LEARNING CONTROL LYAPUNOV FUNCTIONS FOR HIGH-DIMENSIONAL UNKNOWN SYSTEMS USING GUIDED ITERATIVE STATE SPACE EXPLORATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Designing stable controllers in complex, high-dimensional systems with unknown dynamics is a critical problem when we deploy robots in the real world. Prior works use learning-based control Lyapunov functions (CLFs) or adaptive control to derive such controllers, but they suffer from two significant challenges: scalability and model transparency. This paper proposes a general framework to jointly learn the local dynamics, a stable controller, and the corresponding CLF in high-dimensional unknown systems. Our approach, GIE-CLF, does not need any knowledge of the environment, such as the dynamics, reward functions, etc, and can scale up to high dimensional systems using only local knowledge of the dynamics inside a trusted tunnel instead of global knowledge required by other methods. We provide theoretical guarantees for our framework and demonstrate it on highly complex systems including a high-fidelity F-16 jet aircraft model that has a 16-dimensional state space and a 4-dimensional input space. Experimental results show that GIE-CLF significantly outperforms prior works in reinforcement learning and imitation learning. We also show that our algorithm can also be extended to learn other control certificate functions for unknown systems.

1 INTRODUCTION

Designing guaranteed stable controllers for dynamical systems that potentially have unknown dynamics is crucial for deploying robots in real-world scenarios, and Lyapunov-based control design is the key to finding such controllers and providing stability guarantees (Parrilo, 2000). Recent years have seen increasing research efforts to develop methods to automatically construct Lyapunov functions and control Lyapunov functions (CLFs) for systems with varying complexities (Giesl & Hafstein, 2015). However, existing methods face two significant challenges: scalability and model transparency. Classical optimization-based control design finds both the controllers and the corresponding Lyapunov functions through solving a sequence of Semi-definite programming (SDP) problems (Ahmadi & Majumdar, 2016; Parrilo, 2000; Majumdar et al., 2013), but it is difficult for them to scale up to high-dimensional systems (Lofberg, 2009) and they are known to have numerical issues (Permenter & Parrilo, 2018), including strict feasibility and numerical reliability issues. Although recent years have seen studies on scalable SDP solvers (Yurtsever et al., 2021), they rely on assumptions like the sparsity of the decision matrix. Moreover, the above methods need the dynamics of the system as ordinary differential equations (ODEs) to be known, which limits their scope of application. Computationally efficient methods such as sample-based Lyapunov function synthesis can scale better to higher dimensional systems; however, they can only find Lyapunov functions for scenarios where a stable controller is already known, providing little insight to engineers attempting to synthesize a stable controller.

In recent years, using Neural Networks (NNs) to represent Lyapunov functions has become increasingly popular (Dawson et al., 2022a; Chang et al., 2019; Dawson et al., 2022b). The effectiveness of deep learning tools for NNs can relieve limitations on the dimensionality of the systems for which one can find a CLF. However, these methods require a precise ODE model of the system to be specified *a priori*, which can be unrealistic for practical systems. For example, the F-16 fighter jet model (Heidlauf et al., 2018) we study in this paper is represented as a mixture of look-up tables, block diagrams, and C programs. It is very difficult to use an ODE to describe the entire system.

For systems without precise mathematical descriptions, researchers have tried to first use system identification, *e.g.* learning the ODE model using NNs, and then find a controller together with Lyapunov functions (Dai et al., 2021). However, fitting a high-dimensional dynamical system as a single NN needs a massive amount of training samples to cover the entire state space, and usually those surrogate NN models can have large prediction errors in space where few samples exist.

For high-dimensional systems, we argue that it is both unnecessary and infeasible to collect data on transitions in all regions of the state space, since only a small subset of that state space is reachable for the agent. As a result, having a model of the system over the entire state space is excessive and unnecessary, if not impossible. Instead, we can learn a model only in this small reachable subspace, then learn a CLF and a corresponding controller that continuously expands this subspace towards the goal. It is non-trivial to construct such a subspace. Therefore, we make a weak assumption that some imperfect and potentially unstable demonstrations are available as an initial guidance on exploring the possible reachable states, and as an initialization of the controller. Such an assumption is realistic as engineers can use existing methods such as PID (Bennett, 1996), model predictive control (Camacho & Alba, 2013), and even hand-crafted controllers such as the one implemented in the F-16 model (Heidlauf et al., 2018) or human demonstrations (Bain & Sammut, 1995). Such controllers often generate unsatisfactory demonstrations because their performance is restricted by the hand-tuned parameters, linearization of the systems, and human limitations. However, our goal is to learn a guaranteed stable controller using the imperfect controller as the initial guidance.

To this end, we propose a novel framework, Learning **C**ontrol **L**yapunov **F**unctions using **G**uided **I**terative State Space **E**xploration (GIE-CLF), to jointly learn the dynamics in the reachable subspace and a stable controller with the corresponding CLF for high-dimensional unknown systems. In each iteration, we learn the dynamics in the reachable subspace using previous experience, jointly update the CLF and the controller, and do constrained exploration to expand the subspace towards the goal. After convergence, our learned controller is guaranteed to be stable with a valid CLF as a certificate. As a result, our dynamics are learned in the reachable subspace and the closed-loop systems are guaranteed to reach the goal without leaving the reachable subspace (see Theorem 2). The main contributions of the paper can be summarized as:

- We propose a novel framework, GIE-CLF, to jointly learn the dynamics in the reachable subspace and a stable controller with the corresponding CLF for high-dimensional unknown systems.
- Theoretical results show that the learned CLF satisfies the CLF conditions in the reachable subspace, and the learned controller is guaranteed to be stable.
- We conduct experiments on benchmarks including inverted pendulum, neural lander (Shi et al., 2019), and the F-16 model (Heidlauf et al., 2018) with 2 tasks. Our results suggest that our learned controller behaves better than other RL and IL algorithms in terms of stabilizing the systems while reducing the number of samples by an order of magnitude.

2 Related Work

Control Lyapunov Functions Our work builds on the widely used Lyapunov theory for stability guarantees. More specifically, if we can find a CLF for a system, it is sufficient to provide formal guarantees of the stability of the closed-loop system with some corresponding control inputs. The majority of classical CLF-based controllers rely on hand-craft CLFs (Choi et al., 2020; Castaneda et al., 2021), or SoS-based SDP (Parrilo, 2000; Majumdar et al., 2013; Ahmadi & Majumdar, 2016), but these approach need known dynamics and are hard to generalize to high-dimensional systems. One promising line of work in learning certificates (Qin et al., 2021b; Saveriano & Lee, 2019; Tsukamoto & Chung, 2020; Sun et al., 2021) uses neural networks as function approximators to learn the Lyapunov Functions (Richards et al., 2018; Abate et al., 2020; 2021; Gaby et al., 2021) or the CLF and the controller simultaneously (Chang et al., 2019; Dawson et al., 2022b; Mehrjou et al., 2021; Zinage & Bakolas, 2022). Most of these works randomly sample states in the whole state space, and do supervised learning to enforce the CLF conditions. These works either assume having knowledge of the dynamics, or fit the dynamics of the entire state space, so they are hard to generalize to high-dimensional real-world scenarios where the number of samples needed is too high to be feasible. Compared with these approaches, our algorithm has the capability to work in environments with totally unknown dynamics, and does not suffer from the curse of dimensionality caused by sampling states in the entire state space.

Reinforcement Learning Reinforcement learning (RL) has shown great power to deal with problems without any knowledge of the dynamics (Schulman et al., 2015; 2017; Haarnoja et al., 2018). However, it is hard for them to have strong guarantees on properties like stability. In addition, the hand-craft reward functions needed by RL algorithms and the sample inefficiency also impede them to generalize to complex environments. The most recent works in the field of learning for control tend to solve this problem by adding certificates in the RL process (Berkenkamp et al., 2017; Chow et al., 2018; Cheng et al., 2019; Han et al., 2020; Chang & Gao, 2021; Zhao et al., 2021; Qin et al., 2021a). However, they all have limitations that make them unsuitable to solve the problem we consider in this paper. For example, Berkenkamp et al. (2017) needs a handcrafted certificate function as a prior, which is not considered in our problem setting. Han et al. (2020) and Chang & Gao (2021) focuses on using CLF to guide the RL exploration instead of making strong stability guarantees, by adding the CLF-related terms to the RL loss. As opposed to these approaches, our algorithm learns the CLF from scratch without any prior knowledge, and we provide a structured way to design loss function instead of using reward functions. In addition, the system controlled by our learned controller is guaranteed to be stable by the learned CLF.

Imitation Learning Our work is also related to imitation learning (IL) since we both use demonstrations. However, classical IL algorithms like behavioral cloning (BC) (Pomerleau, 1991; Bain & Sammut, 1995; Schaal, 1999; Argall et al., 2009; Ross et al., 2011), inverse reinforcement learning (IRL) (Abbeel & Ng, 2004; Ramachandran & Amir, 2007; Ziebart et al., 2008), and adversarial learning (Ho & Ermon, 2016; Finn et al., 2016; Fu et al., 2018; Henderson et al., 2018) focus on recovering the exact policy of the demonstrations thus behaving badly given imperfect demonstrations. A recent line of work on learning from suboptimal demonstrations provide a possible route to learn a policy that surpasses the demonstrations, but they either require different kind of manually-labelled supervisions, *e.g.*, rankings (Brown et al., 2019; Zhang et al., 2021), weights of demonstrations (Wu et al., 2019; Cao & Sadigh, 2021), or have additional assumptions about the environment (Brown et al., 2020; Chen et al., 2021), the demonstrations (Tangkaratt et al., 2020; 2021), or the training process (Novoseller et al., 2020). Moreover, neither of them can provide certificates on the learned policy. Another line of work learns certificates from demonstrations (Robey et al., 2020; Chou et al., 2020; Boffi et al., 2021; Ravanbakhsh & Sankaranarayanan, 2019), but they need additional assumptions like known dynamics, perfect demonstrations, or the ability to query the demonstrator. To move further, our approach uses CLF as natural guidance of the exploration process, and does not require any further supervision or assumptions on either demonstrations or environments.

3 Problem Setting and Preliminaries

We consider deterministic continuous-time, unknown dynamical system $\dot{x} = h(x, u)$, where $x \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$ is the state, $u \in \mathcal{U} \subseteq \mathbb{R}^{n_u}$ is the control input, and $h : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is the *unknown* dynamics. We assume that $h(x, u)$ is Lipschitz continuous like (Berkenkamp et al., 2017). We aim to find a control policy $u = \pi(x)$, $\pi : \mathcal{X} \to \mathcal{U}$, such that from initial states $x(0) \in \mathcal{X}_0$, following the policy π, we are guaranteed to reach the goal x_{goal}, *i.e.*, $\forall x(t)$ satisfying $\dot{x} = h(x, \pi(x))$ and $x(0) \in \mathcal{X}_0$, we have with tolerance δ, $\lim_{t\to\infty} \|x(t) - x_{\text{goal}}\| \leq \delta$.

We assume that we are given a set of demonstration trajectories $\mathcal{D} = \{\xi_1, ..., \xi_{n_D}\}$ generated by a demonstrator d following policy π_d. Unlike the assumption in classical imitation learning works, where the demonstrations are optimal, our demonstrations can be *suboptimal* and *unstable*.

Lyapunov theory is widely used to guarantee the stability of control systems, and CLFs provide further guidance for controller synthesis by defining a set of stabilizing control inputs at a given point in the state space. Following Dawson et al. (2022b), we provide the definition of CLF as:

Definition 1. (Control Lyapunov Functions) *Given a closed-loop system $\dot{x} = h(x, u)$ and a goal point $x_{\text{goal}} \in \mathcal{X}$, suppose there exists a continuously differentiable function $V(x) : \mathcal{X} \to \mathbb{R}$, some constant $\lambda > 0$, such that:*

$$V(x_{\text{goal}}) = 0 \tag{1a}$$

$$V(x) > 0, \quad \forall\, x \in \mathcal{X} \setminus x_{\text{goal}} \tag{1b}$$

$$\inf_u \left\{ \frac{\partial V(x)}{\partial x} h(x, u) + \lambda V(x) \right\} \leq 0, \quad \forall\, x \in \mathcal{X} \setminus x_{\text{goal}} \tag{1c}$$

Then $V(x)$ is a valid CLF for the system.

Moreover, if we can find a valid CLF $V(x)$, we can have the following proposition for the stability of the system:

Proposition 1. (CLF for Stability) *If $V(x)$ is a CLF for system $\dot{x} = h(x,u)$, then starting from $x(0) \in \mathcal{X}$, the system is stable at x_{goal} with any control input $u \in \mathcal{K}(x) = \left\{u | \frac{\partial V(x)}{\partial x} h(x,u) + \lambda V(x) \leq 0\right\}$, i.e., $u \in \mathcal{K}(x) \implies \lim_{t\to\infty} x(t) = x_{\text{goal}}$.*

Remark 1. *In practice, we don't have to show that the CLF conditions (1) are satisfied in the whole state space to prove the stability. Starting from $x(0) \in \mathcal{X}_0$, if we can prove that the CLF conditions are satisfied inside a subset of the state space $\mathcal{G} \subset \mathcal{X}$ and show that following some control policy π, $x(t) \in \mathcal{G}, \forall t$, then it is sufficient to prove the stability.*

4 GIE-CLF

Here we propose our algorithm, Learning **C**ontrol **L**yapunov **F**unctions using **G**uided **I**terative State Space **E**xploration (GIE-CLF), which jointly learns the dynamics in the reachable states and a stable controller with the corresponding CLF. Given imperfect demonstrations $\mathcal{D}$, we first use imitation learning to learn an initial controller $\pi_{\text{init}}(x)$ and fit a local model $\hat{h}_\alpha(x,u)$ which has a small error within the reachable region $\mathcal{H}$ of the demonstrations. Then we jointly learn a controller and the corresponding CLF valid inside the region $\mathcal{H}$. During the training, the divergence of the learned controller and the controller from the last iteration $\pi_{\text{last}}(x)$ and the initial controller $\pi_{\text{init}}(x)$ is limited so that the agent cannot go far from $\mathcal{H}$. Once we have the learned controller, we apply the controller in the environment to drive us closer to the goal point while collecting data that is added to the demonstrations. We repeat this process for several iterations to enlarge the region $\mathcal{H}$ towards the goal. After convergence, the goal will be inside region $\mathcal{H}$, and our controller is guaranteed to reach the goal. We will further introduce the theoretical guarantees of GIE-CLF in Section 5.

Learning from Demonstrations Our framework starts from learning an initial policy and an initial dynamics model from imperfect and sometimes unstable demonstrations $\mathcal{D}$. To learn an initial policy $\pi_{\text{init}}(x)$, we can adopt any off-the-shelf IL algorithms. In our implementation, we use the simplest: BC. The initial policy is unstable because it directly recovers the unstable behavior of the demonstrations. To learn the dynamics model, we parameterize the dynamics as $\hat{h}_\alpha(x,u) : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$, which is a discrete neural network approximation of the real dynamics:

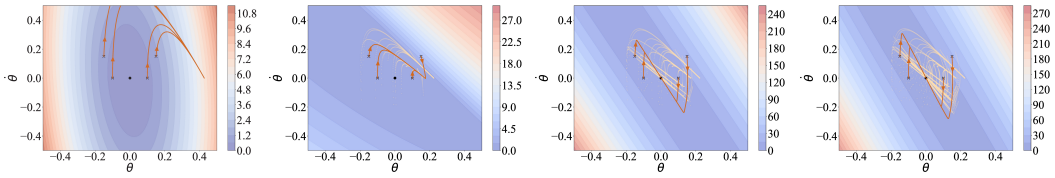
$$x(t+1) - x(t) = \hat{h}_\alpha(x(t), u(t)) \tag{2}$$

where $\hat{h}_\alpha(x) : \mathcal{X} \to \mathbb{R}^{n_x}$ and is a neural network parameterized by α. The dynamics model can be trained with gradient decent algorithms (Kingma & Ba, 2014) using the following loss:

$$\mathcal{L}_{\text{dyn}}(\alpha) = \frac{1}{N} \sum_{x(t),u(t),x(t+1)\in\mathcal{D}} \left\| x(t+1) - x(t) - \hat{h}_\alpha(x(t), u(t)) \right\|^2 + \mu_{\text{dyn}}(\|\alpha\|^2) \tag{3}$$

with weight-decay coefficient $\mu_{\text{dyn}} \geq 0$ to prevent over-fitting. $\{(x(t), u(t), x(t+1)\}$ are transitions sampled from demonstrations $\mathcal{D}$, and N is the number of samples. Since all the training data are drawn from demonstrations, the learned dynamics model is only accurate with in a tunnel $\mathcal{H} \subseteq \mathbb{R}^{n_x \times n_u}$ around the demonstrations. We call $\mathcal{H}$ the trusted tunnel, which is defined as with radius γ, $\mathcal{H} = \{(x,u) | \exists (\bar{x}, \bar{u}) \in \mathcal{D}, \|(x,u) - (\bar{x}, \bar{u})\| \leq \gamma\}$.

Learning the CLF and the Controller Once we have learned the initial policy $\pi_{\text{init}}(x)$ and the local dynamics model $\hat{h}_\alpha(x,u)$, a controller and the corresponding CLF can be trained inside tunnel $\mathcal{H}$. We parameterize the CLF using $V_\theta(x) = x^\top S^\top S x + p_{\text{NN}}(x)^\top p_{\text{NN}}(x)$, where $S \in \mathbb{R}^{n_x \times n_x}$ is a matrix of parameters, $p_{\text{NN}}(x) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$ is a neural network, and θ represents for all the parameters. With this formulation, $V_\theta(x)$ is positive definite by construction. The first term in $V_\theta(x)$ is a quadratic term to capture the linear part of the nonlinear dynamics since quadratics is the form of CLF in linear systems. The second term is used to model the residue of the CLF which is not quadratic. The controller is also parameterized with a neural network $\pi_\phi(x) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$ with parameters ϕ. The desired CLF and the corresponding controller should go towards the goal without going far away from the tunnel $\mathcal{H}$. So ideally, $V_\theta(x)$ and $\pi_\phi(x)$ should be the solution of



(a) Iteration 0 (b) Iteration 2 (c) Iteration 6 (d) Iteration 12

Figure 1: Trajectories generated by GIE-CLF in different iterations in inverted pendulum environment. The counters show the learned CLF. The orange trajectories are generated by the learned controller in current iteration. The light orange dots are the demonstrations generated by the controllers in previous iterations, which also indicate the trusted tunnel $\mathcal{H}$. The black dot is the goal point.

the optimization problem:

$$\min \quad \mathbb{E}\left[\text{dist}((x, \pi_\phi(x)), \mathcal{H})\right] \tag{4a}$$

$$\text{s.t.} \quad V_\theta(x_{\text{goal}}) = 0 \tag{4b}$$

$$\frac{\partial V_\theta(x)}{\partial x} \frac{\hat{h}_\alpha(x, \pi_\phi(x))}{dt} + \lambda V_\theta(x) \leq 0, \quad \forall x \in \mathcal{H} \tag{4c}$$

where the objective is the expected distance of the points on a controlled trajectory to the trusted tunnel $\mathcal{H}$, and dt is the simulation time step. Condition (4c) is a widely used discrete version of condition (1c) (Berkenkamp et al., 2017; Dawson et al., 2022b;a; Chang et al., 2019; Chang & Gao, 2021). In the experiments, we can only simulate the environments in a discrete way, so it is reasonable to use the discrete version. We do not include the condition (1b) because it is already satisfied by the construction of $V_\theta(x)$. We solve problem (4) via self-supervised learning (Dawson et al., 2022a). For conditions (4b) and (4c), we introduce the loss:

$$\mathcal{L}_{\text{CLF}} = V_\theta(x_{\text{goal}})^2 + \frac{\eta_{\text{pos}}}{N} \sum_{x \in \mathcal{D}} \max\left[\epsilon + \frac{\partial V_\theta(x)}{\partial x} \frac{\hat{h}_\alpha(x, \pi_\phi(x))}{dt} + \lambda V_\theta(x), 0\right] \tag{5}$$

where η_{pos} is a positive tuning parameter, dt is the time step, and $\epsilon > 0$ is a parameter that makes the learned CLF more robust w.r.t. the error of the dynamics model. The function of ϵ will be further discussed in Section 5. Both of the terms in loss (5) are zero if the conditions (4b) and (4c) are satisfied inside $\mathcal{H}$. For objective (4a), we introduce the loss $\mathcal{L}_{\text{ctrl}} = \frac{1}{N} \sum_{x \in \mathcal{D}} \|\pi_\phi(x) - \pi_{\text{init}}(x)\|^2$. We train the CLF $V_\theta(x)$ and the controller $\pi_\phi(x)$ by minimizing the loss $\mathcal{L}_{\text{CLF}} + \eta_{\text{ctrl}} \mathcal{L}_{\text{ctrl}}$ with gradient decent methods, where η_{ctrl} is a tuning parameter. If we converge to $\mathcal{L}_{\text{CLF}} = 0$, the CLF will be valid inside $\mathcal{H}$, and the controller will drive us closer to the goal than the initial controller.

Exploration and Iterative Update After the controller is trained for a fixed number of iterations in the last step, we denote it as π_{last} and execute it in the environment to collect more transition data. These data are added to our demonstrations $\mathcal{D}$ to enlarge the trusted tunnel $\mathcal{H}$. One can find the proof of the enlargement of $\mathcal{H}$ in Lemma 1 in the appendix. Then, we repeat the process to learn the dynamics model $\hat{h}_\alpha(x, u)$, the CLF $V_\theta(x)$, and the controller $\pi_\phi(x)$ with the larger set of demonstrations $\mathcal{D}$. To further limit the exploration region of $\pi_\phi(x)$, we add the term of the difference between the current controller $\pi_\phi(x)$ and the converged controller in the last iteration π_{last} in loss $\mathcal{L}_{\text{ctrl}}$, i.e. we modify $\mathcal{L}_{\text{ctrl}}$ to be $\mathcal{L}_{\text{ctrl}} = \frac{1}{N} \sum_{x \in \mathcal{D}} \|\pi_\phi(x) - \pi_{\text{init}}(x)\|^2 + \|\pi_\phi(x) - \pi_{\text{last}}(x)\|^2$. Finally, after several iterations, the goal will be inside the tunnel $\mathcal{H}$, and then our controller $\pi_\phi(x)$ will be guaranteed to reach the goal. We show the process in an inverted pendulum environment in Figure 1. In Figure 1(a), since the demonstrations are imperfect, our initial controller π_{init} cannot reach the goal. Figure 1(b) and Figure 1(c) show that after several iterations, we expand the trusted tunnel $\mathcal{H}$ (the light orange dots) towards the goal and the learned controller gets closer and closer to the goal. After convergence (Figure 1(d)), our controller is guaranteed to reach the goal.

5 Theoretical Guarantees

To show the theoretical guarantees of GIE-CLF, we introduce the following theorems. We provide detailed proofs in Appendix A.

Theorem 1. (Correctness of the Learned CLF) *Suppose the ground truth environment dynamics $h(x, u)$ is Lipschitz continuous with constant L_h. Let dt be the time step and ω be the maximum error of the learned dynamics on the training data, i.e. $\|\hat{h}(\bar{x}, \bar{u}) - h(\bar{x}, \bar{u})dt\| \leq \omega, \forall(\bar{x}, \bar{u}) \in \mathcal{D}$. Let $\mathcal{H}$ be the trusted tunnel around the training data $\mathcal{D}$ with radius γ, i.e. $\mathcal{H} = \{(x, u)|\exists(\bar{x}, \bar{u}) \in \mathcal{D}, \|(x, u) - (\bar{x}, \bar{u})\| \leq \gamma\}$. Assume that the learned CLF $V_\theta(x)$ is Lipschitz continuous with constant L_V, and the learned controller $\pi_\phi(x)$ is Lipschitz continuous with constant L_π. Let $\epsilon \geq L_V(\gamma L_h(1 + L_\pi) + \lambda\gamma + \frac{\omega}{dt}) + \epsilon'$ during the CLF training, where ϵ' is a small positive number, then if the first term of the loss $\mathcal{L}_{\text{CLF}}$ converges to 0 and the second term of the loss $\mathcal{L}_{\text{CLF}}$ is smaller or equal than $\epsilon' > 0$ on the training points, i.e. $\frac{\partial V_\theta(x)}{\partial x}\frac{\hat{h}(x,\pi_\phi(x))}{dt} + \lambda V_\theta(x) + \epsilon \leq \epsilon', \forall x \in \mathcal{D}$, we have*

$$\frac{\partial V_\theta(x)}{\partial x} h(x, \pi_\phi(x)) + \lambda V_\theta(x) \leq 0, \forall x \in \mathcal{H} \tag{6}$$

In practice, ϵ' is often a small number (approximately 10^{-2}), and we will provide the details of the training losses in Appendix B.2. Theorem 1 shows the learned CLF satisfies the CLF conditions inside $\mathcal{H}$, therefore, by Lyapunov theories, the learned controller will drive us closer to the goal. 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, which can be tuned. Larger ϵ makes the learned CLF more robust, but it can also make the training harder, requiring larger neural networks and longer training time. To limit the Lipschitz constant of the neural networks L_V and L_π, we add spectral normalization (Miyato et al., 2018) to each layer in the neural networks.

Theorem 2. (Convergence Guarantee of GIE-CLF) *Let $\pi(x)$ be the converged controller of GIE-CLF, and assume that we sample dense enough in the set of initial states $\mathcal{X}_0$, then $\pi(x)$ can reach the goal, i.e., let $x(t)$ be the trajectory generated by $\pi(x)$ with $x(0) \in \mathcal{X}_0$, we have*

$$\lim_{t\to\infty} \|x(t) - x_{\text{goal}}\| = 0 \tag{7}$$

Theorem 2 provides the guarantee of our controller to be stable. It also shows the capability of our algorithm to find a stable controller.

6 EXPERIMENTS

We conduct experiments in four environments including the inverted pendulum, neural lander (Shi et al., 2019), and the F-16 jet aircraft model (Heidlauf et al., 2018) with two tasks: ground collision avoidance system (GCAS) and tracking. For each environment, we collect imperfect and unstable demonstrations using nominal controllers such as LQR (Kwakernaak, 1969) or PID (Bennett, 1996) with noise, to simulate the imperfect demonstrations given by some bad controllers or non-expert humans. In the first two environments, we collect 20 trajectories as demonstrations, and in the F-16 environments, we collect 40 of them. In our experiments, we aim to answer the following questions:

1. How does GIE-CLF compare with other algorithms in the case of goal reaching?
2. Can the sampling method in GIE-CLF increase the sampling efficiency?
3. Can GIE-CLF generalize to high dimensional systems?

We provide additional implementation details, descriptions of the environments, videos of the behavior of demonstrations and the algorithms, and more results in Appendix B.

Baselines We compare GIE-CLF with the most relevant works in our problem setting including the state-of-the-art reinforcement learning algorithm PPO (Schulman et al., 2017), standard imitation learning algorithm AIRL (Fu et al., 2018), and algorithms of imitation learning from suboptimal demonstrations D-REX (Brown et al., 2020) and SSRR (Chen et al., 2021). For PPO, since it needs reward functions, we hand-craft reward functions in each environment for it. For the imitation learning algorithms AIRL, D-REX and SSRR, we let them learn directly from the demonstrations. Other relevant approaches including 2IWIL (Wu et al., 2019), IC-GAIL (Wu et al., 2019), CAIL (Zhang et al., 2021), and DPS (Novoseller et al., 2020) need different kinds of additional supervision, which is not implementable under the assumptions in our setting, so we do not include them. For a fairer comparison, we initialize all the algorithms with the policy learned from the demonstrations using BC. Compared with these baselines, our algorithm has one additional assumption that we have the

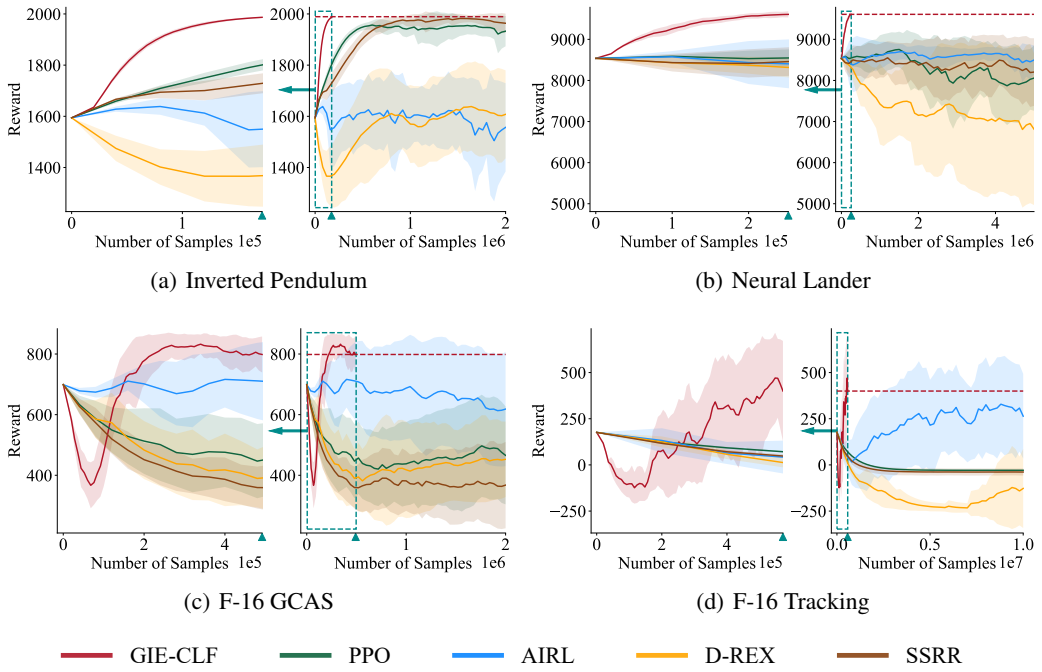


Figure 2: The expected return of GIE-CLF and baselines with respect to the number of samples. The dashed red line shows the converged reward of GIE-CLF. In the right of each subplots, we show the whole curve of the expected return w.r.t. the number of samples, and since our GIE-CLF converges too fast compared with other algorithms, 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 GIE-CLF to converge. GIE-CLF converges after sampling only 160, 240, 960, 1120 trajectories in environments (a)-(d). The reward at 0 number of samples is the reward of demonstrations.

knowledge of the desired goal point. However, we believe the comparison is fair because compared with PPO we do not need the reward function, and compared with AIRL we do not need optimal demonstrations. It is true that we need more information than D-REX and SSRR, but the performance increase of GIE-CLF is large which worth the additional information.

6.1 Environments

Inverted Pendulum Inverted pendulum is a standard environment for testing control algorithms. The demonstrations are collected by a biased LQR controller with noises which makes the inverted pendulum vibrate at a non-goal point to simulate human demonstrations. For the reward function needed by the RL algorithms, we use $r(x) = 2.0 - |\theta|$.

Neural Lander Neural lander (Shi et al., 2019) is a widely used benchmark for systems with unknown disturbance. The state space is 6D including the 3D position and the 3D velocity, with 3D linear acceleration as the control input. The goal is to stabilize the neural lander at a point near the ground. The dynamics include a neural network trained to approximate the aerodynamics ground effect, which is highly nonlinear and unknown. We use a PID controller to collect demonstrations, which oscillates and cannot reach the goal point because of the strong ground effect. For the reward function, we use 10 minus the norm of the velocity and the distance to the goal.

F-16 F-16 (Heidlauf et al., 2018; Djeumou et al., 2021) is a fixed-wing fighter model, with 16D state space and 4D control inputs. The dynamics is complex and cannot be described as ODEs. Instead, the authors of the F-16 model provide a look-up table to describe the aerodynamics. We introduce two tasks in this environment: ground collision avoidance system (GCAS) and waypoint tracking. In GCAS, the F-16 starts at a posture with the head pointing at the ground. The goal is to pull up the fighter, avoid colliding with the ground, and fly smoothly at a height between 800 ft and 1200 ft. We are provided with a nominal controller as the demonstrator which can only pull up the fighter at

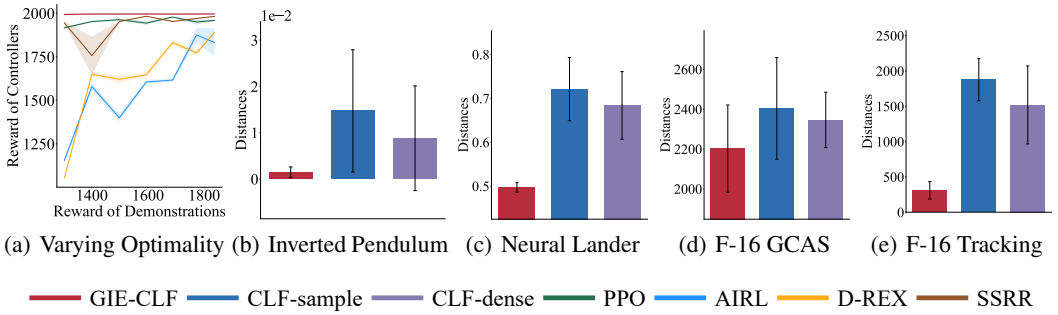


Figure 3: Ablations Studies. (a) The converged reward of learned controllers w.r.t. the reward of the demonstrations. (b-e) Comparison of GIE-CLF with CLF-sample and CLF-dense. In Inverted Pendulum, Neural Lander, and F-16 Tracking, we report the minimum distance to the goal of each algorithm. In F-16 GCA environment, we report the minimum dropping distance of each algorithm.

about 300 ft. For the reward function, we use $r(x) = 2 - 0.001(h - 1200)$ when $h > 1200$, and $r(x) = 1 - 0.001(800 - h)$ when $h < 800$, where h is the height of the fighter. In the tracking environment, the goal for the fighter is to reach a goal point. The demonstrator can drive the F-16 to a point that is near the goal. As the reward function, the F-16 will be punished by the distance to the goal point at each step, and it can receive a bonus after reaching the goal.

6.2 Results

We train each algorithm in each environment 5 times with different random seeds, and at each evaluation point during the training, we test the controllers 5 times. In Figure 2, we show the expected rewards and standard deviations of different algorithms with respect to the number of samples used in the training process. In all the environments, GIE-CLF achieves the highest reward, because it has the ability to explore the environment guided by the learned CLF, and can learn a controller that is guaranteed goal-reaching. PPO does not perform well because it lacks stability guarantees of the learned controller, and its performance depends on the hand-crafted reward function. PPO is a policy gradient method, which does not directly optimize for the Bellman equation, and there is no indicator about how stable the solution of PPO is. However, our algorithm explicitly learns the CLF to provide stability guarantees. In addition, the reward function usually tells the RL agent where the goal is, but our learned CLF can further tell the agent how to get to the goal, which is much more informative. For the imitation learning algorithms, AIRL learns the same policy as the demonstrations and cannot make any improvements. D-REX and SSRR are guided by rankings, which is weaker and less natural guidance compared with our CLF guidance. Therefore, they cannot achieve the same performance as GIE-CLF, and they even behave worse than AIRL in complex systems. From Figure 2, we can also observe that GIE-CLF converges very fast, using about one order of magnitude fewer samples than RL and IL algorithms. This proves our argument that the CLF-guided exploration explores only the necessary regions in the state space and thus improving the sample efficiency. From the inverted pendulum to the F-16 Tracking environment, as the complexity of the environment increases, the gap between GIE-CLF and other algorithms becomes larger and larger. This result demonstrates the capability of GIE-CLF to deal with high dimensional systems.

6.3 Ablation Studies

We do ablation experiments to show the influence of optimality of the demonstrations. We use the inverted pendulum environment, and collect demonstrations with different levels of optimality by adding directed noise in the demonstrator controllers. For each noise level, we train each algorithm 3 times with different random seeds and test each converged controller 20 times. The results are shown in Figure 3(a). We observe that GIE-CLF outperforms other algorithms with demonstrations at different levels of optimality. In addition, we do not observe a significant performance drop of GIE-CLF as the demonstrations become worse. This is because the quality of the demonstrations only influences the convergence speed of GIE-CLF, instead of the learned controller. No matter how bad the demonstrations are, GIE-CLF will finally find a goal-reaching controller. PPO's behavior is

also consistent since the reward function remains unchanged, but it performs worse than GIE-CLF. Other imitation learning algorithms, however, have a performance drop as the demonstrations get worse, because they all by some means depend on the quality of demonstrations.

To show the sampling efficiency and the generalization ability of GIE-CLF, we design another two baselines CLF-sample and CLF-dense. There is a stronger assumption for them, that they can sample transitions from *any* states in the state space, which is unrealistic in the real world. For these two algorithms, we follow the same training process as GIE-CLF, but instead of collecting samples of transitions by applying the current controller in each iteration, we directly sample transitions from the whole state space before training, and let them be the training set of the dynamics. CLF-sample uses the same number of samples as GIE-CLF, while CLF-dense uses the same number of samples as the RL and IL algorithms, which is much more than GIE-CLF. We train the algorithms in our four environments. For Inverted Pendulum, Neural Lander, and F-16 Tracking environments, we report the minimum distance of these algorithms to the goal. For F-16 GCA environment, since every controller will pass the goal region (height between 800ft and 1200ft), we compare the minimum dropping distance instead. As shown in Figure 3 (b)-(e), GIE-CLF outperforms these two baselines. This shows the strong sampling efficiency of GIE-CLF. In addition, although CLF-dense uses much more samples than CLF-sample, its performance still does not increase a lot. This proves that naively increasing the number of samples cannot solve the problem.

7 Extensions

Our framework is general since it can be directly applied to learn other certificates in environments with unknown dynamics. For example, Control Contraction Matrices (CCMs) are differential analogues Lyapunov functions (proving stability in the tangent state space), and can also be learned with a similar framework and theoretical guarantees. We change the learning CLF part to learning CCM algorithms (Sun et al., 2021; Chou et al., 2021), and use the same framework to jointly learn the local dynamics, CCM, and the tracking controller. We test our algorithm in a Dubins car path tracking environment, and as shown in Figure 4, our algorithm outperforms all the RL and IL baselines. We will further discuss the extensions in Appendix D.

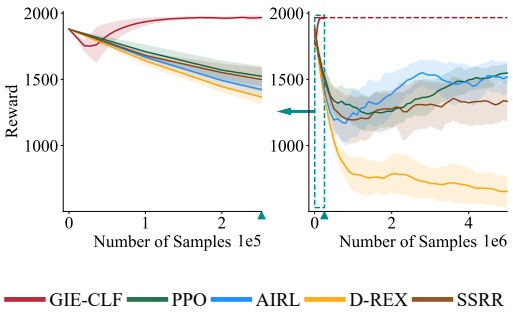


Figure 4: The expected return w.r.t. the number of samples in Dubins car path tracking environment.

8 Conclusion

Summary We propose a general learning framework, GIE-CLF, for learning guaranteed stable controllers in high-dimensional environments with unknown dynamics. GIE-CLF iteratively learns the local dynamics to form the trusted tunnel, and jointly learns a CLF and a controller that expands the trusted tunnel towards the goal. Once the goal is inside the trusted tunnel, GIE-CLF is guaranteed to be goal-reaching. We provide theoretical guarantees on the stability of GIE-CLF and show that the learned controller outperforms baselines in various environments. We also show that the same framework can be applied to learn other certificates in environments with unknown dynamics.

Limitations and Future Work Our framework is limited in a few ways: We need some imperfect demonstrations for initialization in high-dimensional systems. Without the demonstrations, GIE-CLF may take a long time to expand the trusted tunnel to the goal. In addition, we need to have Lipschitz assumptions with the dynamics and to sample dense enough in the initial set of states to derive the theoretical guarantees. If the dynamics are not Lipschitz continuous or the dimensions of the state space is too high that we cannot sample dense enough even in the initial set, it will be hard to learn a valid CLF inside the trusted tunnel. Finally, if we want strong guarantee of the learned controller, we need to do NN verification using falsification tools (Gao et al., 2012). However, we do not include that part because these tools cannot scale to large NN like the ones we consider in this paper. We leave these limitations to future work. We will discuss more in Appendix C.

9 REPRODUCIBILITY STATEMENT

To ensure reproducibility, we provide the information of the environments and the baselines in both Section 6 and Appendix B. In addition, we provide the implementation details including the network structures, hyper-parameters, and the dynamics of environments in Appendix B. Moreover, we provide numerical results in Appendix B. We also provide the video of the learned controllers and the source code in the supplementary materials. For all the theoretical guarantees we introduce in Section 5, we provide proofs in Appendix A.

REFERENCES

Alessandro Abate, Daniele Ahmed, Mirco Giacobbe, and Andrea Peruffo. Formal synthesis of lyapunov neural networks. *IEEE Control Systems Letters*, 5(3):773–778, 2020.

Alessandro Abate, Daniele Ahmed, Alec Edwards, Mirco Giacobbe, and Andrea 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*, pp. 1–11, 2021.

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 1, 2004.

Amir Ali Ahmadi and Anirudha Majumdar. Some applications of polynomial optimization in operations research and real-time decision making. *Optimization Letters*, 10(4):709–729, 2016.

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, pp. 103–129, 1995.

Stuart Bennett. A brief history of automatic control. *IEEE Control Systems Magazine*, 16(3):17–25, 1996.

Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. *Advances in neural information processing systems*, 30, 2017.

Ruxandra Bobiti and Mircea Lazar. Automated-sampling-based stability verification and doa estimation for nonlinear systems. *IEEE Transactions on Automatic Control*, 63(11):3659–3674, 2018.

Nicholas Boffi, Stephen Tu, Nikolai Matni, Jean-Jacques Slotine, and Vikas Sindhwani. Learning stability certificates from data. In *Conference on Robot Learning*, pp. 1341–1350. PMLR, 2021.

Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International conference on machine learning*, pp. 783–792. PMLR, 2019.

Daniel S Brown, Wonjoon Goo, and Scott Niekum. Better-than-demonstrator imitation learning via automatically-ranked demonstrations. In *Conference on robot learning*, pp. 330–359. PMLR, 2020.

Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer science & business media, 2013.

Zhangjie Cao and Dorsa Sadigh. Learning from imperfect demonstrations from agents with varying dynamics. *IEEE Robotics and Automation Letters*, 6(3):5231–5238, 2021.

Fernando Castaneda, Jason J Choi, Bike Zhang, Claire J Tomlin, and Koushil Sreenath. Gaussian process-based min-norm stabilizing controller for control-affine systems with uncertain input effects and dynamics. In *2021 American Control Conference (ACC)*, pp. 3683–3690. IEEE, 2021.

Ya-Chien Chang and Sicun Gao. Stabilizing neural control using self-learned almost lyapunov critics. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1803–1809. IEEE, 2021.

Ya-Chien Chang, Nima Roohi, and Sicun Gao. Neural lyapunov control. *Advances in neural information processing systems*, 32, 2019.

Letian Chen, Rohan Paleja, and Matthew Gombolay. Learning from suboptimal demonstration via self-supervised reward regression. In *Conference on Robot Learning*, pp. 1262–1277. PMLR, 2021.

Richard Cheng, Gábor Orosz, Richard M Murray, and Joel 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*, volume 33, pp. 3387–3395, 2019.

Jason Choi, Fernando Castañeda, Claire J Tomlin, and Koushil 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.

Glen Chou, Necmiye Ozay, and Dmitry Berenson. Uncertainty-aware constraint learning for adaptive safe motion planning from demonstrations. In *Conference on Robot Learning*, 2020.

Glen Chou, Necmiye Ozay, and Dmitry Berenson. Model error propagation via learned contraction metrics for safe feedback motion planning of unknown systems. *arXiv preprint arXiv:2104.08695*, 2021.

Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

Hongkai Dai, Benoit Landry, Lujie Yang, Marco Pavone, and Russ Tedrake. Lyapunov-stable neural-network control. *arXiv preprint arXiv:2109.14152*, 2021.

Charles Dawson, Sicun Gao, and Chuchu Fan. Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods. *arXiv preprint arXiv:2202.11762*, 2022a.

Charles Dawson, Zengyi Qin, Sicun Gao, and Chuchu Fan. Safe nonlinear control using robust neural lyapunov-barrier functions. In *Conference on Robot Learning*, pp. 1724–1735. PMLR, 2022b.

Franck Djeumou, Aditya Zutshi, and Ufuk Topcu. On-the-fly, data-driven reachability analysis and control of unknown systems: an f-16 aircraft case study. In *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, pp. 1–2, 2021.

Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *arXiv preprint arXiv:1611.03852*, 2016.

Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *International Conference on Learning Representations*, 2018.

Nathan Gaby, Fumin Zhang, and Xiaojing Ye. Lyapunov-net: A deep neural network architecture for lyapunov function approximation. *arXiv preprint arXiv:2109.13359*, 2021.

Sicun Gao, Jeremy Avigad, and Edmund M Clarke. δ-complete decision procedures for satisfiability over the reals. In *International Joint Conference on Automated Reasoning*, pp. 286–300. Springer, 2012.

Peter Giesl and Sigurdur Hafstein. Review on computational methods for lyapunov functions. *Discrete & Continuous Dynamical Systems-B*, 20(8):2291, 2015.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Minghao Han, Lixian Zhang, Jun Wang, and Wei Pan. Actor-critic reinforcement learning for control with stability guarantee. *IEEE Robotics and Automation Letters*, 5(4):6217–6224, 2020.

Peter Heidlauf, Alexander Collins, Michael Bolender, and Stanley Bak. Verification challenges in f-16 ground collision avoidance and other automated maneuvers. In *ARCH@ ADHS*, pp. 208–217, 2018.

Peter Henderson, Wei-Di Chang, Pierre-Luc Bacon, David Meger, Joelle Pineau, and Doina Precup. Optiongan: Learning joint reward-policy options using generative adversarial inverse reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Huibert Kwakernaak. *Linear optimal control systems*, volume 1072. 1969.

Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, Mykel J Kochenderfer, et al. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization*, 4(3-4):244–404, 2021.

Shenyu Liu, Daniel Liberzon, and Vadim Zharnitsky. Almost lyapunov functions for nonlinear systems. *Automatica*, 113:108758, 2020.

Johan Lofberg. Pre-and post-processing sum-of-squares programs in practice. *IEEE transactions on automatic control*, 54(5):1007–1011, 2009.

Anirudha Majumdar, Amir Ali Ahmadi, and Russ Tedrake. Control design along trajectories with sums of squares programming. In *2013 IEEE International Conference on Robotics and Automation*, pp. 4054–4061. IEEE, 2013.

Ian R Manchester and Jean-Jacques E Slotine. Control contraction metrics: Convex and intrinsic criteria for nonlinear feedback design. *IEEE Transactions on Automatic Control*, 62(6):3046–3053, 2017.

Arash Mehrjou, Mohammad Ghavamzadeh, and Bernhard Schölkopf. Neural lyapunov redesign. *Proceedings of Machine Learning Research vol*, 144:1–24, 2021.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Ellen Novoseller, Yibing Wei, Yanan Sui, Yisong Yue, and Joel Burdick. Dueling posterior sampling for preference-based reinforcement learning. In *Conference on Uncertainty in Artificial Intelligence*, pp. 1029–1038. PMLR, 2020.

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

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Frank Permenter and Pablo Parrilo. Partial facial reduction: simplified, equivalent sdps via approximations of the psd cone. *Mathematical Programming*, 171(1):1–54, 2018.

Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97, 1991.

Zengyi Qin, Yuxiao Chen, and Chuchu Fan. Density constrained reinforcement learning. In *International Conference on Machine Learning*, pp. 8682–8692. PMLR, 2021a.

Zengyi Qin, Kaiqing Zhang, Yuxiao Chen, Jingkai Chen, and Chuchu Fan. Learning safe multi-agent control with decentralized neural barrier certificates. In *International Conference on Learning Representations*, 2021b.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pp. 2586–2591, 2007.

Hadi Ravanbakhsh and Sriram Sankaranarayanan. Learning control lyapunov functions from counterexamples and demonstrations. *Autonomous Robots*, 43(2):275–307, 2019.

Spencer M Richards, Felix Berkenkamp, and Andreas Krause. The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems. In *Conference on Robot Learning*, pp. 466–476. PMLR, 2018.

Alexander Robey, Haimin Hu, Lars Lindemann, Hanwen Zhang, Dimos V Dimarogonas, Stephen Tu, and Nikolai Matni. Learning control barrier functions from expert demonstrations. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pp. 3717–3724. IEEE, 2020.

Stéphane Ross, Geoffrey Gordon, and Drew 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*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Matteo Saveriano and Dongheui Lee. Learning barrier functions for constrained motion planning with dynamical systems. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 112–119. IEEE, 2019.

Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3 (6):233–242, 1999.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Guanya Shi, Xichen Shi, Michael O'Connell, Rose Yu, Kamyar Azizzadenesheli, Animashree Anandkumar, Yisong Yue, and Soon-Jo Chung. Neural lander: Stable drone landing control using learned dynamics. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 9784–9790. IEEE, 2019.

Dawei Sun, Susmit Jha, and Chuchu Fan. Learning certified control using contraction metric. In *Conference on Robot Learning*, pp. 1519–1539. PMLR, 2021.

Voot Tangkaratt, Bo Han, Mohammad Emtiyaz Khan, and Masashi Sugiyama. Variational imitation learning with diverse-quality demonstrations. In *International Conference on Machine Learning*, pp. 9407–9417. PMLR, 2020.

Voot Tangkaratt, Nontawat Charoenphakdee, and Masashi Sugiyama. Robust imitation learning from noisy demonstrations. In *AISTATS*, 2021.

Hiroyasu Tsukamoto and Soon-Jo Chung. Neural contraction metrics for robust estimation and control: A convex optimization approach. *IEEE Control Systems Letters*, 5(1):211–216, 2020.

Steven Wang, Sam Toyer, Adam Gleave, and Scott Emmons. The `imitation` library for imitation learning and inverse reinforcement learning. `https://github.com/HumanCompatibleAI/imitation`, 2020.

Yueh-Hua Wu, Nontawat Charoenphakdee, Han Bao, Voot Tangkaratt, and Masashi Sugiyama. Imitation learning from imperfect demonstration. In *International Conference on Machine Learning*, pp. 6818–6827. PMLR, 2019.

Alp Yurtsever, Joel A Tropp, Olivier Fercoq, Madeleine Udell, and Volkan Cevher. Scalable semidefinite programming. *SIAM Journal on Mathematics of Data Science*, 3(1):171–200, 2021.

Songyuan Zhang, Zhangjie Cao, Dorsa Sadigh, and Yanan Sui. Confidence-aware imitation learning from demonstrations with varying optimality. *Advances in Neural Information Processing Systems*, 34, 2021.

Weiye Zhao, Tairan He, and Changliu Liu. Model-free safe control for zero-violation reinforcement learning. In *5th Annual Conference on Robot Learning*, 2021.

Brian D Ziebart, Andrew Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd national conference on Artificial intelligence-Volume 3*, pp. 1433–1438, 2008.

Vrushabh Zinage and Efstathios Bakolas. Neural koopman lyapunov control. *arXiv preprint arXiv:2201.05098*, 2022.

A PROOFS

In this section, we provide proofs of the theorems introduced in Section 5.

Theorem 1. (Correctness of the Learned CLF) *Suppose the ground truth environment dynamics $h(x,u)$ is Lipschitz continuous with constant L_h. Let dt be the time step and ω be the maximum error of the learned dynamics on the training data, i.e. $\|\hat{h}(\bar{x},\bar{u}) - h(\bar{x},\bar{u})dt\| \leq \omega, \forall(\bar{x},\bar{u}) \in \mathcal{D}$. Let $\mathcal{H}$ be the trusted tunnel around the training data $\mathcal{D}$ with radius γ, i.e. $\mathcal{H} = \{(x,u)|\exists(\bar{x},\bar{u}) \in \mathcal{D}, \|(x,u) - (\bar{x},\bar{u})\| \leq \gamma\}$. Assume that the learned CLF $V_\theta(x)$ is Lipschitz continuous with constant L_V, and the learned controller $\pi_\phi(x)$ is Lipschitz continuous with constant L_π. Let $\epsilon \geq L_V(\gamma L_h(1+L_\pi) + \lambda\gamma + \frac{\omega}{dt}) + \epsilon'$ during the CLF training, where ϵ' is a small positive number, then if the first term of the loss $\mathcal{L}_{\text{CLF}}$ converges to 0 and the second term of the loss $\mathcal{L}_{\text{CLF}}$ is smaller or equal than $\epsilon' > 0$ on the training points, i.e. $\frac{\partial V_\theta(x)}{\partial x}\frac{\hat{h}(x,\pi_\phi(x))}{dt} + \lambda V_\theta(x) + \epsilon \leq \epsilon', \forall x \in \mathcal{D}$, we have*

$$\frac{\partial V_\theta(x)}{\partial x} h(x,\pi_\phi(x)) + \lambda V_\theta(x) \leq 0, \forall x \in \mathcal{H} \tag{8}$$

Proof. For $\forall \bar{x} \in \mathcal{D}$, we have

$$\begin{aligned}
&\frac{\partial V_\theta(\bar{x})}{\partial x}\frac{\hat{h}(\bar{x},\pi_\phi(\bar{x}))}{dt} + \lambda V_\theta(\bar{x}) \\
&= \frac{\partial V_\theta(\bar{x})}{\partial x}\frac{\hat{h}(\bar{x},\pi_\phi(\bar{x})) - h(\bar{x},\pi_\phi(\bar{x}))dt + h(\bar{x},\pi_\phi(\bar{x}))dt}{dt} + \lambda V_\theta(\bar{x}) \\
&= \frac{\partial V_\theta(\bar{x})}{\partial x}\frac{\hat{h}(\bar{x},\pi_\phi(\bar{x})) - h(\bar{x},\pi_\phi(\bar{x}))dt}{dt} + \frac{\partial V_\theta(\bar{x})}{\partial x} h(\bar{x},\pi_\phi(\bar{x})) + \lambda V_\theta(\bar{x}) \\
&\geq -L_V\frac{\omega}{dt} + \frac{\partial V_\theta(\bar{x})}{\partial x} h(\bar{x},\pi_\phi(\bar{x})) + \lambda V_\theta(\bar{x})
\end{aligned} \tag{9}$$

The inequality is the Cauchy-Schwartz Inequality with the fact that $V_\theta(x)$ is Lipschitz continuous so it also has bounded gradient L_V. After the convergence of the training, we have $\frac{\partial V_\theta(\bar{x})}{\partial x}\frac{\hat{h}(\bar{x},\pi_\phi(\bar{x}))}{dt} + \lambda V_\theta(\bar{x}) \leq -\epsilon + \epsilon', \forall \bar{x} \in \mathcal{D}$. Substitute this in Equation (9), we have

$$-L_V\frac{\omega}{dt} + \frac{\partial V_\theta(\bar{x})}{\partial x} h(\bar{x},\pi_\phi(\bar{x})) + \lambda V_\theta(\bar{x}) \leq -\epsilon + \epsilon' \tag{10}$$

Therefore,

$$\frac{\partial V_\theta(\bar{x})}{\partial x} h(\bar{x},\pi_\phi(\bar{x})) + \lambda V_\theta(\bar{x}) \leq L_V\frac{\omega}{dt} - \epsilon + \epsilon' \leq -L_V\gamma(L_h(1+L_\pi) + \lambda), \quad \forall \bar{x} \in \mathcal{D} \tag{11}$$

Then, for $\forall x \in \mathcal{H}$, we have

$$\begin{aligned}
&\frac{\partial V_\theta(x)}{\partial x} h(x,\pi_\phi(x)) + \lambda V_\theta(x) \\
&= \frac{\partial V_\theta(x)}{\partial x}(h(x,\pi_\phi(x)) - h(\bar{x},\pi_\phi(\bar{x})) + h(\bar{x},\pi_\phi(\bar{x}))) + \lambda V_\theta(x) \\
&= \frac{\partial V_\theta(x)}{\partial x}(h(x,\pi_\phi(x)) - h(\bar{x},\pi_\phi(\bar{x}))) + \frac{\partial V_\theta(x)}{\partial x} h(\bar{x},\pi_\phi(\bar{x})) + \lambda(V_\theta(x) - V_\theta(\bar{x})) + \lambda V_\theta(\bar{x}) \\
&\leq L_V(h(x,\pi_\phi(x)) - h(\bar{x},\pi_\phi(\bar{x}))) + \lambda L_V\gamma + \frac{\partial V_\theta(x)}{\partial x} h(\bar{x},\pi_\phi(\bar{x})) + \lambda V_\theta(\bar{x})
\end{aligned} \tag{12}$$

where $\bar{x} \in \mathcal{D}$ is the nearest training data of x. The inequality is from the Lipschitz continuity of $V_\theta(x)$ and the definition of the trusted tunnel $\mathcal{H}$. From the Lipschitz continuity of the dynamics $h(x,u)$ and the learned controller $\pi_\phi(x)$, we have

$$\begin{aligned}
h(x,\pi_\phi(x)) - h(\bar{x},\pi_\phi(\bar{x})) &\leq L_h\|(x,\pi_\phi(x)) - (\bar{x},\pi_\phi(\bar{x}))\| = L_h\|(x-\bar{x},\pi_\phi(x) - \pi_\phi(\bar{x}))\| \\
&\leq L_h\|x-\bar{x}\| + \|\pi_\phi(x) - \pi_\phi(\bar{x})\| \leq (1+L_\pi)\gamma L_h
\end{aligned} \tag{13}$$

where the second inequality is the triangle inequality, and the first and the third inequality uses the Lipschitz continuity of $h(x,u)$ and $\pi_\phi(x)$, and the definition of the trusted tunnel $\mathcal{H}$. Substitute Equation (11) and Equation (13) to Equation (12), we have

$$\begin{aligned}&\frac{\partial V_\theta(x)}{\partial x}h(x,\pi_\phi(x))+\lambda V_\theta(x)\\&\leq L_V L_h\gamma(1+L_\pi)+\lambda L_V\gamma-L_V\gamma(L_h(1+L_\pi)+\lambda)\\&=0,\qquad\forall x\in\mathcal{H}\end{aligned}\tag{14}$$

□

Theorem 2. (Convergence Guarantee of GIE-CLF) *Let $\pi(x)$ be the converged controller of GIE-CLF, and assume that we sample dense enough in the set of initial states $\mathcal{X}_0$, then $\pi(x)$ can reach the goal, i.e., let $x(t)$ be the trajectory generated by $\pi(x)$ with $x(0)\in\mathcal{X}_0$, we have*

$$\lim_{t\to\infty}\|x(t)-x_{\text{goal}}\|=0\tag{15}$$

Proof. The proof follows the following several lemmas.

Lemma 1. *Let $\mathcal{H}_\tau$ be the trusted tunnel of the τ-th iteration. Then we have $\mathcal{H}_\tau\subsetneqq\mathcal{H}_{\tau+1}$ when $x_{\text{goal}}\notin\mathcal{H}_\tau$.*

Proof. The assumption that we sample dense enough in the set of initial states $\mathcal{X}_0$ makes sure that our Monte Carlo estimation of the trusted tunnel $\mathcal{H}$ is correct. (This may not be strict, and we will further discuss about it in Appendix C.2). By the construction of the trusted tunnel $\mathcal{H}$, we have $\mathcal{H}_\tau\subseteq\mathcal{H}_{\tau+1}$, because we collect more data by executing controller π_τ in the environment, which will enlarge the set of demonstrations $\mathcal{D}$. Since $\mathcal{H}$ is defined as the tunnel around $\mathcal{D}$ with error γ, we can conclude that $\mathcal{H}$ becomes larger or stays the same size after each iteration, *i.e.*, $\mathcal{H}_\tau\subseteq\mathcal{H}_{\tau+1}$. Using Theorem 1, we know that the CLF conditions are satisfied in $\mathcal{H}_\tau$ with controller π_τ, and in $\mathcal{H}_{\tau+1}$ with controller $\pi_{\tau+1}$. By Lyapunov theories, when $x_{\text{goal}}\notin\mathcal{H}_\tau$, we have $V_\theta(x)>0,\forall x\in\mathcal{H}_\tau$, so $\dot{V}_\theta(x)<0,\forall x\in\mathcal{H}_\tau$. Therefore, the controller will converge exponentially towards the point $V(x)=0$, which is x_{goal}. Since $x_{\text{goal}}\notin\mathcal{H}_\tau$, we can conclude that the controller will finally leave $\mathcal{H}_\tau$. Once the controller leaves $\mathcal{H}_\tau$, it will collect demonstrations that is not in $\mathcal{H}_\tau$, which will enlarge the trusted tunnel $\mathcal{H}$, so $\mathcal{H}_\tau\subsetneqq\mathcal{H}_{\tau+1}$. □

Lemma 2. *Let $V_\tau(x)$ be the learned Lyapunov function at the τ-th iteration. Then*

$$\min_{x\in\mathcal{H}_{\tau+1}}V_\tau(x)\leq\min_{x\in\mathcal{H}_\tau}V_\tau(x)\tag{16}$$

and the equality holds if and only if $x_{\text{goal}}\in\mathcal{H}_\tau$.

Proof. Using Lemma 1, we know that $\mathcal{H}_\tau\subseteq\mathcal{H}_{\tau+1}$, so the inequality holds. When $x_{\text{goal}}\in\mathcal{H}_\tau$, we have $\min_{x\in\mathcal{H}_{\tau+1}}V_\tau(x)=\min_{x\in\mathcal{H}_\tau}V_\tau(x)=V(x_{\text{goal}})=0$, so the sufficient condition holds. For the necessary condition, we prove its inverse negative proposition, that $\min_{x\in\mathcal{H}_{\tau+1}}V_\tau(x)<\min_{x\in\mathcal{H}_\tau}V_\tau(x)$ if $x_{\text{goal}}\notin\mathcal{H}$. This is obvious since $\mathcal{H}_{\tau+1}$ is generated from $\mathcal{H}_\tau$ by exploring towards the direction that $V_\tau(x)$ decreases. □

Lemma 3. *If $x_{\text{goal}}\in\mathcal{H}_\tau$, our learned controller π_τ can reach the goal.*

Proof. Note that by construction, $\mathcal{H}$ is the reachable set of the controller π_τ starting from $x(0)\in\mathcal{X}_0$. Also, by Theorem 1, the CLF conditions are satisfied inside $\mathcal{H}_\tau$ with $V(x_{\text{goal}})=0$. Therefore, by Lyapunov theories, π_τ can reach the goal. □

Combining the above lemmas, we can conclude that $\mathcal{H}$ will grow in each iteration towards the goal, and once it contains x_{goal}, our learned controller is goal-reaching. Therefore, our converged controller can reach the goal. □

B EXPERIMENTS

In this section, we provide additional experimental details and results. We provide the code of our experiments in the file 'gie-clf.zip' in the supplementary materials.

B.1 EXPERIMENTAL DETAILS

Here we introduce the details of the experiments, including the implementation details of GIE-CLF and the baselines, choice of hyper-parameters, and detailed introductions of environments. 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).

B.1.1 IMPLEMENTATION DETAILS

Implementation of GIE-CLF Our framework contains three models: the dynamics model $\hat{h}_\alpha(x,u)$, the CLF $V_\theta(x) = x^\top S^\top S x + p_{\text{NN}}(x)^\top p_{\text{NN}}(x)$, and the controller $\pi_\phi(x)$. $\hat{h}_\alpha(x,u)$, $p_{\text{NN}}(x)$, and $\pi_\phi(x)$ 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 (Miyato et al., 2018) to each layer in the neural networks. We implement our algorithm in the PyTorch framework (Paszke et al., 2019) based on the rCLBF repository[1] (Dawson et al., 2022b). During training, we use ADAM (Kingma & Ba, 2014) as the optimizer to optimize the parameters of the neural networks. The loss function used in training the controller and the CLF is

$$\mathcal{L} = \eta_{\text{goal}}\mathcal{L}_{\text{goal}} + \eta_{\text{pos}}\mathcal{L}_{\text{pos}} + \eta_{\text{ctrl}}\mathcal{L}_{\text{ctrl}} \tag{17}$$

where η_{goal}, η_{pos}, η_{ctrl} are tuning parameters, which we will further introduce in Appendix B.1.2, and

$$\begin{aligned}
\mathcal{L}_{\text{goal}} &= V_\theta(x_{\text{goal}})^2 \\
\mathcal{L}_{\text{pos}} &= \frac{1}{N}\sum_{x\in\mathcal{D}} \max\left[\epsilon + \frac{\partial V_\theta(x)}{\partial x}\frac{\hat{h}_{\alpha,\beta}(x,\pi_\phi(x))}{dt} + \lambda V_\theta(x), 0\right] \\
\mathcal{L}_{\text{ctrl}} &= \frac{1}{N}\sum_{x\in\mathcal{D}} \|\pi_\phi(x) - \pi_{\text{init}}(x)\|^2 + \|\pi_\phi(x) - \pi_{\text{last}}(x)\|^2
\end{aligned} \tag{18}$$

Implementation of the baselines We implement PPO based on the open-source python package stablebaselines3[2] (Raffin et al., 2021), AIRL based on the open-source python package Imitation[3] (Wang et al., 2020), and D-REX and SSRR based on their official implementations [4][5], with some adjustments based on the CAIL repository [6] (Zhang et al., 2021). 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 (Kingma & Ba, 2014) as the optimizer with learning rate 3×10^{-4} to optimize the parameters of the neural networks.

B.1.2 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 and the weight decay coefficient μ_{dyn}. 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 $\lambda = 1.0$. 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, but it also makes the training harder. We choose $\epsilon = 1.0$ for the inverted pendulum and the neural lander environments, 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

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

[2] https://github.com/DLR-RM/stable-baselines3 (MIT license)

[3] https://github.com/HumanCompatibleAI/imitation (MIT license)

[4] https://github.com/dsbrown1331/CoRL2019-DREX (MIT license)

[5] https://github.com/CORE-Robotics-Lab/SSRR

[6] https://github.com/Stanford-ILIAD/Confidence-Aware-Imitation-Learning (MIT license)

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. We will further introduce the choice of the weights in Appendix B.1.3. 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}. For the weight decay coefficient μ_{dyn}, we let it be 10^{-3} to make the learned dynamics model smooth, without deviating too much from the origin mean square error loss.

B.1.3 ENVIRONMENTS

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

$$f(x) = \begin{bmatrix} \dot{\theta} \\ \frac{g\theta}{L} - \frac{b\dot{\theta}}{mL^2} \end{bmatrix} \qquad g(x) = \begin{bmatrix} 1 \\ \overline{mL^2} \end{bmatrix} \tag{19}$$

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 $x_{\text{goal}} = [0, 0]^\top$. The simulating time step is 0.01.

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 (19), we use $\eta_{\text{goal}} = 10.0$, $\eta_{\text{pos}} = 10.0$, $\eta_{\text{ctrl}} = 1.0$.

Neural Lander Neural lander (Shi et al., 2019) 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

$$f(x) = \left[v_x, v_y, v_z, \frac{F_{a1}}{m}, \frac{F_{a2}}{m}, \frac{F_{a3}}{m} - g'\right]^\top$$
$$g(x) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1/m & 0 & 0 \\ 0 & 1/m & 0 \\ 0 & 0 & 1/m \end{bmatrix} \tag{20}$$

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. We define the goal point at $x_{\text{goal}} = [0, 0, 0.5, 0, 0, 0]^\top$. The simulating time step is 0.01.

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

$$u = \begin{bmatrix} -8p_x - v_x \\ -8p_y - v_y \\ -8p_z - v_z + mg' \end{bmatrix} \tag{21}$$

We collect 20 trajectories for the demonstrations, where each trajectory has 1000 time steps. For hyper-parameters in the loss function (19), we use $\eta_{\text{goal}} = 100.0$, $\eta_{\text{pos}} = 50.0$, $\eta_{\text{ctrl}} = 1.0$.

F-16 Ground Collision Avoidance System (GCAS) F-16 (Heidlauf et al., 2018)[7] is a fixed-wing fighter model. Its state space is 16D including air speed v, angle of attack α, angle of sideslip β,

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

Method	Inverted Pendulum	Neural Lander	F-16 GCAS	F-16 Tracking
GIE-CLF	$\mathbf{1986.84} \pm 2.31$	$\mathbf{9607.81} \pm 75.66$	$\mathbf{798.48} \pm 59.41$	$\mathbf{399.80} \pm 271.54$
PPO	1932.63 ± 58.98	8057.52 ± 794.97	465.56 ± 161.59	-29.00 ± 0.22
AIRL	1556.76 ± 172.55	8496.47 ± 354.62	618.39 ± 188.15	263.74 ± 229.44
D-REX	1607.71 ± 171.85	6814.79 ± 1846.51	452.72 ± 124.36	-127.24 ± 214.49
SSRR	1963.60 ± 37.85	8204.53 ± 823.13	367.60 ± 142.50	-38.57 ± 19.18
Demo	1594.23 ± 57.72	8536.57 ± 95.93	698.61 ± 40.31	176.35 ± 477.94

Table 1: Converged rewards and standard deviations of GIE-CLF and the baselines in the four environments

Method	Inverted Pendulum	Neural Lander	F-16 GCAS	F-16 Tracking
GIE-CLF	**172**	**252**	**492**	**572**
PPO	2000	5000	2000	10000
AIRL	2000	5000	2000	10000
D-REX	2000	5000	2000	10000
SSRR	2000	5000	2000	10000

Table 2: Number of samples (k) used for GIE-CLF and the baselines in the four environments

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 a look-up table to describe the aerodynamics. The look-up table is 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. We define the goal point at $h = 1000$. 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 (19), 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 GCAS 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. We collect 40 trajectories for the demonstrations, where each trajectory has 500 time steps. For hyper-parameters in the loss function (19), we use $\eta_{\text{goal}} = 100.0$, $\eta_{\text{pos}} = 50.0$, $\eta_{\text{ctrl}} = 1000.0$.

B.2 More Results

Numerical Comparison We provide the numerical comparison of the converged rewards and the number of samples of GIE-CLF and the baselines in Table 1 and Table 2, corresponding to Figure 2 in the main text. We can observe that GIE-CLF outperforms all the baseline methods in all environments, and the number of samples GIE-CLF used is about one order of magnitude less than the baselines. We also provide the numerical results of the ablation studies in Table 3 corresponding to Figure 3(a) in the main text. It is clear that the performance of GIE-CLF is consistently the best with demonstrations with different optimality, without significant performance drop as the demonstrations become worse.

Demonstrations	1837	1773	1688	1594
GIE-CLF	$\mathbf{1995.63} \pm 2.69$	$\mathbf{1995.38} \pm 3.71$	$\mathbf{1995.00} \pm 3.16$	$\mathbf{1995.52} \pm 3.05$
PPO	1958.74 ± 2.15	1949.79 ± 8.93	1978.03 ± 1.52	1943.27 ± 5.75
AIRL	1831.78 ± 47.37	1875.50 ± 23.55	1616.20 ± 4.62	1604.92 ± 5.97
D-REX	1888.75 ± 3.61	1772.05 ± 8.98	1831.68 ± 6.50	1646.63 ± 6.56
SSRR	1981.96 ± 0.65	1970.13 ± 0.84	1952.56 ± 1.34	1982.67 ± 0.61
Demonstrations	1498	1400	1302	
GIE-CLF	$\mathbf{1995.28} \pm 2.81$	$\mathbf{1995.41} \pm 3.12$	$\mathbf{1992.91} \pm 3.22$	
PPO	1962.92 ± 2.80	1952.21 ± 2.34	1916.23 ± 2.95	
AIRL	1399.69 ± 11.65	1578.24 ± 11.39	1154.61 ± 7.45	
D-REX	1620.61 ± 12.01	1649.49 ± 6.22	1053.79 ± 3.22	
SSRR	1952.05 ± 1.73	1756.77 ± 62.98	1944.05 ± 5.48	

Table 3: Converged reward of GIE-CLF and the baselines in the inverted pendulum environment with demonstrations with different optimality.

Method	Inverted Pendulum	Neural Lander	F-16 GCAS	F-16 Tracking
GIE-CLF	$\mathbf{0.0014} \pm 0.0012$	$\mathbf{0.50} \pm 0.01$	$\mathbf{2202.10} \pm 219.26$	$\mathbf{310.54} \pm 124.44$
PPO	0.032 ± 0.035	1.09 ± 0.36	2450.37 ± 351.60	8267.63 ± 5.61
AIRL	0.19 ± 0.01	0.73 ± 0.06	2558.85 ± 201.13	550.84 ± 83.90
D-REX	0.23 ± 0.12	0.87 ± 0.36	2738.22 ± 83.67	4809.76 ± 2184.88
SSRR	0.040 ± 0.034	1.03 ± 0.31	2269.20 ± 824.33	8060.87 ± 410.69
CLF-sample	0.015 ± 0.013	0.72 ± 0.07	2404.74 ± 255.45	1874.60 ± 266.85
CLF-dense	0.0088 ± 0.0113	0.68 ± 0.08	2346.28 ± 139.02	1517.89 ± 494.08

Table 4: Comparison of GIE-CLF with baselines. In Inverted Pendulum, Neural Lander, and F-16 Tracking, we report the minimum distance to the goal of each algorithm. In F-16 GCA environment, we report the minimum dropping distance of each algorithm.

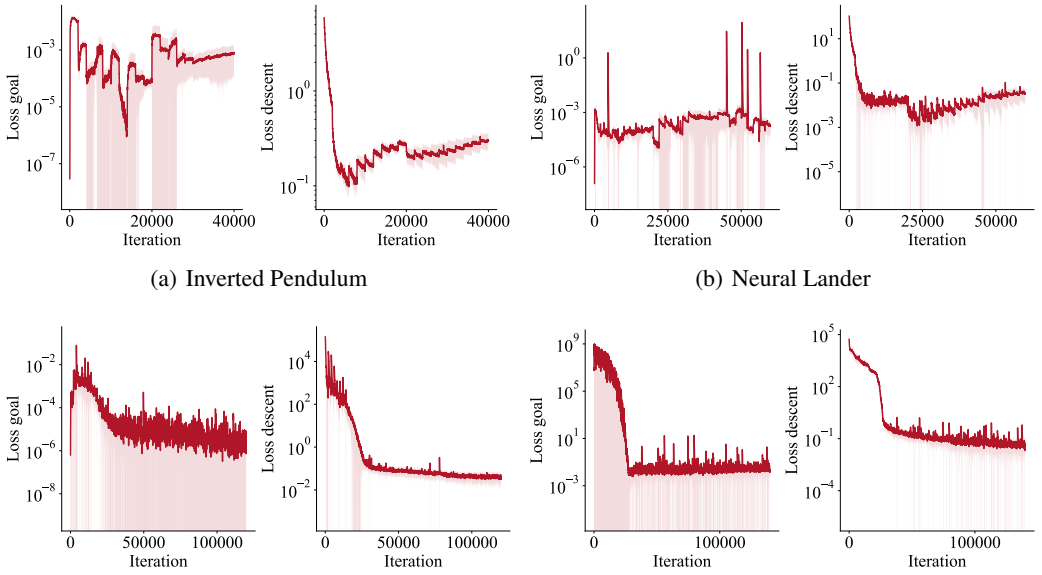
Minimum Distance to the Goal In the main text, we compared the expected rewards of GIE-CLF and the baselines w.r.t. the number of samples. Here we use another metric to further show the efficacy of GIE-CLF. Like Figure 3 in the main text, we compare the minimum distance to the goal in Inverted Pendulum, Neural Lander, and F-16 Tracking environments, and in F-16 GCAS environment, since every controller will pass the goal region (height between 800ft and 1200ft), we compare the minimum dropping distance instead. The results are shown in Table 4. We can observe that GIE-CLF outperforms all the baselines in all environments.

Training Losses In the theoretical results we provide, we make an assumption that the training losses of the framework should be small. The loss $\mathcal{L}_{\text{CLF}}$ is defined as:

$$\mathcal{L}_{\text{CLF}} = V_\theta(x_{\text{goal}})^2 + \frac{\eta_{\text{pos}}}{N} \sum_{x \in \mathcal{D}} \max \left[\epsilon + \frac{\partial V_\theta(x)}{\partial x} \frac{\hat{h}_\alpha(x, \pi_\phi(x))}{dt} + \lambda V_\theta(x), 0 \right] \tag{22}$$

We call the first term "Loss goal" and the second term "Loss descent". In Figure 5, we show the curves of the two loss terms w.r.t. the training iterations. We can observe that in all environments, "Loss goal" converges to about 10^{-3} and "Loss descent" converges to about 10^{-2}, which supports the assumptions we make in the theoretical results. In addition, because we use the PyTorch (Paszke et al., 2019) framework, which often initialize the neural networks to output 0 with 0 input, the initial "Loss goal" is very close to 0, and it remains low during the training process.

Videos for Learned Controllers We show the videos of the learned policies of the experiments in the file 'experiments.mp4' in the supplementary materials.



(a) Inverted Pendulum (b) Neural Lander

(c) F-16 GCAS (d) F-16 Tracking

Figure 5: The training loss of GIE-CLF w.r.t. the training iterations.

C Discussions

C.1 Verification of the Learned CLFs

Our focus of the 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. However, we do not claim that our learned CLF is exhaustively verified. Although the theories we provide show that the learned CLF is valid under some assumptions, these assumptions may not be completely satisfied in experiments. For example, the loss $\mathcal{L}_{\text{CLF}}$ may not be zero, and the trusted tunnel may have a large error, etc. If we want to verify the learned CLF is valid in the whole trusted tunnel, additional verification tools are needed, including SMT solvers (Gao et al., 2012; Chang et al., 2019), Lipschitz-informed sampling methods (Bobiti & Lazar, 2018), etc. However, these verification tools are known to have poor scalability, and scalable verification for the learned CLFs remains an open problem.

C.2 Construction of the Reachable Sets

In our approach, we use Monte Carlo method to approximate the reachable states of the learned controller to construct the trusted tunnel. However, this is not strict, since it is impossible to sample all the states in the initial set, and the discretization of the time can also cause issues. To formally construct the reachable set, we need additional neural network reachability analysis (Liu et al., 2021) tools, but they are hard to be used in high-dimensional systems because of their high computational complexity. Neural network reachability analysis in high-dimensional complex environments is still an open problem.

C.3 Possible Future Directions

Our algorithm can also benefit from the verification tools. For instance, neural network reachability analysis enables us to approximate the reachable set better than the Monte Carlo method we used, and SMT solvers allow us to find counterexamples to augment the training data to make our learned CLF converge faster. In addition, almost Lyapunov functions (Liu et al., 2020) 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.

D Details about the Extensions

D.1 Learning Control Contraction Matrices with Unknown Dynamics

In Section 7 in the main text, we introduced that our algorithm can also be directly applied to learn Control Contraction Matrices (CCMs) in environments with unknown dynamics. Here we provide more details.

We consider the control-affine systems

$$\dot{x} = f(x) + g(x)u \tag{23}$$

where $x \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$ is the state, $u \in \mathcal{U} \subseteq \mathbb{R}^{n_u}$ is the control input. The tracking problem we consider is to design a controller $u = \pi(x, x^*, u^*)$, such that the controlled trajectory $x(t)$ can track *any* target trajectory $x^*(t)$ generated by some reference control $u^*(t)$ when $x(0)$ is near $x^*(0)$.

CCMs are widely used to provide contraction guarantees for tracking controllers. A fundamental theorem in CCM theory (Manchester & Slotine, 2017) says that if there exists a metric $M(x)$ and a constant $\lambda > 0$, such that

$$g_\perp^\top \left(-\partial_f W(x) + \widehat{\frac{\partial f(x)}{\partial x} W(x)} + 2\lambda W(x) \right) g_\perp \prec 0 \tag{24a}$$

$$g_\perp^\top \left(\partial_{g_j} W(x) - \widehat{\frac{\partial g_j(x)}{\partial x} W(x)} \right) g_\perp = 0, \quad j = 1, ..., n_u \tag{24b}$$

where $g_\perp(x)$ is an annihilator matrix of $g(x)$ satisfying $g_\perp^\top(x) g(x) = 0$, $W(x) = M(x)^{-1}$ is the dual metric, g_j is the j-th column of matrix g, and for a matrix P, $\widehat{P} = P + P^\top$, then there exists a controller $u = \pi(x, x^*, u^*)$, such that the controlled trajectory $x(t)$ will converge to the reference trajectory $x^*(t)$ exponentially. Such controller can be find by satisfying the following condition:

$$\dot{M} + \widehat{M(A + gK)} + 2\lambda M \prec 0 \tag{25}$$

where $A = \frac{\partial f}{\partial x} + \sum_{i=1}^{n_u} u^i \frac{\partial g_i}{\partial x}$, u^i is the i-th element of the vector u, and $K = \frac{\partial u}{\partial x}$.

We use a similar framework as GIE-CLF to learn the CCM in unknown systems. Since CCM theories require the environment dynamics to be control-affine, we change the model of the dynamics to be $\hat{h}_{\alpha,\beta}(x, u) = f_\alpha(x) + g_\beta(x)u$, where $f_\alpha(x) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$ and $g_\beta(x) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x} \times \mathbb{R}^{n_u}$ are neural networks with parameters α and β. Given imperfect demonstrations, we still first use imitation learning to learn an initial controller $\pi_{\text{init}}(x, x^*, u^*)$, and fit the local model $\hat{h}_{\alpha,\beta}(x, u)$. In order to find $g_\perp(x)$, we need the learned $g_\beta(x)$ to be sparse so that we can hand-craft $g_\perp(x)$ for it, so we add the Lasso regression term in the loss $\mathcal{L}_{\text{dyn}}$:

$$\begin{aligned} \mathcal{L}_{\text{dyn}}(\alpha, \beta) = \frac{1}{N} \sum_{x(t), u(t), x(t+1) \in \mathcal{D}} \left\| x(t+1) - x(t) - \hat{h}_{\alpha,\beta}(x(t), u(t)) \right\|^2 \\ + \mu_{\text{dyn}}(\|\alpha\|^2 + \|\beta\|^2 + \|\beta\|_1) \end{aligned} \tag{26}$$

where $\|\beta\|_1$ is the 1-norm of β. Then, we jointly learn the controller and the corresponding CCM inside $\mathcal{H}$. We parameterize the controller and the dual metric using neural networks $\pi_\phi(x, x^*, u^*)$ and $W_\theta(x)$ with parameters ϕ and θ, and train them by replacing the CLF loss $\mathcal{L}_{\text{CLF}}$ in the main text with the following loss:

$$\mathcal{L}_{\text{CCM}} = \frac{1}{N} \sum_{(x, x^*, u^*) \in \mathcal{D}} \left[L_{\text{PD}}(-C_1(x; \theta)) + \sum_{j=1}^{n_u} \|C_2^j(x; \theta)\|_F + L_{\text{PD}}(-C_u(x, x^*, u^*; \phi)) \right] \tag{27}$$

where $C_1(x; \theta)$, $C_2(x; \theta)$, and $C_u(x, x^*, u^*; \phi)$ are the LHS of Equation (24a), Equation (24b), and Equation (25), respectively. $\|\cdot\|_F$ is the Frobenius norm. L_{PD} is the loss function to make its input

Method	Converged Reward	Number of Samples (k)	Mean Tracking Error
Ours	$\mathbf{1966.91} \pm 15.42$	$\mathbf{252}$	$\mathbf{0.0228} \pm 0.0073$
PPO	1547.52 ± 74.74	5000	0.597 ± 0.157
AIRL	1517.81 ± 119.11	5000	0.494 ± 0.255
D-REX	653.46 ± 114.71	5000	0.701 ± 0.166
SSRR	1334.08 ± 155.85	5000	0.476 ± 0.214

Table 5: Converged Reward, Number of Samples, and Tracking Error of our algorithm and baselines in the Dubins car path tracking environment

positive definite. In our implementation, we use $L_{\mathrm{PD}}(\cdot) = \frac{1}{N}\sum \mathrm{ReLU}(\lambda(\cdot))$, where $\lambda(\cdot)$ is the eigenvalues. Note that θ is not a parameter of C_u since we only use C_u to find the controller. For more detailed discussions of the loss functions, one can refer to Sun et al. (2021) and Chou et al. (2021). Once we have the learned controller, we apply it in the environment to collect more data and enlarge the trusted tunnel $\mathcal{H}$, following the same process of GIE-CLF. We repeat this process several times until convergence.

D.2 Experimental Details of CCM

D.2.1 Implementation Details

We implement our algorithm using the PyTorch framework (Paszke et al., 2019) based on the CCM repository [8] (Sun et al., 2021). The neural networks in the dynamics model $f_\alpha(x)$ and $g_\beta(x)$ have two hidden layers with size 128 and $\tanh$ as the activation function. We parameterize our controller using

$$\pi_\phi(x, x^*, u^*) = \omega_2(x, x^*) \cdot \tanh\left(\omega_1(x, x^*) \cdot (x - x^*)\right) + u^* \tag{28}$$

where $\omega_1(x, x^*)$ and $\omega_2(x, x^*)$ are two neural networks with two hidden layers with size 128 and $\tanh$ as the activation function. Therefore, we have $x = x^* \implies u = u^*$ by construction. We model the dual metric using

$$W_\theta(x) = C(x)^\top C(x) + \underline{\omega} I \tag{29}$$

where $C(x) \in \mathbb{R}^{n_x \times n_x}$ is a neural networks with two hidden layers with size 128 and $\tanh$ as the activation function, I is the identity matrix and $\underline{\omega}$ is the minimum eigenvalue. By construction, $W(x)$ is symmetric. We use ADAM as the optimizer with learning rate 3×10^{-4} to optimize the parameters. For the hyper-parameters, we set the convergence rate of CCM $\lambda = 0.5$, and the minimum eigenvalue $\underline{\omega} = 0.1$.

D.2.2 Environment

We test our algorithm in a Dubins car path tracking environment. The state of the Dubins car is $x = [p_x, p_y, \psi, v]^\top$, where p_x, p_y are the position of the car, ψ is the heading, and v is the velocity. The control input is $u = [\omega, a]\top$ where ω is the angular acceleration and a is the longitudinal acceleration. The dynamics is given by $\dot{x} = f(x) + g(x)u$, with

$$\begin{aligned} f(x) &= [v\cos\psi, v\sin\psi, 0, 0]^\top \\ g(x) &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \end{aligned} \tag{30}$$

We set the initial state with $p_x, p_y \in [-0.2, 0.2]$, $\psi \in [-0.5, 0.5]$, and $v \in [0, 0.2]$. For the demonstrations, we use the LQR controller solved with the error dynamics. We collect 20 trajectories for the demonstrations with randomly generated reference paths, where each trajectory has 1000 time steps. For the reward function, we use $r(x) = 2 - \|(p_x, p_y) - (p_x^*, p_y^*)\|$, where (p_x^*, p_y^*) is the position on the reference path.

[8] https://github.com/MIT-REALM/ccm

D.2.3 NUMERICAL RESULTS

In Section 7 in the main text, we compare the expected reward of our algorithm and the baselines. Here we provide more detailed numerical results. In Table 5, we show the converged reward, the number of samples used in training, and the mean tracking error of our algorithm and the baselines. We can observe that our algorithm achieves the highest reward and the lowest mean tracking error, with a large gap compared with other algorithms. In addition, the samples we used in training are more than one order of magnitude less than other algorithms.