Learning Lyapunov (Potential) Functions from Counterexamples and Demonstrations

Hadi Ravanbakhsh and Sriram Sankaranarayanan
Department of Computer Science, University of Colorado Boulder, Boulder, Colorado 80302
Email: firstname.lastname@colorado.edu

***Abstract*—We present a technique for learning control Lyapunov (potential) functions, which are used in turn to synthesize controllers for nonlinear dynamical systems. The learning framework uses a *demonstrator* that implements a black-box, untrusted strategy presumed to solve the problem of interest, a *learner* that poses finitely many queries to the demonstrator to infer a candidate function and a *verifier* that checks whether the current candidate is a valid control Lyapunov function. The overall learning framework is iterative, eliminating a set of candidates on each iteration using the counterexamples discovered by the verifier and the demonstrations over these counterexamples. We prove its convergence using ellipsoidal approximation techniques from convex optimization. We also implement this scheme using nonlinear MPC controllers to serve as demonstrators for a set of state and trajectory stabilization problems for nonlinear dynamical systems. Our approach is able to synthesize relatively simple polynomial control Lyapunov functions, and in that process replace the MPC using a guaranteed and computationally less expensive controller.**

I. Introduction

We propose a novel *learning from demonstration* scheme for inferring control Lyapunov functions (potential functions) for stabilizing nonlinear dynamical systems to reference states/ trajectories. Control Lyapunov functions (CLFs) have wide applications to motion planning problems in robotics. They extend the classic notion of Lyapunov functions to systems involving control inputs [3]. Finding a CLF also leads us to an associated feedback control law that can be used to solve the stabilization problem. Additionally, they can be extended for feedback motion planning using extensions to time-varying or sequential CLFs [5, 53]. Likewise, they have been investigated in the robotics community in many forms including *artificial potential functions* to solve path planning problems involving obstacles [30].

However, synthesizing CLFs for nonlinear systems remains a challenge [41, 42]. Standard approaches to finding CLFs include the use of dynamic programming, wherein the value function satisfies the conditions of a CLF [4], or using non-convex bilinear matrix inequalities (BMI) [13]. BMIs can be solved using alternating minimization methods [8, 51, 31]. However, these approaches often get stuck in local minima and exhibit poor convergence guarantees [12].

In this article, we investigate the problem of learning a CLF using a black-box *demonstrator* that can be queried with a given system state, and responds by demonstrating control inputs to stabilize the system starting from that state. However, our framework uses just the control input at the query state. Such a demonstrator can be realized using an expensive nonlinear model predictive controller (MPC) that uses a local optimization scheme, or even a human operator under certain assumptions [1]. The framework has a Learner which selects a candidate CLF and a Verifier that tests whether this CLF is valid. If the CLF is invalid, the Verifier returns a state at which the current candidate fails. The Learner queries the demonstrator to obtain a control input corresponding to this state. It subsequently eliminates the current candidate along with a set of related functions from further consideration. The framework continues to exhaust the space of candidate CLFs until no CLFs remain or a valid CLF is found in this process.

We prove the process can converge in finitely many steps provided the Learner chooses the candidate function appropriately at each step. We also provide efficient SDP-based approximations to the verification problem that can be used to drive the framework. Finally, we test this approach on a variety of examples, by solving stabilization problems for nonlinear dynamical systems. We show that our approach can successfully find simple CLFs using finite horizon nonlinear MPC schemes with appropriately chosen cost functions to serve as demonstrators. In these instances, the CLFs yield control laws that are computationally inexpensive, and guaranteed against the original dynamical model.

A. Illustrative Example: Tora System

Figure 1(a) shows a mechanical system consisting of a cart attached to a wall using a spring. The position of the cart x is controlled by an arm with a weight that can be moved back and forth by applying a force u, as shown. The goal is to stabilize the cart to $x = 0$, with its velocity, angle, and angular velocity $\dot{x} = \theta = \dot{\theta} = 0$. We refer the reader to the Jankovic et al. [17] for a derivation of the dynamics shown below in terms of state variables $(x_1, \ldots, x_4)$ and control input u_1, after a suitable change of basis transformation:

$$\dot{x_1} = x_2,\ \dot{x_2} = -x_1 + \epsilon \sin(x_3),\ \dot{x_3} = x_4,\ \dot{x_4} = u_1\,. \quad (1)$$

We approximate $\sin(x_3)$ using a degree 3 approximation which is quite accurate over the range $x_3 \in [-2, 2]$. The equilibrium $x = \dot{x} = \theta = \dot{\theta} = 0$ now corresponds to $x_1 = x_2 = x_3 = x_4 = 0$. The state space is taken to be $S : [-1, 1] \times [-1, 1] \times [-2, 2] \times [-1, 1]$, the control input $u_1 \in [-1.5, 1.5]$.

[1] We do not handle noisy or erroneous demonstrations in this paper

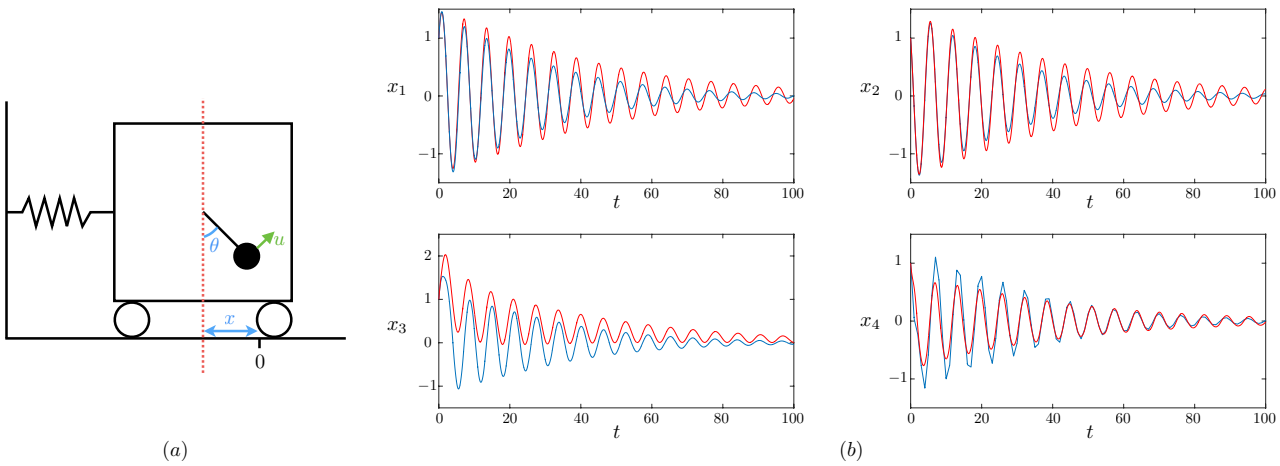


Fig. 1. Tora System. (a) A schematic diagram of the TORA system. (b) Execution traces of the system using MPC control (blue traces) and Lyapunov based control (red traces) starting from same initial point.

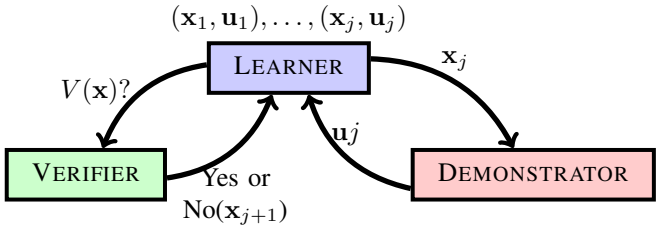


Fig. 2. Overview of the learning framework for learning a Lyapunov function.

MPC Scheme: A first approach to doing so uses a nonlinear model-predictive control (MPC) scheme using the time horizon $\mathcal{H} = 30$, time step $\tau = 1$ and a simple cost function

$$\sum_{t=[0,\tau,\ldots,\mathcal{H}]} \left(||\mathbf{x}(t)||_2^2 + ||\mathbf{u}(t)||_2^2\right) + \mathcal{H}\ ||\mathbf{x}(\mathcal{H})||_2^2 .$$

Given the model in (1), such a control is implemented using a first order numerical gradient descent method to minimize the cost function over a finite time horizon. The convergence was informally confirmed by observing hundreds of such simulations from different initial conditions for the system. However, the MPC scheme is expensive, requiring repeated solutions to (constrained) nonlinear optimization problems in real-time. Furthermore, the closed loop lacks formal guarantees despite the *high confidence* gained from numerous simulations.

Learning a Control Lyapunov Function: The approach in this paper uses the MPC scheme as a "DEMONSTRATOR", and attempts to learn a simpler control law through a control Lyapunov function. The key idea depicted in Fig. 2 is to pose queries to the MPC at finitely many *witness* states $W = \{\mathbf{x}_1, \ldots, \mathbf{x}_j\}$ and use the corresponding instantaneous control inputs $\mathbf{u}_1, \ldots, \mathbf{u}_j$, respectively. The LEARNER attempts to find a candidate function $V(\mathbf{x})$ that is positive definite and which decreases at each witness state $\mathbf{x}_j$ through the control input $\mathbf{u}_j$. This function is fed to the VERIFIER, which checks whether $V(\mathbf{x})$ is indeed a CLF, or discover a state $\mathbf{x}_{j+1}$ at which the condition fails. This new state is added to the witness set and the process is iterated.

The procedure described in this paper synthesizes the control Lyapunov function after 60 iterations of the learning loop and synthesizes the CLF $V(\mathbf{x})$ below:

$$\begin{pmatrix} 1.22\ x_2^2 + 0.31\ x_2x_3\ + 0.44\ x_3^2 - 0.28\ x_4x_2 + 0.8\ x_4x_3 \\ +1.69\ x_4^2 + 0.069\ x_1x_2 - 0.68\ x_1x_3 - 1.85\ x_4x_1 + 1.60\ x_1^2 \end{pmatrix}$$

This function yields a simple associated control law that can be implemented and guarantees the stabilization of the model (1). Figure 1(b) shows a closed loop trajectory of this control law vs control law extracted by MPC. The advantage of this law is that its calculation is *much simpler*, and furthermore, the control is formally guaranteed, at least for the model of the system.

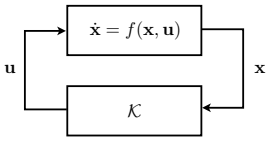
Contributions: In this paper, we instantiate the learning scheme sketched above, and show that under suitable assumptions terminates in finitely many iterations to either yield a control Lyapunov function $V(\mathbf{x})$ that is guaranteed to be valid, or show that Lyapunov function of a specific form does not exist. We demonstrate this scheme and its scalability on several interesting vehicle dynamics taken from the literature to solve stabilization to state and trajectory stabilization problems.

II. BACKGROUND

In this section, we briefly describe control Lyapunov (potential) functions. A state feedback control system $\Psi(X, U, \mathcal{P}, \mathcal{C})$ consists of a plant $\mathcal{P}$ and a controller $\mathcal{C}$ over state space $X \subseteq \mathbb{R}^n$ and input space $U \subseteq \mathbb{R}^m$. The plant $\mathcal{P}$ has a state $\mathbf{x} \in X$ and input $\mathbf{u} \in U$. The vector field for the plant is defined by a smooth function $f : X \times U \to \mathbb{R}^n$. Throughout the paper, we consider control affine systems that are possibly nonlinear in $\mathbf{x}$, but affine in $\mathbf{u}$, of the form

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) = f_0(\mathbf{x}) + \sum_{i=1}^{m} f_i(\mathbf{x}) u_i\,, \tag{2}$$

where $f_i : X \to \mathbb{R}^n$. The controller $\mathcal{C}$ measures the state of the plant ($\mathbf{x} \in X$) and provides feedback $\mathbf{u} \in U$. The controller is defined by a continuous feedback function $\mathcal{K} : X \to U$.



For a given feedback function $\mathcal{K}$, the execution trace of the system Ψ is defined as $\mathbf{x}(.) : \mathbb{R}^+ \rightarrow X$, which maps time to state. Formally, given $\mathbf{x}(0) = \mathbf{x}_0$, $\mathbf{x}(.)$ is defined according to $\dot{\mathbf{x}}(t) = f_0(\mathbf{x}(t)) + \sum_{i=1}^{m} f_i(\mathbf{x}(t))(\mathcal{K}(\mathbf{x}(t)))_i$, where $\dot{\mathbf{x}}(.)$ is the right derivative of $\mathbf{x}(.)$.

In this article, we study stabilization of nonlinear systems using Lyapunov functions (or potential function) inside a compact set S. More precisely, we consider a compact and connected set $S \subset X$. Without loss of generality, the origin $\mathbf{0}$ is the state we seek to stabilize to. Furthermore, $\mathbf{0} \in int(S)$. We restrict the set S to be a basic semi-algebraic set defined by a conjunction of polynomial inequalities. Likewise, the control inputs U are restricted to a polytope.

Our approach relies on control Lyapunov functions. A *control Lyapunov function* (CLF) [3] V, is a continuous function that respects the following conditions:

$$\begin{aligned} & V(\mathbf{0}) = 0 \\ (\forall \mathbf{x} \in S \setminus \{0\}) \quad & V(\mathbf{x}) > 0 \\ (\forall \mathbf{x} \in S \setminus \{0\})\ (\exists \mathbf{u} \in U) \quad & \nabla V \cdot f(\mathbf{x}, \mathbf{u}) < 0\,. \end{aligned} \tag{3}$$

The last condition ensures that the value of V can be decreased everywhere by choosing a proper feedback u. Let

$$V^{\bowtie \beta} = \{\mathbf{x} | V(\mathbf{x}) \bowtie \beta\}, \text{where } \ \bowtie \in \{=, \leq, <, \geq, >\}\,.$$

Let β^* be maximum β s.t. $V^{\leq \beta} \subseteq S$. Once a CLF is obtained, it guarantees that the system initialized to any state belonging to $V^{<\beta^*}$, can be stabilized to the origin (Fig. 3). A control Lyapunov function guarantees that there is a control strategy, which stabilizes the system.

Theorem 1: Given a control affine system Ψ, where $U : \mathbb{R}^m$ and a polynomial control Lyapunov function V satisfying Eq. (3), there is a feedback function $\mathcal{K}$ for which if $\mathbf{x}_0 \in V^{<\beta^*}$, then:

1) $(\forall t \geq 0)\ \mathbf{x}(t) \in S$
2) $(\forall \epsilon > 0)\ (\exists T \geq 0)\ \|\mathbf{x}(T) - \mathbf{0}\| < \epsilon\,.$

Given, a control Lyapunov function, it is possible to then obtain a feedback law in a closed form that stabilizes the system. Sontag [49] provides a method for extracting a continuous feedback function $\mathcal{K}$ for control affine systems from a control Lyapunov function. This can be extended to systems with constraints on the control inputs [33]. Also, feedback synthesis for periodic time/event triggered switching is possible [43, 7].

We have thus far considered the problem of stabilizing to a fixed equilibrium state. However, given this primitive, we can extend the CLF approach to related problems of (a) *Reach-while-stay:* reaching a given target set of states T starting from an initial set I, while staying inside a safe set S using Lyapunov-barrier functions [39, 44]; (b) *Trajectory stabilization:* stabilizing to a trajectory $\mathbf{x}(t)$ rather than to a fixed state using time-varying Lyapunov functions; or similarly, (c) *Feedback motion planning* which addresses the robustness of a plan using funnels [5, 53] ; (d) *Obstacles:* problems that involve reaching while avoiding an obstacle region in the state-space using artificial potential functions [30]. We will focus our exposition on the basic formulation for stability (Eq. (3)) while demonstrating extensions to some of other applications mentioned above through numerical examples.

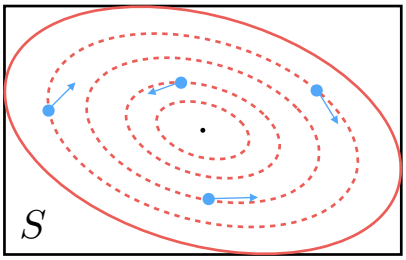


Fig. 3. Local Control Lyapunov Function (CLF): Level-sets of a CLF V are shown using the red lines. For each state (blue dot), the vector field $f(\mathbf{x}, \mathbf{u})$ for $\mathbf{u} = \mathcal{K}(\mathbf{x})$ is the blue arrow, and it points to a direction which decreases V. The β^*-level set of V ($V^{=\beta^*}$) is shown as a solid red line.

III. Algorithmic Learning Framework

Finding CLFs is known to be a hard problem, requiring the solution to BMIs [51] or hard polynomial constraints[43]. A standard approach to discovering such functions is to choose a set of basis functions $g_1, \ldots, g_r$ and search of a function of the form

$$V_{\mathbf{c}}(\mathbf{x}) = \sum_{j=1}^{r} c_j g_j(\mathbf{x})\,, \tag{4}$$

where $\mathbf{c} \in \mathbb{R}^r$ is vector of unknowns. One possible choice of basis functions involves monomials $g_j(\mathbf{x}) : \mathbf{x}^{\alpha_j}$ wherein $|\alpha_j|_1 \leq D_L$ for some degree bound D_L for the learning concept (CLF). Then, the problem is reduced to finding $\mathbf{c}$ s.t. $V_{\mathbf{c}}$ satisfies Eq. (3).

We now present the algorithmic learning framework. Let us fix a control affine system $\mathcal{P}$ over a state-space X, control inputs U given by (2). Let $\mathbf{x}^* = \mathbf{0}$ be the equilibrium we wish to stabilize the system to, while remaining inside $S \subset X$.

Next, we assume a DEMONSTRATOR as a function $\mathcal{D} : S \mapsto U$ that given a state $\mathbf{x} \in S$, provides us an appropriate feedback $\mathcal{D}(\mathbf{x}) \in U$ for the state $\mathbf{x}$, such that $\mathcal{D}$ is presumed to be a valid function that stabilizes the system.

Remark 1: Our definition of a demonstrator is general enough to allow offline MPC, sample based methods [27, 23], human operator demonstrations [21], or even demonstrations that rely on opaque models such as neural networks.

Also, we assume that the demonstrator is *presumed* correct. However, the approach can work even if the demonstrator may fail on some input states. Finally, a faulty demonstrator may, at the worst, lead our technique to fail without finding a CLF. In particular, such a demonstrator will not cause our technique to synthesize an incorrect CLF.

Definition 1 (Problem Statement): The CLF learning problem has the following inputs:

1) A dynamical system $\mathcal{P}$ in the form (2),

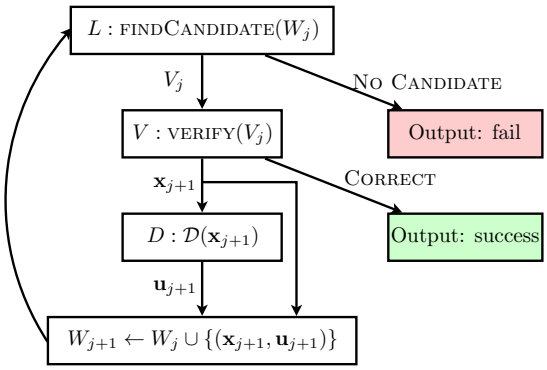


Fig. 4. Flowchart for the algorithmic learning framework. L: Learner, V: Verifier and D: Demonstrator.

2) A safe set S,
3) A "black-box" demonstrator function $\mathcal{D} : S \mapsto U$ that presumably stabilizes the system, and
4) A candidate space for CLFs of the form $\sum_{j=1}^{r} c_j g_j(\mathbf{x})$ given by basis functions $\mathbf{g}(\mathbf{x}) : \langle g_1(\mathbf{x}), \ldots, g_r(\mathbf{x}) \rangle$ and a compact set $C \ni (c_1, \ldots, c_r)$. We represent the coefficients $(c_1, \ldots, c_r)$ collectively as $\mathbf{c}$.

The output can be SUCCESS: a function $V_c(\mathbf{x}) : \mathbf{c}^t \cdot \mathbf{g}(\mathbf{x})$ that is a CLF; or FAILURE: no function could be discovered by our procedure.

A. Algorithmic Learning Framework

The algorithmic learning framework is shown in Fig. 4, and implements two modules (a) LEARNER and (b) VERIFIER that interact with each other and the demonstrator. The framework works iteratively until termination. At the j^{th} iteration, the learner maintains a (witness) set

$$W_j : \{(\mathbf{x}_1, \mathbf{u}_1), \ldots, (\mathbf{x}_j, \mathbf{u}_j)\} \subseteq S \times U \,.$$

W_j is a finite set of pairs of states $\mathbf{x}_i$ and corresponding demonstrated feedback $\mathbf{u}_i$. Corresponding to W_j, $C_j \subseteq C$ is defined as set of candidate coefficients for functions $V_{\mathbf{c}}(\mathbf{x}) : \mathbf{c}^t \mathbf{g}(\mathbf{x})$ with $\mathbf{c} \in C_j$. Formally, C_j is a set of all candidates $\mathbf{c}$ s.t. $V_{\mathbf{c}}$ satisfies the CLF condition (3) for every point in the finite set W_j:

$$C_j : \left\{ \mathbf{c} \in C \;\middle|\; \bigwedge_{(\mathbf{x}_i, \mathbf{u}_i) \in W_j} \begin{array}{c} V_{\mathbf{c}}(\mathbf{x}_i) > 0 \wedge \\ \nabla V_c \cdot f(\mathbf{x}_i, \mathbf{u}_i) < 0 \end{array} \right\} . \quad (5)$$

The flowchart for the overall procedure is shown in Fig. 4.

To begin with, $W_0 : \emptyset$ and $C_0 : C$. Each iteration works as follows:

1) The learner samples a value $\mathbf{c}_j \in C_j$ and outputs the corresponding function $V_j(\mathbf{x}) : \mathbf{c}_j^t \cdot \mathbf{g}(\mathbf{x})$. If $C_j = \emptyset$ then no sample is found and the algorithm fails.
2) The verifier checks if V_j is a CLF by checking the conditions in (3). If V_j satisfies the conditions, then the algorithm stops to declare success. Otherwise the verifier selects a (counterexample) state $\mathbf{x}_{j+1} \in S$ for which the CLF condition fails. Assume without loss of generality that $\mathbf{x}_{j+1} \neq \mathbf{0}$.
3) Failing verification, the demonstrator is called to choose a suitable control $\mathbf{u}_{j+1}$ corresponding to $\mathbf{x}_{j+1}$.
4) The new set $W_{j+1} := W_j \cup \{(\mathbf{x}_{j+1}, \mathbf{u}_{j+1})\}$. Furthermore,

$$C_{j+1} : \; C_j \cap \left\{ \mathbf{c} \;\middle|\; \begin{array}{c} V_{\mathbf{c}}(\mathbf{x}_{j+1}) > 0 \wedge \\ \nabla V_{\mathbf{c}} \cdot f(\mathbf{x}_{j+1}, \mathbf{u}_{j+1}) < 0 \end{array} \right\} . \quad (6)$$

We now prove some core properties that guarantee the correctness of the proposed scheme. We assume that the learner and the verifier are implemented without any approximations/relaxations (as will be subsequently presented).

Theorem 2: The algorithmic learning framework as described above has the following property:

1) $\mathbf{c}_j \notin C_{j+1}$. I.e, the candidate found at the j^{th} step is eliminated from further consideration.
2) If the algorithm succeeds at iteration j, then the output function $V_j(\mathbf{x})$ is a valid CLF for stabilization.
3) The algorithm declares failure at iteration j if and only if no linear combination of the basis functions is a CLF compatible with the demonstrator.

Proofs are available in [45].

Inverse results [38] suggest polynomial basis for Lyapunov functions are expressive enough for verification of exponentially stable, smooth nonlinear systems. This, justifies using polynomial basis for CLF.

Lemma 3.1: Assuming (i) the demonstrator function $\mathcal{D}$ is smooth, (ii) the closed loop system with controller $\mathcal{D}$ is exponentially stable, then there exists a polynomial CLF, compatible with $\mathcal{D}$.

Lemma 3.1 guarantees success of the learning procedure if the set of basis functions is rich enough. We now present implementations of each of the modules involved, starting with the learner.

B. Learner: Finding a Candidate

The FINDCANDIDATE function simply samples a point from the set C_j defined in Eq. (5). Note that $V_{\mathbf{c}}(\mathbf{x}_j) : \mathbf{c}^t \cdot \mathbf{g}(\mathbf{x}_j)$ is linear in $\mathbf{c}$ and therefore $\nabla V_{\mathbf{c}} . f(\mathbf{x}_j, \mathbf{u}_j)$ is linear in $\mathbf{c}$ as well. The initial space of all candidates C is assumed to be a hyper-rectangular open box. At each iteration, the candidate $\mathbf{c}_j \in C_j$ is chosen. Suppose the algorithm does not terminate at this iteration. According to Eq. (6), the new set C_{j+1} is obtained as $C_{j+1} : \; C_j \cap H_j$, wherein H_j is defined by two linear inequalities $H_{j1} \wedge H_{j2}$ ($H_{j1} : \mathbf{a}_{j1}^t \mathbf{c} < b_{j1}$, $H_{j2} : \mathbf{a}_{j2}^t \mathbf{c} < b_{j2}$). Let $\overline{C_j}$ represent the topological closure of the set C_j.

Lemma 3.2: For each $j \geq 0$, $\overline{C_j}$ is a convex polyhedron.

Thus the problem of implementing findCandidate is that of checking emptiness of a polyhedron with some strict inequality constraints. This is readily solved using a slight modification of standard linear programming algorithms using infinitesimals for strict inequalities.

Lemma 3.3: There exists a halfspace $H_{jk} : \; \mathbf{a}_{jk}^t \mathbf{c}_j \geq b_{jk}$ that passes through $\mathbf{c}_j$, and $C_{j+1} \subseteq C_j \cap H_{jk}$.

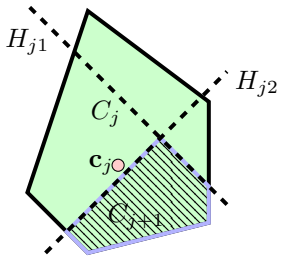


Fig. 5. Original candidate region C_j (green) at the start of the j^{th} iteration, the candidate $\mathbf{c}_j$, and the new region C_{j+1} (hatched region with blue lines). Also, $\mathbf{a}_{j2}\mathbf{c}_j > b_{j2}$ (H_{j2} passes through $\mathbf{c}_j$)

Choosing a Candidate $\mathbf{c}_j$: The choice of $\mathbf{c}_j \in C_j$ governs the overall convergence of the algorithm. Figure 5 demonstrates the importance of this choice by showing the candidate $\mathbf{c}_j$, the hyperplane H_{jk} and the new region C_{j+1}.

We wish to choose $\mathbf{c}_j$ s.t. $\textsc{Vol}(C_{j+1}) \leq \alpha \textsc{Vol}(C_j)$, for a fixed α, independent of H_{jk}.

Theorem 3 (Tarasov et al.[52]): Let $\mathbf{c}_j$ be chosen as the center of the maximum volume ellipsoid (MVE) of C_j. Then,

$$\textsc{Vol}(C_{j+1}) \leq \left(1 - \frac{1}{r}\right) \textsc{Vol}(C_j) \ .$$

This leads us to a scheme that guarantees termination of the overall iterative scheme in finitely many steps under a robustness assumption.

Definition 2 (Robust Candidate): A candidate $\mathbf{c} \in C$ is δ-robust ($\delta > 0$), iff for each $\hat{\mathbf{c}} \in \mathcal{B}_\delta(\mathbf{c})$, $V_{\hat{\mathbf{c}}} : \hat{\mathbf{c}}^t \cdot \mathbf{g}(\mathbf{x})$ is a CLF, where $\mathcal{B}_\delta(\mathbf{c})$ is a ball of radius δ around $\mathbf{c}$.

Let $\textsc{Vol}(C) < \gamma\Delta^r$ for $\gamma > 0$ the volume of the unit r-ball, and $\Delta > 0$. Furthermore, the procedure terminates whenever $\textsc{Vol}(C_j) < \gamma\delta^r$ following the robustness assumption above.

Theorem 4: If at each step $\mathbf{c}_j$ is chosen as the center of the maximum volume ellipsoid in C_j, the learning loop terminates in at most

$$\frac{r(\log(\Delta) - \log(\delta))}{-\log\left(1 - \frac{1}{r}\right)} = O(r^2) \text{ iterations} \, .$$

The maximum volume ellipsoid itself can be computed by solving a convex optimization problem[55].

C. Implementing the Verifier

The verifier given a candidate $V_j(\mathbf{x}) : \mathbf{c}_j^t \cdot \mathbf{g}(\mathbf{x})$ checks the CLF conditions in Eq. (3), split into two separate checks:
(A) Check if $V_j(\mathbf{x})$ is a positive definite polynomial over $\mathbf{x} \in S$. Assuming that $\mathbf{g}(\mathbf{0}) = 0$ for the basis, this reduces to:

$$(\exists \ \mathbf{x} \in S \setminus \{\mathbf{0}\}) \ V_j(\mathbf{x}) \leq 0 \, . \tag{7}$$

(B) Check if the Lie derivative of V_j can be made negative for each $\mathbf{x} \in S$ by a choice $\mathbf{u} \in U$:

$$(\exists \mathbf{x} \in S \setminus \{\mathbf{0}\}) \ (\forall \mathbf{u} \in U) \ (\nabla V_j) \cdot f(\mathbf{x}, \mathbf{u}) \geq 0 \, . \tag{8}$$

This problem *seems* harder due to the presence of a *quantifier alternation*. Let U be a polyhedral set defined by $U : \ \{\mathbf{u} \in \mathbb{R}^m \mid A\mathbf{u} \geq \mathbf{b}\}$. Recall that $f(\mathbf{x}, \mathbf{u})$ is control affine function $f_0(\mathbf{x}) + \sum_{i=1}^m f_i(\mathbf{x}) u_i$.

Lemma 3.4: Eq. (8) holds for some $\mathbf{x} \in S$ iff

$$\begin{aligned} (\exists \ \mathbf{x} \in S \setminus \{\mathbf{0}\}, \lambda) \ \ & \lambda \geq \mathbf{0}, \lambda^t \mathbf{b} \geq -\nabla V_j . f_0(\mathbf{x}) \\ & A_i^t \lambda = \nabla V_j . f_i(\mathbf{x}) (i \in \{1 \ldots m\}). \end{aligned} \tag{9}$$

In other words, assuming that the dynamics and chosen bases are polynomials, the verification problem reduces to checking if a given semi-algebraic set defined by polynomial inequalities has a solution.

Failing the polynomial assumption, the problem of verification is *in general* undecidable. However, it can be approximated by techniques such as δ-decision procedures proposed by Gao et al [10]. Solvers like dReal can thus be directly used for the verification problem. While dReal does a good job in adaptive space decomposition, in our experience, they do not scale *reliably*. Nevertheless, these solvers allow us to conveniently implement a verifier for small but hard problems involving rational and trigonometric functions.

The verification problem for polynomial systems and polynomial CLFs through a polynomial basis function $\mathbf{g}(\mathbf{x})$ is NP-hard, in general. Exact approaches using semi-algebraic geometry and the branch-and-bound solvers (including the dReal approach cited above) can tackle this problem precisely. However, scalability is still an issue.

We sketch a relaxation using SDP solvers [35]. Let us fix a basis of monomial terms of degree up to D_V, $\mathbf{m}^t : [1 \ x_1 \ \ldots \ \mathbf{x}^{D_V}]$, wherein D_V is chosen as at least half of the maximum degree in $\mathbf{x}$ among all monomials in $g_j(\mathbf{x})$ and $\nabla g_j \cdot f(\mathbf{x})$: $D_V \geq \frac{1}{2} \max\left(\bigcup_j \left(\{\deg(g_j)\} \cup \{\deg(\nabla g_j \cdot f)\}\right)\right)$. Let $Z(\mathbf{x}) : \mathbf{m}\mathbf{m}^t$. Thus, each polynomial of degree up to $2D_V$ may now be written as a trace inner product $p(\mathbf{x}) = \langle P, Z(\mathbf{x}) \rangle = \mathsf{trace}(PZ(\mathbf{x}))$, wherein the matrix P is constant.

Let S be the semi-algebraic set defined as

$$S : \{\mathbf{x} \in \mathbb{R}^n \mid r_1(\mathbf{x}) \leq 0, \ldots, r_k(\mathbf{x}) \leq 0\} \, ,$$

for polynomials $r_1, \ldots, r_k$. The constraint in (7) is equivalent to solving the following optimization problem over $\mathbf{x}$

$$\begin{aligned} & \mathsf{max}_{\mathbf{x}} \ \langle I, Z(\mathbf{x}) \rangle \\ & \mathsf{s.t.} \qquad \langle R_i, Z(\mathbf{x}) \rangle \leq 0, \ i \in \{1, \ldots, k\} \\ & \qquad\quad \langle \mathcal{V}_j, Z(\mathbf{x}) \rangle \leq 0 \, , \end{aligned} \tag{10}$$

where $V_j(\mathbf{x})$ ($r_i(\mathbf{x})$) is written as $\langle \mathcal{V}_j, Z(\mathbf{x}) \rangle$ ($\langle R_i, Z(\mathbf{x}) \rangle$). and those in (9) are written as

$$\begin{aligned} & \mathsf{max}_{\mathbf{x},\lambda} \ \langle I, Z(\mathbf{x}) \rangle \\ & \mathsf{s.t.} \qquad \langle R_i, Z(\mathbf{x}) \rangle \leq 0, \ i \in \{1, \ldots, k\} \\ & \qquad\quad \langle F_{ji}, Z(\mathbf{x}) \rangle = A_i^t \lambda, \ i \in \{1, \ldots, m\} \\ & \qquad\quad \langle -F_{j0}, Z(\mathbf{x}) \rangle \leq \mathbf{b}^t \lambda, \ \lambda \geq 0 \, , \end{aligned} \tag{11}$$

wherein the components $\nabla V_j \cdot f_i(\mathbf{x})$ defining the Lie derivatives of V_j are now written in terms of $Z(\mathbf{x})$ as $\langle F_{ji}, Z(\mathbf{x}) \rangle$.

The SDP relaxation is used to solve these problems and provide an upper bound of the solution [14]. The result of the relaxation treats $Z(\mathbf{x})$ as a matrix variable Z that will satisfy $Z \succeq 0$. Notice that each optimization problem is feasible simply by setting Z and λ to be zero. However, if the optimal solution of both problems is $Z = 0$ in the SDP relaxation, then we will conclude that the given candidate is a CLF.

Lemma 3.5: If the relaxed optimization problems in Eqs. (10) and (11) yield a zero solution, then the given candidate $V_j(\mathbf{x})$ is in fact a CLF.

However, the converse is not true. It is possible for $Z \succ 0$ to be optimal for either relaxed condition, but no $\mathbf{x} \in \mathbb{R}^n$ corresponds to the solution. This happens because the relaxation drops two key constraints to convexify the conditions: (1) Z has to be a rank one matrix written as $Z : \mathbf{m}\mathbf{m}^t$ and (2) there is a $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{m}$ is the matrix of monomials corresponding to $\mathbf{x}$.

To deal with this, we adapt our learning framework to work with witnesses $W_j : \{(Z_i, \mathbf{u}_i)\}_{i=1}^{j}$ replacing states $\mathbf{x}_i$ by matrices Z_i.

1) Each basis function $g_j(\mathbf{x})$ in $\mathbf{g}$ is now written instead as $\langle G_j, Z\rangle$. The candidates are therefore, $\sum_{j=1}^{r} c_j \langle G_j, Z\rangle$. Likewise, we write the components of its Lie derivative $\nabla g_j \cdot f_i$ in terms of Z.
2) The learner maintains the set W as $\{(Z_j, \mathbf{u}_j)\}$, wherein Z_j is the feasible solution returned by the SDP solver while solving Eqs. (11) and (10). In other words, the CLF conditions are, in fact, taken to be the relaxed conditions.
3) We use a suitable projection operator π mapping each Z to a state $\mathbf{x} : \pi(Z)$, such that the demonstrator receives $\mathbf{x}$. In practice, since the vector of monomials used to define Z from $\mathbf{x}$ includes the terms $1, x_1, \ldots, x_n$, the projection operator simply selects a few entries from Z corresponding to the variables. Other more sophisticated projections are also possible.

The relaxed framework thus lifts counterexamples to work over matrices Z_j. However, the candidate space begins with C and is refined each step as before. I.e, the relaxed framework continues to satisfy Lemmas 3.2, 3.3, Theorem 4 and Theorem 2 with the definition of (control) Lyapunov function changed to relaxed conditions.

IV. Experiments

In this section, we describe numerical results on some example benchmark systems. The algorithmic framework is implemented using quadratic template forms for the CLFs with the tool Globoptipoly used to implement the verifier [13]. The demonstrator is implemented using a nonlinear MPC implemented using a gradient descent algorithm. For each benchmark, we tuned the time horizon, discretization step and the cost function until the control objectives were satisfied by the MPC over hundreds of simulations starting from randomly selected initial states.

Most of the benchmarks consider a *reach-while-stay* problem, wherein the goal is to reach target set T, starting initial set I, while remaining in the safe set S. We also illustrate an example involving a trajectory stabilization problem. All the computations are performed on a Mac Book Pro with 2.9 GHz Intel Core i7 processor and 16GB of RAM. The reported CLFs are rounded to 2 decimal points. The summary of results is provided in Table. I.

Bicycle Model: This system is two-wheeled mobile robot modeled with five states $[x, y, v, \theta, \gamma]$ and two control inputs [9]. The goal is to stabilize the car to a target velocity $v^* = 5$, as shown in Fig. 6. We drop the variable x (since it is immaterial to our stabilization problem) and obtain a model with four state variables:

$$\begin{bmatrix} \dot{y} \\ \dot{v} \\ \dot{\theta} \\ \dot{\sigma} \end{bmatrix} = \begin{bmatrix} v\sin(\theta) \\ u_1 \\ v\sigma \\ u_2 \end{bmatrix}, \quad \begin{array}{l} U : [-10, 10] \times [-10, 10] \\ S : [-2, 2] \times [3, 7] \times [-1, 1]^2 \\ I : \mathcal{B}_{0.4}(\mathbf{0}) \\ T : \mathcal{B}_{0.1}(\mathbf{0}), \end{array}$$

where $\sigma = tan(\gamma)$ (see Fig. 6). sin function is approximated with a polynomial of degree 1. The method finds the following CLF:

$$\begin{aligned} V = &+ 0.42y^2 + 0.59y\theta + 2.57\theta^2 + 0.79y\sigma + 4.64\sigma\theta \\ &+ 4.06\sigma^2 - 0.38vy + 1.46v\theta + 1.18v\sigma + 2.39v^2 . \end{aligned}$$

Inverted Pendulum on a Cart: This example has applications in balancing two-wheeled robots [6] (cf. [1] for list of such robots). The system has four state variables $[x, \dot{x}, \theta, \dot{\theta}]$ and one input u. The dynamics, after partial linearization, have the following form [24]:

$$\begin{bmatrix} \ddot{\mathbf{x}} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 4u + \frac{4(M+m)g\tan(\theta) - 3mg\sin(\theta)\cos(\theta)}{4(M+m) - 3m\cos^2(\theta)} \\ \frac{-3u\cos(\theta)}{l} \end{bmatrix},$$

where $m = 0.21$, $M = 0.815$, $g = 9.8$ and $l = 0.305$. The trigonometric and rational functions are approximated with polynomials of degree 3. The sets are $S : [-1, 1]^4, U : [-20, 20], I : \mathcal{B}_{0.2}(\mathbf{x}), T : \mathcal{B}_{0.1}(\mathbf{x})$. We obtain the following

TABLE I
Results on the numerical examples. n: # variables, m: # control inputs, τ: MPC time step, $\mathcal{H}$: MPC time horizon, D_V: SDP relaxation degree bound for the Verifier, #Itr: # of iterations, Time: total computation time (minutes).

System Name	n	m	τ	$\mathcal{H}$	D_V	# Itr	Time
Tora	4	1	1	30	4	53	30
Bicycle	4	2	0.4	8	3	51	9
Bicycle × 2	8	4	0.4	8	3	536	303
Inverted Pendulum	4	1	0.04	2	5	85	31
Forward Flight	4	2	0.4	16	5	32	26
Hover Mode	6	2	0.4	16	4	213	163
Unicycle (Seg.1)	3+1*	2	0.1	2	4	41	15
Unicycle (Seg.2)	3+1*	2	0.1	3	4	31	7

* +1 refers to the time variable.

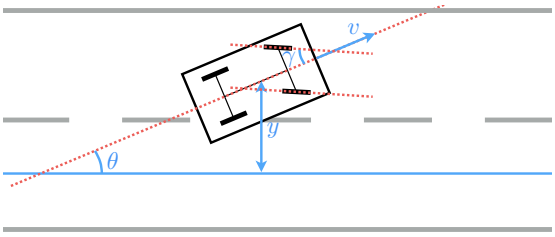


Fig. 6. A Schematic View of the Bicycle Model for Stabilizing to the Road.

CLF:

$$V = +11.64\dot{\theta}^2 + 45.93\dot{\theta}\theta + 85.47\theta^2 + 12.15x\dot{\theta} + 36.57x\theta + 6.44x^2 + 15.07\dot{\theta}\dot{x} + 33.06\dot{x}\theta + 8.98\dot{x}x + 6.09x^2 .$$

Forward Flight for Caltech Ducted Fan: The Caltech ducted fan models the aerodynamics of a single wing of a thrust vectored, fixed wing aircraft [16]. This problem is to design forward flight control in which the angle of attack needs to be changed. The model of the system is carefully calibrated through wind tunnel experiments. The system has 4 states $[v, \gamma, \theta, q]$ and two control inputs: u and δ_u. The dynamics are:

$$\begin{bmatrix} m\dot{v} \\ mv\dot{\gamma} \\ \dot{\theta} \\ J\dot{q} \end{bmatrix} = \begin{bmatrix} -D(v,\alpha) - W\sin(\gamma) + u\cos(\alpha+\delta_u) \\ L(v,\alpha) - W\cos(\gamma) + u\sin(\alpha+\delta_u) \\ q \\ M(v,\alpha) - ul_T\sin(\delta_u) \end{bmatrix},$$

where $\alpha = \theta - \gamma$, and D, L, and M are polynomials in v and α. For full list of parameters, see [16]. The goal is to reach $\mathbf{x}^*$: $[6, 0, 0.1771, 0]$ as the steady state. We perform a translation so that the $\mathbf{x}^*$ is the origin in the new coordinate system.

$$\begin{aligned} U &: [0, 13.5] \times [-0.45, 0.45] \\ S &: [3, 9] \times [-0.75, 0.75] \times [-0.75, 0.75] \times [-2, 2]] \\ I &: \{[v, \gamma, \theta, q] | (4v)^2 + (10\gamma)^2 + (10\theta)^2 + (10q)^2 < 4^2\} \\ T &: \{[v, \gamma, \theta, q] | (4v)^2 + (10\gamma)^2 + (10\theta)^2 + (10q)^2 < 0.5^2\} . \end{aligned}$$

First, we approximate v^{-1}, sin and cos with polynomials of degree 1, 3 and 3 respectively, to get polynomial dynamics. However, the system is not affine in control. We replace u and δ_u input variables with $u_1 = u\sin(\delta_u)$ and $u_2 = u\cos(\delta_u)$ and U is under-approximated with a polytope. These changes yield a polynomial control affine dynamics, which fits the description of our model. The procedure finds the following CLF:

$$V = +3.21q^2 + 2.18q\theta + 3.90\theta^2 + 0.40qv - 0.15v\theta + 0.56v^2 + 1.78q\gamma - 1.42\gamma\theta - 0.11v\gamma + 3.90\gamma^2 .$$

Hover Mode for Caltech Ducted Fan: This problem is again taken from [16]. The goal is to keep the ducted fan in a hover mode. The system has 6 variables x, y, θ, $\dot{x}$, $\dot{y}$, $\dot{\theta}$ and two control inputs u_1, u_2. The dynamics are

$$\begin{bmatrix} m\ddot{x} \\ m\ddot{y} \\ J\ddot{\theta} \end{bmatrix} = \begin{bmatrix} -d_c\dot{x} + u_1\cos(\theta) - u_2\sin(\theta) \\ -d_c\dot{y} + u_2\cos(\theta) + u_1\sin(\theta) - mg \\ ru_1 \end{bmatrix},$$

where $m = 11.2$, $g = 0.28$, $J = 0.0462$, $r = 0.156$ and $d_c = 0.1$. The sets are:

$$\begin{aligned} S &: [-1, 1] \times [-1, 1] \times [-0.7, 0.7] \times [-1, 1]^3 \\ U &: [-10, 10] \times [0, 10], I : \mathcal{B}_{0.25}(\mathbf{0}), T : \mathcal{B}_{0.05}(\mathbf{0}) . \end{aligned}$$

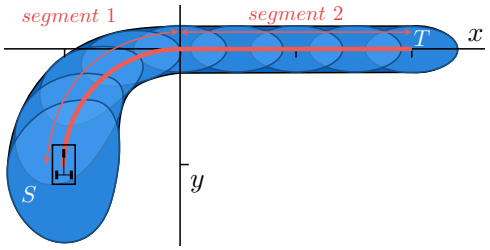


Fig. 7. Trajectory Tracking using Time-varying CLF - Projected on x-y plane. The reference trajectory is shown with the red line, consists of two segments. Starting from S, the state remains in the funnel (blue region) until it reaches T. Boundary of each smaller blue region shows a level-set of V for a specific time.

The sin and cos are approximated with polynomials of degree 2 and the procedure finds a quadratic CLF:

$$V = 1.63\dot{\theta}^2 - 1.09\dot{\theta}\dot{y} + 15.00\dot{y}^2 - 0.02\dot{\theta}y + 1.54y\dot{y} + 1.25y^2 + 1.74\theta\dot{\theta} + 0.79\dot{y}\theta + 0.08y\theta + 4.86\theta^2 - 4.93\dot{\theta}\dot{x} + 0.57\dot{x}\dot{y} + 0.05y\dot{x} - 8.26\dot{x}\theta + 12.58\dot{x}^2 - 0.44\dot{\theta}x - 0.38\dot{y}x - 0.20yx - 4.27x\theta + 3.86x\dot{x} + 2.14x^2 .$$

Unicycle: Now, we consider a simpler system, namely the unicycle model [28], which is known not to have a polynomial CLF. However, for trajectory tracking problem, one can provide a time varying CLF [53] for a finite time horizon control (using funnels). The unicycle model has the dynamics: $\dot{x_1} = u_1, \dot{x_2} = u_2, \dot{x_3} = x_1u_2 - x_2u_1$.

We consider a planning problem: starting near $[\frac{\pi}{2}, -1, -1]$, the goal is to reach near $[0, 2, 0]$, and by near we mean within distance 1. In the first step, a feasible trajectory $\mathbf{x}(t)$ is generated as shown in Fig. 7. Then $\mathbf{x}(t)$ is approximated with piecewise polynomials. More precisely, trajectory consists of two segments. The first segment brings the car to the origin and the second segment moves the car to the destination. Each segment is approximated using polynomials in t with degree up to 3. The time varying CLF V is a function of both the state $\mathbf{x}$ and time t. Choosing a template for V, our method is applied to this problem and we are able to find a strategy to implement the plan with guarantees. A level-set of the funnel (CLF) is shown in Fig. 7.

V. Related Work

Synthesis of Lyapunov Functions from Data: The problem of synthesizing Lyapunov functions for a control system by observing the states of the system in simulation has been investigated in the past by Topcu et al to learn Lyapunov functions along with the resulting basin of attraction [54]. Whereas the original problem is bilinear, the use of simulation data makes it easier to postulate states that belong to the region of attraction and therefore find Lyapunov functions that belong to this region by solving LMIs in each case. The application of this idea to larger black-box systems is demonstrated by Kapinski et al [20]. Our approach focuses on controller synthesis through learning a control Lyapunov function in a

bid to replace an existing controller. In doing so, we do not attempt to prove that the original demonstrator is necessarily correct but find a control Lyapunov function by assuming that the demonstrator is able to stabilize the system for the initial conditions we query on. Another important contribution lies in our analysis of the convergence of the learning with a bound on the maximum number of queries needed. In fact, these results can also be applied to the Lyapunov function synthesis approaches mentioned earlier.

Counter-Example Guided Inductive Synthesis (CEGIS): Our approach of alternating between a learning module that proposes a candidate and a verification module that checks the proposed candidate is identical to the CEGIS framework originally proposed by Solar-Lezama et al. [48, 47]. As such, the CEGIS approach does not include a demonstrator that can be queried. The extension of this approach Oracle-guided inductive synthesis [18], generalizes CEGIS using an *oracle* that serves a similar role as a demonstrator in this paper. Jha et al. prove bounds on the number of queries for discrete concept classes using results on exact concept learning in discrete spaces [11].

The CEGIS procedure has been used for the synthesis of CLFs recently by authors [43, 44], combining it with SDP solvers for verifying CLFs and a robust version for switched systems. The key difference here lies in the use of the demonstrator module that simplifies the learning module. In the absence of a demonstrator module, the problem of finding a candidate reduces to solving linear constraints with disjunctions, an NP-hard problem [44]. Likewise, the convergence results are quite weak [43]. In the setting of this paper, however, the use of a MPC scheme as a demonstrator allows us to use faster LP solvers and provide convergence guarantees.

Learning from Demonstration: The idea of learning from demonstration has had a long history in robotics [2]. The overall framework uses a demonstrator that can in fact be a human operator [21] or a complex MPC-based control law [19, 57, 58, 34]. The approaches differ on the nature of the interaction between the learner and the demonstrator; as well as how the policy is inferred. Our approach stands out in many ways: (a) We represent our policies by CLFs which are polynomial. On one hand, these are much less powerful than approaches that use neural networks [57, 19], for instance. However, the advantage lies in our ability to solve verification problems to ensure that the resulting policy learned through the CLF is correct with respect to the underlying dynamical model. (b) Our framework is *adversarial*. The choice of the counterexample to query the demonstrator comes from a failed attempt to validate the current candidate. (c) Finally, we use simple yet powerful ideas from convex optimization to place bounds on the number of queries, paralleling some results on concept learning in discrete spaces [11].

Control Lyapunov functions were originally introduced by Artstein and the construction of a feedback law given a CLF was first given by Sontag [3, 49]. As such, the problem of learning CLFs is well known to be hard, involving bilinear matrix inequalities (BMIs) [51]. An equivalent approach involves solving bilinear problems simultaneously for a control law and a Lyapunov function certifying it [8, 31]. BMIs are well known to be NP-hard, and hard to solve for a feasible solution [13]. The common approach is to perform an alternating minimization by fixing one set of bilinear variables while minimizing in the other. Such an approach has poor guarantees in practice, often "getting stuck" on a saddle point that does not allow the technique to make progress in finding a feasible solution. To combat this, Majumdar et al. (ibid) use LQR controllers and their associated Lyapunov functions for the linearization of the dynamics as good initial seed solutions [31]. In contrast, our approach simply assumes a demonstrator in the form of a MPC controller that can be used to resolve the bilinearity. Furthermore, our approach does not encounter the local saddle point problem.

The use of the learning framework with a demonstrator distinguishes the approach in this paper from recently developed ideas based on formal synthesis [56, 29, 46, 36, 37, 22, 50, 43, 44, 15]. These techniques focus on a given dynamical system and a specification of the correctness in temporal logic to solve the problem of controller design to ensure that the resulting trajectories of the closed loop satisfy the temporal specifications. Majority of the approaches are based on discretization of the state-space into cells to compute a discrete abstraction of the overall system [56, 29, 46, 36, 37, 22]. A smaller set of approaches synthesize Lyapunov-like functions by solving nonlinear constraints either through branch-and-bound techniques or by sum-of-square (SOS) relaxation techniques [50, 43, 44].

In this paper, we use the Lyapunov function approach to synthesizing controllers. An alternative is to use occupation measures [42, 40, 26, 32]. These methods formulate an infinite dimensional problem to maximize the region of attraction and obtain a corresponding control law. This is relaxed to a sequence of finite dimensional SDPs [25]. Note however that the approach computes an over approximation of the finite time backward reachable set from the target and a corresponding control. Our framework here instead seeks an under approximation that yields a guaranteed controller.

VI. Conclusion And Future Work

We have proposed an algorithmic learning framework for synthesizing CLFs using a demonstrator and demonstrated our approach on challenging numerical examples with 4-8 state variables. As future work, we are considering many directions including the extensions to noisy/erroneous demonstrations, using output feedback (rather than full state feedback) synthesis and allowing disturbances in our framework. We are also working on integrating our control framework with RRT-based path planning and implementing it on board robotic vehicles.

Acknowledgments

This work was funded in part by NSF under award numbers SHF 1527075 and CPS 1646556. All opinions expressed are those of the authors and not necessarily of the NSF.

References

[1] David P Anderson. nbot balancing robot, 2002.

[2] 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.

[3] Zvi Artstein. Stabilization with relaxed controls. *Nonlinear Analysis: Theory, Methods & Applications*, 7(11): 1163–1173, 1983.

[4] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

[5] Robert R Burridge, Alfred A Rizzi, and Daniel E Koditschek. Sequential composition of dynamically dexterous robot behaviors. *The International Journal of Robotics Research*, 18(6):534–555, 1999.

[6] Ronald Ping Man Chan, Karl A Stol, and C Roger Halkyard. Review of modelling and control of two-wheeled robots. *Annual Reviews in Control*, 37(1):89–103, 2013.

[7] J Willard Curtis. Clf-based nonlinear control with polytopic input constraints. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, volume 3, pages 2228–2233. IEEE, 2003.

[8] L El Ghaoui and V Balakrishnan. Synthesis of fixed-structure controllers via numerical optimization. In *Decision and Control, 1994., Proceedings of the 33rd IEEE Conference on*, volume 3, pages 2678–2683. IEEE, 1994.

[9] Bruce A Francis and Manfredi Maggiore. Models of mobile robots in the plane. In *Flocking and Rendezvous in Distributed Robotics*, pages 7–23. Springer, 2016.

[10] Sicun Gao, Soonho Kong, and Edmund M Clarke. dreal: An smt solver for nonlinear theories over the reals. In *International Conference on Automated Deduction*, pages 208–214. Springer, 2013.

[11] Sally A Goldman and Michael J Kearns. On the complexity of teaching. *Journal of Computer and System Sciences*, 50(1):20–31, 1995.

[12] J. W. Helton and O. Merino. Coordinate optimization for bi-convex matrix inequalities. In *Proc. IEEE CDC*, volume 4, pages 3609–3613 vol.4, Dec 1997.

[13] Didier Henrion, J Lofberg, Michal Kocvara, and Michael Stingl. Solving polynomial static output feedback problems with penbmi. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 7581–7586. IEEE, 2005.

[14] Didier Henrion, Jean-Bernard Lasserre, and Johan Löfberg. Gloptipoly 3: moments, optimization and semidefinite programming. *Optimization Methods & Software*, 24(4-5):761–779, 2009.

[15] Zhenqi Huang, Yu Wang, Sayan Mitra, Geir E Dullerud, and Swarat Chaudhuri. Controller synthesis with inductive proofs for piecewise linear systems: An smt-based algorithm. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 7434–7439. IEEE, 2015.

[16] Ali Jadbabaie and John Hauser. Control of a thrust-vectored flying wing: a receding horizon-lpv approach. *International Journal of Robust and Nonlinear Control*, 12(9):869–896, 2002.

[17] Mrdjan Jankovic, Dan Fontaine, and Petar V KokotoviC. Tora example: cascade-and passivity-based control designs. *IEEE Transactions on Control Systems Technology*, 4(3):292–297, 1996.

[18] Susmit Jha and Sanjit A Seshia. A theory of formal synthesis via inductive learning. *arXiv preprint arXiv:1505.03953*, 2015.

[19] Gregory Kahn, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Plato: Policy learning using adaptive trajectory optimization. *arXiv preprint arXiv:1603.00622*, 2016.

[20] James Kapinski, Jyotirmoy V Deshmukh, Sriram Sankaranarayanan, and Nikos Arechiga. Simulation-guided lyapunov analysis for hybrid dynamical systems. In *Proceedings of the 17th international conference on Hybrid systems: computation and control*, pages 133–142. ACM, 2014.

[21] Seyed Mohammad Khansari-Zadeh and Oussama Khatib. Learning potential functions from human demonstrations with encapsulated dynamic and compliant behaviors. *Autonomous Robots*, 41(1):45–69, 2017.

[22] Marius Kloetzer and Calin Belta. A fully automated framework for control of linear systems from temporal logic specifications. *Automatic Control, IEEE Transactions on*, 53(1):287–297, 2008.

[23] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[24] Maria Landry, Sue Ann Campbell, Kirsten Morris, and Cesar O Aguilar. Dynamics of an inverted pendulum with delayed feedback control. *SIAM Journal on Applied Dynamical Systems*, 4(2):333–351, 2005.

[25] Jean B Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001.

[26] Jean B Lasserre, Didier Henrion, Christophe Prieur, and Emmanuel Trélat. Nonlinear optimal control via occupation measures and lmi-relaxations. *SIAM Journal on Control and Optimization*, 47(4):1643–1666, 2008.

[27] Steven M LaValle and James J Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. 2000.

[28] Daniel Liberzon. *Switching in systems and control*. Springer Science & Business Media, 2012.

[29] Jun Liu, Necmiye Ozay, Ufuk Topcu, and Richard M Murray. Synthesis of reactive switching protocols from temporal logic specifications. *Automatic Control, IEEE Transactions on*, 58(7):1771–1785, 2013.

[30] Ismael Lopez and Colin R McInnes. Autonomous rendezvous using artificial potential function guidance. *Journal of Guidance, Control, and Dynamics*, 18(2):237–241, 1995.

[31] Anirudha Majumdar, Amir Ali Ahmadi, and Russ Tedrake. Control design along trajectories with sums of squares programming. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4054–4061. IEEE, 2013.

[32] Anirudha Majumdar, Ram Vasudevan, Mark M Tobenkin, and Russ Tedrake. Convex optimization of nonlinear feedback controllers via occupation measures. *The International Journal of Robotics Research*, page 0278364914528059, 2014.

[33] Michael Malisoff and Eduardo D Sontag. Universal formulas for feedback stabilization with respect to minkowski balls. *Systems & Control Letters*, 40(4):247–260, 2000.

[34] Igor Mordatch and Emanuel Todorov. Combining the benefits of function approximation and trajectory optimization. In *Robotics: Science and Systems (RSS)*, 2014.

[35] APS Mosek. The mosek optimization software. *Online at http://www. mosek. com*, 54:2–1, 2010.

[36] Sebti Mouelhi, Antoine Girard, and Gregor Gössler. Cosyma: a tool for controller synthesis using multi-scale abstractions. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 83–88. ACM, 2013.

[37] Necmiye Ozay, Jun Liu, Priyanka Prabhakar, and Richard M Murray. Computing augmented finite transition systems to synthesize switching protocols for polynomial switched systems. In *American Control Conference (ACC), 2013*, pages 6237–6244. IEEE, 2013.

[38] Matthew M Peet and Pierre-Alexander Bliman. Polynomial lyapunov functions for exponential stability of nonlinear systems on bounded regions. *IFAC Proceedings Volumes*, 41(2):1111–1116, 2008.

[39] Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *International Workshop on Hybrid Systems: Computation and Control*, pages 477–492. Springer, 2004.

[40] Stephen Prajna, Pablo A Parrilo, and Anders Rantzer. Nonlinear control synthesis by convex optimization. *IEEE Transactions on Automatic Control*, 49(2):310–314, 2004.

[41] James A Primbs, Vesna Nevistić, and John C Doyle. Nonlinear optimal control: A control lyapunov function and receding horizon perspective. *Asian Journal of Control*, 1(1):14–24, 1999.

[42] Anders Rantzer. A dual to lyapunov's stability theorem. *Systems & Control Letters*, 42(3):161–168, 2001.

[43] Hadi Ravanbakhsh and Sriram Sankaranarayanan. Counter-example guided synthesis of control lyapunov functions for switched systems. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 4232–4239. IEEE, 2015.

[44] Hadi Ravanbakhsh and Sriram Sankaranarayanan. Robust controller synthesis of switched systems using counterexample guided framework. In *Embedded Software (EMSOFT), 2016 International Conference on*, pages 1–10. IEEE, 2016.

[45] Hadi Ravanbakhsh and Sriram Sankaranarayanan. Learning lyapunov (potential) functions from counterexamples and demonstrations. *arXiv preprint arXiv:1705.09619*, 2017.

[46] Matthias Rungger and Majid Zamani. Scots: A tool for the synthesis of symbolic controllers. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pages 99–104. ACM, 2016.

[47] Armando Solar-Lezama. *Program synthesis by sketching*. ProQuest, 2008.

[48] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. *ACM SIGOPS Operating Systems Review*, 40(5):404–415, 2006.

[49] Eduardo D Sontag. A 'universal' construction of artstein's theorem on nonlinear stabilization. *Systems & control letters*, 13(2):117–123, 1989.

[50] Ankur Taly, Sumit Gulwani, and Ashish Tiwari. Synthesizing switching logic using constraint solving. *International journal on software tools for technology transfer*, 13(6):519–535, 2011.

[51] Weehong Tan and Andrew Packard. Searching for control lyapunov functions using sums of squares programming. *sibi*, 1:1, 2004.

[52] S.P Tarasov, L.G Kachiyan, and I.I Erlikh. The method of inscribed ellipsoids. *Doklady Akademy Nauk, SSSR*, 298(5):1081–1085, 1988.

[53] Russ Tedrake, Ian R Manchester, Mark Tobenkin, and John W Roberts. Lqr-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 2010.

[54] Ufuk Topcu, Andrew Packard, Peter Seiler, and Timothy Wheeler. Stability region analysis using simulations and sum-of-squares programming. In *Proceedings of the American control conference*, pages 6009–6014, 2007.

[55] Lieven Vandenberghe, Stephen Boyd, and Shao-Po Wu. Determinant maximization with linear matrix inequality constraints. *SIAM journal on matrix analysis and applications*, 19(2):499–533, 1998.

[56] Tichakorn Wongpiromsarn, Ufuk Topcu, Necmiye Ozay, Huan Xu, and Richard M Murray. Tulip: a software toolbox for receding horizon temporal logic planning. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pages 313–314. ACM, 2011.

[57] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search. *arXiv preprint arXiv:1509.06791*, 2015.

[58] Mingyuan Zhong, Mikala Johnson, Yuval Tassa, Tom Erez, and Emanuel Todorov. Value function approximation and model predictive control. In *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 100–107. IEEE, 2013.