zhang, Y. Zhou, C. Fan, P. Zhang, and Y. A. Shamash, "Physics-informed, safety and stability certified neural control for uncertain

networked microgrids," *IEEE Transactions on Smart Grid*, pp. 1–1, 2023. DOI: 10.1109/TSG.2023.3309534.

[47] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, PMLR, 2015, pp. 1889–1897.

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

[49] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, PMLR, 2018, pp. 1861–1870.

[50] U. Rosolia and F. Borrelli, "Learning how to autonomously race a car: A predictive control approach," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2713–2719, 2019.

[51] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," *Advances in neural information processing systems*, vol. 30, 2017.

[52] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3387–3395.

[53] W. Zhao, T. He, and C. Liu, "Model-free safe control for zero-violation reinforcement learning," in *5th Annual Conference on Robot Learning*, 2021.

[54] Z. Qin, Y. Chen, and C. Fan, "Density constrained reinforcement learning," in *International Conference on Machine Learning*, PMLR, 2021, pp. 8682–8692.

[55] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural computation*, vol. 3, no. 1, pp. 88–97, 1991.

[56] M. Bain and C. Sammut, "A framework for behavioural cloning.," in *Machine Intelligence 15*, 1995, pp. 103–129.

[57] S. Schaal, "Is imitation learning the route to humanoid robots?" *Trends in cognitive sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[58] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[59] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 1.

[60] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning," in *Proceedings of the 20th international joint conference on Artifical intelligence*, 2007, pp. 2586–2591.

[61] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proceedings of the 23rd national conference on Artificial intelligence-Volume 3*, 2008, pp. 1433–1438.

[62] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, 2016.

[63] C. Finn, P. Christiano, P. Abbeel, and S. Levine, "A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models," *arXiv preprint arXiv:1611.03852*, 2016.

[64] J. Fu, K. Luo, and S. Levine, "Learning robust rewards with adverserial inverse reinforcement learning," in *International Conference on Learning Representations*, 2018.

[65] P. Henderson, W.-D. Chang, P.-L. Bacon, D. Meger, J. Pineau, and D. Precup, "Optiongan: Learning joint reward-policy options using generative adversarial inverse reinforcement learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[66] D. Brown, W. Goo, P. Nagarajan, and S. Niekum, "Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations," in *International conference on machine learning*, PMLR, 2019, pp. 783–792.

[67] S. Zhang, Z. Cao, D. Sadigh, and Y. Sui, "Confidence-aware imitation learning from demonstrations with varying optimality," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[68] Y.-H. Wu, N. Charoenphakdee, H. Bao, V. Tangkaratt, and M. Sugiyama, "Imitation learning from imperfect demonstration," in *International Conference on Machine Learning*, PMLR, 2019, pp. 6818–6827.

[69] Z. Cao and D. Sadigh, "Learning from imperfect demonstrations from agents with varying dynamics," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5231–5238, 2021.

[70] D. S. Brown, W. Goo, and S. Niekum, "Better-than-demonstrator imitation learning via automatically-ranked demonstrations," in *Conference on robot learning*, PMLR, 2020, pp. 330–359.

[71] L. Chen, R. Paleja, and M. Gombolay, "Learning from suboptimal demonstration via self-supervised reward regression," in *Conference on Robot Learning*, PMLR, 2021, pp. 1262–1277.

[72] V. Tangkaratt, B. Han, M. E. Khan, and M. Sugiyama, "Variational imitation learning with diverse-quality demonstrations," in *International Conference on Machine Learning*, PMLR, 2020, pp. 9407–9417.

[73] V. Tangkaratt, N. Charoenphakdee, and M. Sugiyama, "Robust imitation learning from noisy demonstrations," in *AISTATS*, 2021.

[74] E. Novoseller, Y. Wei, Y. Sui, Y. Yue, and J. Burdick, "Dueling posterior sampling for preference-based reinforcement learning," in *Conference on Uncertainty in Artificial Intelligence*, PMLR, 2020, pp. 1029–1038.

[75] H. Ravanbakhsh and S. Sankaranarayanan, "Learning control lyapunov functions from counterexamples and demonstrations," *Autonomous Robots*, vol. 43, no. 2, pp. 275–307, 2019.

[76] A. Robey, H. Hu, L. Lindemann, H. Zhang, D. V. Dimarogonas, S. Tu, and N. Matni, "Learning control barrier functions from expert demonstrations," in *2020 59th IEEE Conference on Decision and Control (CDC)*, IEEE, 2020, pp. 3717–3724.

[77] G. Chou, N. Ozay, and D. Berenson, "Uncertainty-aware constraint learning for adaptive safe motion planning from demonstrations," in *Conference on Robot Learning*, 2020.

[78] N. Boffi, S. Tu, N. Matni, J.-J. Slotine, and V. Sindhwani, "Learning stability certificates from data," in *Conference on Robot Learning*, PMLR, 2021, pp. 1341–1350.

[79] C. Dawson, Z. Qin, S. Gao, and C. Fan, "Safe nonlinear control using robust neural lyapunov-barrier functions," *Conference on Robot Learning*, 2021.

[80] W. Xiao, R. Hasani, X. Li, and D. Rus, "Barriernet: A safety-guaranteed layer for neural networks," *arXiv preprint arXiv:2111.11277*, 2021.

[81] A. Peruffo, D. Ahmed, and A. Abate, "Automated and formal synthesis of neural barrier certificates for dynamical models," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2021, pp. 370–388.

[82] M. Srinivasan, A. Dabholkar, S. Coogan, and P. A. Vela, "Synthesis of control barrier functions using a supervised machine learning approach," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 7139–7145.

[83] D. Sun, S. Jha, and C. Fan, "Learning certified control using contraction metric," in *Conference on Robot Learning*, 2020.

[84] G. Chou, N. Ozay, and D. Berenson, "Model error propagation via learned contraction metrics for safe feedback motion planning of unknown systems," *arXiv preprint arXiv:2104.08695*, 2021.

[85] Y. Meng, Z. Qin, and C. Fan, "Reactive and safe road user simulations using neural barrier certificate," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021.

[86] Z.-P. Jiang, I. M. Mareels, and Y. Wang, "A lyapunov formulation of the nonlinear small-gain theorem for interconnected iss systems," *Automatica*, vol. 32, no. 8, pp. 1211–1215, 1996.

[87] T. Liu, Z.-P. Jiang, and D. J. Hill, "Lyapunov formulation of the iss cyclic-small-gain theorem for hybrid dynamical networks," *Nonlinear Analysis: Hybrid Systems*, vol. 6, no. 4, pp. 988–1001, 2012.

[88] L. Grüne, J. Pannek, L. Grüne, and J. Pannek, *Nonlinear model predictive control*. Springer, 2017.

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

[90] H. Kwakernaak and R. Sivan, *Linear optimal control systems*. Wiley-interscience New York, 1969, vol. 1072.

[91] S. Bennett, "A brief history of automatic control," *IEEE Control Systems Magazine*, vol. 16, no. 3, pp. 17–25, 1996.

[92] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," *arXiv preprint arXiv:1802.05957*, 2018.

[93] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[94] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html.

[95] S. Wang, S. Toyer, A. Gleave, and S. Emmons, *The* `imitation` *library for imitation learning and inverse reinforcement learning*, https://github.com/HumanCompatibleAI/imitation, 2020.

[96] S. Gao, J. Avigad, and E. M. Clarke, "δ-complete decision procedures for satisfiability over the reals," in *International Joint Conference on Automated Reasoning*, Springer, 2012, pp. 286–300.

[97] R. Bobiti and M. Lazar, "Automated-sampling-based stability verification and doa estimation for nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 63, no. 11, pp. 3659–3674, 2018.

[98] S. Liu, D. Liberzon, and V. Zharnitsky, "Almost lyapunov functions for nonlinear systems," *Automatica*, vol. 113, p. 108 758, 2020.

[99] H. Kwakernaak, R. Sivan, and B. N. D. Tyreus, "Linear optimal control systems," 1974.

[100] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative, multi-agent games," *arXiv preprint arXiv:2103.01955*, 2021.

[101] T. Huang, S. Gao, and L. Xie, "A neural lyapunov approach to transient stability assessment of power electronics-interfaced networked microgrids," *IEEE Transactions on Smart Grid*, vol. 13, no. 1, pp. 106–118, 2021.

[102] Y. Shi, G. Qu, S. Low, A. Anandkumar, and A. Wierman, "Stability constrained reinforcement learning for real-time voltage control," in *2022 American Control Conference (ACC)*, IEEE, 2022, pp. 2715–2721.

[103] S. H. Low, "Convex relaxation of optimal power flow—part i: Formulations and equivalence," *IEEE Transactions on Control of Network Systems*, vol. 1, no. 1, pp. 15–27, 2014.

[104] L. Thurner, A. Scheidler, F. Schäfer, J. Menke, J. Dollichon, F. Meier, S. Meinecke, and M. Braun, "Pandapower — an open-source python tool for convenient modeling, analysis, and optimization of electric power systems," *IEEE Transactions on Power Systems*, vol. 33, no. 6, pp. 6510–6521, Nov. 2018, ISSN: 0885-8950. DOI: 10.1109/TPWRS.2018.2829021.