2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

Learning Robust and Correct Controllers from Signal Temporal Logic Specifications Using BarrierNet

Wenliang Liu[1], Wei Xiao[2], and Calin Belta[1]

Abstract— **We consider the problem of learning a neural network controller for a system required to satisfy a Signal Temporal Logic (STL) specification. We exploit STL quantitative semantics to define a notion of robust satisfaction. Guaranteeing the correctness of a neural network controller is a difficult problem that received a lot of attention recently. We provide a general procedure to construct a set of trainable High Order Control Barrier Functions (HOCBFs) enforcing the satisfaction of formulas in a fragment of STL. We use the BarrierNet, implemented by a differentiable Quadratic Program (dQP) with HOCBF constraints, as the last layer of the neural network controller, to guarantee the satisfaction of the STL formulas. We train the HOCBFs together with other neural network parameters to further improve the robustness of the controller. Simulation results demonstrate that our approach ensures satisfaction and outperforms existing algorithms.**

I. INTRODUCTION

We consider the problem of controlling a system from a specification given as a Signal Temporal Logic (STL) formula [1]. It was shown that this can be formulated as an optimization problem with the degree of satisfaction (robustness) as objective or as a constraint, which can be solved using Mixed Integer Programming (MIP) [2], [3] or gradient-based optimization [4]. Such methods, however, are computationally expensive, and difficult to use online.

Reinforcement Learning (RL)-based techniques can perform most of the computation offline, hence enabling real-time control. Model-based RL using neural network was applied to control synthesis problems under STL tasks in [5], [6], where the robustness was used as an objective (reward) function to learn a robust controller. However, these works cannot guarantee the correctness of the learned policy. Violation has two main causes. First, while training a neural network, the system can get stuck at a local optimum, which is likely to happen when the STL specification and the system dynamics are complex. Second, even if the neural network converges to a policy that satisfies the STL specification during training, when given unseen initial conditions or environments in testing, the policy can still fail. The work in [6] uses falsification, while [5] uses Control Barrier Functions (CBF) [7] to mitigate the second problem, but none of them can guarantee satisfaction. Q-learning is also considered for STL control synthesis in [8]. This provides no guarantee of satisfaction either. The authors of [9] use constrained Markov Decision Process (cMDP) to provide a lower bound on the probability of satisfying an STL specification.

In this paper, we use model-based RL and assume that the model (system dynamics) is known. We propose an algorithm to learn a control policy that is guaranteed to satisfy the given STL specification during both training and deployment by using CBFs. Such functions have been employed to enforce the satisfaction of STL specifications [10], [11]. These methods require manual design of the CBFs corresponding to the STL and the parameters in the constraints. A bad design may result in increased conservativeness or even infeasibility. Recently, we proposed BarrierNet [12], implemented as a differentiable QP with CBF constraints, as the last layer of a neural network controller to guarantee safety. In this method, the parameters in the CBF constraints can be obtained through training, which results in significant decrease in conservativeness.

Here we combine BarrierNet [12] with time-varying CBFs for STL tasks [10] to train a neural network controller that guarantees the satisfaction of formulas in a fragment of STL that contains no nested temporal operators and the "*until*" operator. We extend [10] to High Order CBFs (HOCBFs) [13] and provide a general, algorithmic procedure to generate these functions. Further, unlike the fixed CBFs in [10], our HOCBFs contain parameters that can be trained together with the neural network controller using BarrierNet. As a result, our approach avoids the complicated manual design in [10] and reduces the conservativeness after training. Our results show that the learned policy achieves a higher robustness than directly applying CBFs as in [10]. Unlike [12], where the policy is trained on a dataset using supervised learning, we apply model-based RL to train the policy as in [5]. Therefore, no dataset is needed during training. The trained controller can be implemented in real-time and generalized to random initial conditions, while retaining correctness.

II. PRELIMINARIES

Consider a nonlinear control-affine system:

$$\dot{\mathbf{x}} = f(\mathbf{x}) + g(\mathbf{x})\mathbf{u}, \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the system state, $\mathbf{u} \in \mathcal{U} \in \mathbb{R}^q$ is the control, $f : \mathbb{R}^n \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}^{n\times q}$ are locally Lipschitz continuous functions. We assume $\mathcal{U}$ is a box constraint, i.e., $\mathbf{u}_{min} \leq \mathbf{u} \leq \mathbf{u}_{max}$, where the inequality is interpreted element-wise. The initial condition $\mathbf{x}(0) = \mathbf{x}_0$ is randomly sampled in a set $\mathcal{X}_0 \in \mathbb{R}^n$ with probability density function $P : \mathcal{X}_0 \to \mathbb{R}$. We consider solutions to (1) over a compact

*This work was partially supported by the National Science Foundation under grant IIS-2024606 at Boston University.

[1]Wenliang Liu and Calin Belta are with Department of Mechanical Engineering, Boston University, MA, USA wliu97@bu.edu, cbelta@bu.edu

[2]Wei Xiao is with the Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, MA, USA. weixy@mit.edu

time interval $[0, T]$. Given an initial condition $\mathbf{x}_0 \in \mathcal{X}_0$ and a control signal $\mathbf{u} : [0, T] \to \mathcal{U}$, a signal $\mathbf{x} : [0, T] \to \mathbb{R}^n$ is a solution of (1) if $\mathbf{x}(t)$ is absolutely continuous and satisfies (1) for all $t \in [0, T]$. A partial solution on $[0, t]$ is denoted as $\mathbf{x}_{0:t} : [0, t] \to \mathbb{R}^n$. We define a state-feedback neural network controller with memory as

$$\mathbf{u}(t) = \pi(\mathbf{x}_{0:t}, \boldsymbol{\theta}), \tag{2}$$

where $\boldsymbol{\theta}$ is a set of neural network parameters. Memory can be enabled by using Recurrent Neural Network (RNN) [14].

A. Signal Temporal Logic (STL)

Signal Temporal Logic [1] is interpreted over real-valued signals $\mathbf{x} : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$, e.g., solutions of (1). In this paper, we consider a fragment of STL with the following syntax: $\phi := \top \mid \mu \mid \neg\mu \mid \phi_1 \wedge \phi_2$; $\varphi := F_{[t_a,t_b]}\phi \mid G_{[t_a,t_b]}\phi \mid \varphi_1 \wedge \varphi_2$, where ϕ and φ are STL formulae, ϕ_1 and ϕ_2 are formulae of class ϕ, while φ_1, φ_2 are formulae of class φ; $\top$ is the logical *true*, μ is a predicate in the form of $h(\mathbf{x}) \geq 0$ with $h : \mathbb{R}^n \to \mathbb{R}$, $\neg$ and $\wedge$ are Boolean negation and conjunction; F and G are temporal *eventually* and *always*; $[t_a, t_b]$ is a time interval with $t_a < t_b$. In the rest of this paper, we refer to the STL fragment defined above simply as STL.

We use $(\mathbf{x}, t) \models \varphi$ to denote that signal $\mathbf{x}$ satisfies φ at time t. A formal definition of qualitative semantics of STL can be found in [1]. Informally, $F_{[t_a,t_b]}\phi$ is satisfied if "ϕ becomes *True* at some time in $[t_a, t_b]$" while $G_{[t_a,t_b]}\phi$ is satisfied if "ϕ is *True* at all time in $[t_a, t_b]$". Other Boolean operators are interpreted in the usual way. Compared with the full STL [1], this STL fragment cannot contain the temporal *until* or nested temporal operators like "eventually always" (the reason will be clear in Sec. IV-B). However, it is still capable of expressing a wide range of useful temporal properties in practice, e.g., safety and reachability constraints with concrete time requirements.

STL is also equipped with quantitative semantics, also called robustness, which is a real value that measures how much a signal satisfies φ. Multiple STL robustness measures have been proposed [4], [15]. In this paper, we use the smooth robustness defined in [16], which is differentiable almost everywhere, and easy to embedded in learning-based algorithms. The robustness is sound in the sense that the robustness value is positive if and only if the STL formula is satisfied. We denote the robustness of φ at time t with respect to a signal $\mathbf{x}$ as $\rho(\varphi, \mathbf{x}, t)$. Further, we define the time horizon of an STL formula φ as $hrz(\varphi)$, which is the closest time point in the future that is required to determine the satisfaction and robustness of φ. In this paper, we only consider the solution of system (1) within the time horizon of the given STL formula, i.e., $T = hrz(\varphi)$.

B. Time-Varying High Order Control Barrier Function

Informally, the relative degree of a (sufficiently many times) differentiable time-varying function $b : \mathbb{R}^n \times [0, T] \to \mathbb{R}$ defined over the state of system (1) is the number of times it needs to be differentiated along its dynamics until all elements in the control $\mathbf{u}$ show up. Consider a constraint $b(\mathbf{x}, t) \geq 0$ where $b : \mathbb{R}^n \times [0, T] \to \mathbb{R}$ is a differentiable function with relative degree m. Let $\psi_0(\mathbf{x}, t) := b(\mathbf{x}, t)$. We define a sequence of functions $\psi_i : \mathbb{R}^n \times [0, T] \to \mathbb{R}$, $i = 1, \ldots, m$ as follows:

$$\psi_i(\mathbf{x}, t) := \dot{\psi}_{i-1}(\mathbf{x}, t) + \alpha_i\big(\psi_{i-1}(\mathbf{x}, t)\big), \tag{3}$$

where α_i, $i = 1, \ldots, m$ is a $(m-i)^{th}$ order differentiable class $\mathcal{K}$ function [13]. Let the super-level set of $\psi_i(\mathbf{x}, t)$ be:

$$\mathcal{C}_i(t) = \{\mathbf{x} \in \mathbb{R}^n | \psi_i(\mathbf{x}, t) \geq 0\}. \tag{4}$$

Definition 1. (HOCBF [13]) Let $\psi_1(\mathbf{x}, t), \ldots, \psi_m(\mathbf{x}, t)$ be defined by (3) and $\mathcal{C}_1(t), \ldots, C_m(t)$ be defined by (4). A differentiable function $b(\mathbf{x}, t)$ is a High Order Control Barrier Function (HOCBF) with relative degree m with respect to system (1) if there exist differentiable class $\mathcal{K}$ functions α_i, $i = 1, \ldots, m$, such that

$$\begin{aligned} \sup_{\mathbf{u} \in \mathcal{U}} \Big[L_f^m b(\mathbf{x}, t) + L_g L_f^{m-1} b(\mathbf{x}, t)\mathbf{u} + \frac{\partial^m b(\mathbf{x}, t)}{\partial t^m} \\ + O(b(\mathbf{x}, t)) + \alpha_m(\psi_{m-1}(\mathbf{x}, t))\Big] \geq 0, \end{aligned} \tag{5}$$

for all $(\mathbf{x}, t) \in \mathcal{C}_1(t) \cap \ldots \cap \mathcal{C}_m(t) \times [0, T]$. In (5), L_f^m (L_g) denotes Lie derivatives along f (g) m (one) times, and $O(b(\mathbf{x}, t))$ denotes the remaining Lie derivatives along f and partial derivatives with respect to t with degree less than m.

Definition 2. (Forward invariant) A set $\mathcal{C}(t) \subset \mathbb{R}^n$ that depends on time is forward invariant for system (1) given a control law $\mathbf{u}$ if for any $\mathbf{x}(0) \in \mathcal{C}(0)$, the solution of system (1) satisfies $\mathbf{x}(t) \in \mathcal{C}(t)$, $\forall t \in [0, T]$.

Theorem 1. [13] Given an HOCBF $b(\mathbf{x}, t)$ with a sequence of sets $\mathcal{C}_1(t), \ldots, C_m(t)$ as defined in (4), if $\mathbf{x}(0) \in \mathcal{C}_1(0) \cap \mathcal{C}_2(0) \cap \ldots \cap \mathcal{C}_m(0)$, then any Lipschitz continuous controller $\mathbf{u}(t)$ that satisfies (5) $\forall t \in [0, T]$ renders $\mathcal{C}_1(t) \cap \mathcal{C}_2(t) \cap \ldots \cap \mathcal{C}_m(t)$ forward invariant for system (1).

III. Problem Formulation and Approach

Let $J(\mathbf{u})$ be a cost function over control signals $\mathbf{u} : [0, T] \to \mathcal{U}$. The problem we consider in this paper is:

Problem 1. Given a system with known dynamics (1), an STL specification φ, and an initial state $\mathbf{x}_0$ sampled from the distribution $P : \mathcal{X}_0 \to \mathbb{R}$, find the optimal control $\mathbf{u}^*(t)$ that maximizes the STL robustness and minimize the cost $J(\mathbf{u})$ while guaranteeing the satisfaction of φ:

$$\begin{aligned} \mathbf{u}^*(t) = &\arg\max_{\mathbf{u}(t)} \rho(\varphi, \mathbf{x}, 0) - J(\mathbf{u}) \\ \text{s.t.} \quad &\dot{\mathbf{x}} = f(\mathbf{x}) + g(\mathbf{x})\mathbf{u}(t), \\ &\mathbf{u}_{min} \leq \mathbf{u}(t) \leq \mathbf{u}_{max}, \\ &(\mathbf{x}, 0) \models \varphi. \end{aligned} \tag{6}$$

To be robust against disturbances, a feedback controller is desired. One can obtain such a feedback controller by solving (6) at each discrete time step in a model predictive control manner as in [2], [3]. However, doing so can be time-consuming and prevent real-time control. Training a neural network controller that maximizes the expected objective in (6) over initial state distribution P can move the online computation to offline. After training, the controller can be computed in real-time and can be generalized to random initial conditions under the distribution P [5]. Moreover, in general, an STL specification is history-dependent [17], i.e., to satisfy it, the desired control $\mathbf{u}(t)$ should depend on not only the current state $\mathbf{x}(t)$ but also history states $\mathbf{x}_{0:t}$. Hence, a controller with memory is needed.

In this paper, we train a neural network controller with memory (2) to solve Problem 1. We first construct a set of trainable time-varying HOCBFs from the STL formula φ. Then we embedded these HOCBFs into the neural network controller using a modified version of the BarrierNet from [12] to guarantee the satisfaction of φ. We train the neural network controller together with the HOCBFs to further increase the STL robustness.

IV. Solution

A. Trainable HOCBF and BarrierNet

Suppose that we have a set of time-varying HOCBFs $b_j(\mathbf{x}, t, \boldsymbol{\theta}_b, \mathbf{x}_0)$ that depend on the initial condition $\mathbf{x}_0$ and contain trainable parameters $\boldsymbol{\theta}_b$, $j = 1, \dots, M$. The reason they depend on $\mathbf{x}_0$ will be clear in Section IV-C. To avoid over-conservativeness, we make the class $\mathcal{K}$ functions also trainable. Rewrite (3) for a HOCBF b_j into:

$$\psi_{i,j}(\mathbf{x}, t) := \dot{\psi}_{i-1,j}(\mathbf{x}, t) + p_{i,j}(\mathbf{x}_0, \boldsymbol{\theta}_p)\alpha_{i,j}\big(\psi_{i-1,j}(\mathbf{x}, t)\big), \tag{7}$$

where $\alpha_{i,j}$ are given class $\mathcal{K}$ functions, $p_{i,j}(\mathbf{x}_0, \boldsymbol{\theta}_p) > 0$, $i = 1, \dots, m_j$, $j = 1, \dots, M$, m_j is the relative degree of HOCBF b_j. $p_{i,j}(\mathbf{x}_0, \boldsymbol{\theta}_p)$ also depends on initial condition and contains trainable parameters $\boldsymbol{\theta}_p$. The reason it depends on $\mathbf{x}_0$ will be clear in Section IV-C as well.

BarrierNet [12] is a neural network layer implemented by a differentiable Quadratic Program (dQP) with HOCBF constraints. We add it as the last layer of a neural network controller (2), i.e., $\pi(\mathbf{x}_{0:t}, \boldsymbol{\theta}) = \mathbf{u}^*(t)$ with $\mathbf{u}^*(t)$ given by:

$$\begin{aligned}
\mathbf{u}^*(t) = &\arg\min_{\mathbf{u}(t)} \frac{1}{2}\mathbf{u}(t)^\top \mathbf{Q}(\mathbf{x}_{0:t}, \boldsymbol{\theta}_q)\mathbf{u}(t) + \mathbf{F}^\top(\mathbf{x}_{0:t}, \boldsymbol{\theta}_f)\mathbf{u}(t) \\
\text{s.t.} \quad & L_f^m b_j(\mathbf{x}, t, \boldsymbol{\theta}_b, \mathbf{x}_0) + L_g L_f^{m-1} b_j(\mathbf{x}, t, \boldsymbol{\theta}_b, \mathbf{x}_0)\mathbf{u}(t) \\
& + \frac{\partial^m b_j(\mathbf{x}, t, \boldsymbol{\theta}_b, \mathbf{x}_0)}{\partial t^m} + O(b_j(\mathbf{x}, t, \boldsymbol{\theta}_b, \mathbf{x}_0)) \\
& + p_{m,j}(\mathbf{x}_0, \boldsymbol{\theta}_p)\alpha_m(\psi_{m-1,j}(\mathbf{x}, t, \boldsymbol{\theta}_b, \mathbf{x}_0)) \geq 0, \\
& t = k\Delta t, \ k = 0, 1, 2, \dots, \ j = 1, \dots, M,
\end{aligned} \tag{8}$$

where $\mathbf{Q}(\mathbf{x}_{0:t}, \boldsymbol{\theta}_q) \in \mathbb{R}^{q\times q}$, $\mathbf{F}(\mathbf{x}_{0:t}, \boldsymbol{\theta}_f) \in \mathbb{R}^q$, $p_{i,j}(\mathbf{x}_0, \theta_p)$, $i = 1, \dots, m$ and $b_j(\mathbf{x}, t, \boldsymbol{\theta}_b, \mathbf{x}_0)$ are all given by previous neural network layers with trainable parameters $(\boldsymbol{\theta}_q, \boldsymbol{\theta}_f, \boldsymbol{\theta}_p, \boldsymbol{\theta}_b) := \boldsymbol{\theta}$, $\mathbf{Q}$ is positive definite. $\boldsymbol{Q}^{-1}\mathbf{F}$ can be interpreted as a reference control. Although in (8), $\mathbf{Q}$, $\mathbf{F}$, p_i are given by previous layers, they can also be directly trainable parameters. The dQP (8) is solved at each time point $k\Delta t$, $k = 0, 1, \dots$ until reaching the time horizon T, and the solution $\mathbf{u}^*(t)$ is applied to the system as a constant for the time period $[k\Delta t, k\Delta t + \Delta t)$. Since (8) is differentiable, the gradient of $\mathbf{u}^*(t)$ with respect to $\boldsymbol{\theta}$ can be calculated using the technique in [18], then $\boldsymbol{\theta}$ can be trained using any methods for training neural networks. Different from the original BarrierNet [12], in (8) we also make the HOCBF b itself trainable besides $\mathbf{Q}$, $\mathbf{F}$ and p_i, as it will be detailed in the next subsection. BarrierNet is able to guarantee the satisfaction of all HOCBF constraints. Meanwhile, through training the controller can also optimize a given objective function.

B. HOCBFs for STL specifications

The authors of [10] proposed the idea of using time-varying CBFs to ensure the satisfaction of STL specifications. However, in [10] only relative degree 1 CBFs are considered and the generation of CBFs is described by examples without explicitly showing the construction rules. In this paper, we extend this method to HOCBFs and provide a general and algorithmic procedure to construct them. Further, we make these HOCBFs trainable so that the manual design is avoided, and the performance of the controller including these HOCBFs can be further improved through training.

Consider an STL formula φ. Since for all predicates with negations $\neg\mu$ we can replace the predicate function with $-h(\mathbf{x})$ and remove the negation, we assume that the formula φ is negation-free without loss of generality. We make the following assumption on the STL formula and the system:

Assumption 1. $\forall \mathbf{x}(0) \in \mathcal{X}_0, \exists \mathbf{u}(t) \in \mathcal{U}$ such that $(\mathbf{x}, 0) \models \varphi$ where $\mathbf{x}$ is the solution of system (1).

Assumption 1 is not restrictive in practice since if it is not true, for some $\mathbf{x}_0$ there is no solution for Problem 1.

Categories of Predicates. Suppose that there are M predicates in φ and they are given by $\mu_j : \ h_j(\mathbf{x}) \geq 0$, $j = 1, \dots, M$. We divide all predicates into three categories:

- Category I: predicates that are satisfied at $t = 0$ and the starting time of the temporal operator wrapping it is 0, e.g., μ_1 in $G_{[0,5]}\mu_1$ and $G_{[0,5]}\mu_1 \wedge \mu_2$, where $h_1(\mathbf{x}_0) \geq 0$. These predicates usually define safety requirements, such as obstacle avoidance in robotic applications.
- Category II: All predicates wrapped by $F_{[t_a,t_b]}$ that do not belong to Category I, e.g., μ_1 in $F_{[2,5]}\mu_1$ and μ_2 in $F_{[0,5]}\mu_2 \wedge \mu_3$ where $h_2(\mathbf{x}_0) < 0$.
- Category III: All predicates wrapped by $G_{[t_a,t_b]}$ that do not belong to Category I, e.g., μ_1 in $G_{[2,5]}\mu_1 \wedge \mu_2$. Note that Assumption 1 avoids formulae like $G_{[0,5]}\mu_1$, where $h_1(\mathbf{x}_0) < 0$.

STL Guarantees. To each predicate μ_j, we assign a (time-varying) HOCBF b_j. Since each predicate μ_j belonging to Category I has already been satisfied at $t = 0$, we assign a fixed and time-invariant HOCBF to retain its satisfaction for the required time:

$$b_j(\mathbf{x}) = h_j(\mathbf{x}). \tag{9}$$

For predicates μ_j in Category II and III, we assign a trainable time-varying HOCBF:

$$b_j(\mathbf{x}, t, \boldsymbol{\theta}_b, \mathbf{x}_0) = h_j(\mathbf{x}) + \gamma_j(t, \boldsymbol{\omega}_j(\boldsymbol{\theta}_b, \mathbf{x}_0)), \tag{10}$$

where $\gamma_j(\cdot, \boldsymbol{\omega}_j) : [0, T] \to \mathbb{R}$ is a function parameterized by $\boldsymbol{\omega}_j$, $\boldsymbol{\omega}_j$ is given by a neural network with input $\mathbf{x}_0$ and parameters $\boldsymbol{\theta}_b$. Details about this neural network will be discussed in Section IV-C. In the rest of this subsection, we will omit $\boldsymbol{\theta}_b$ and $\mathbf{x}_0$ for notation simplicity and just consider $\boldsymbol{\omega}_j$ as a vector. By properly choosing $\gamma_j(t, \boldsymbol{\omega}_j)$, the satisfaction of $b_j(\mathbf{x}, t) \geq 0$, $\forall t \in [0, T]$ can ensure the satisfaction of the predicate μ_j during the required time slots. Next, we discuss the selection of $\gamma_j(t, \boldsymbol{\omega}_j)$.

For simplicity, we omit the subscript j when it is clear from the context. For a predicate μ in Category II that is wrapped with $F_{[t_a,t_b]}$, we choose γ to be a linear function:

$$\gamma(t, \boldsymbol{\omega}) = \omega_1 + \omega_2 t, \tag{11}$$

where $\boldsymbol{\omega} = (\omega_1, \omega_2)$, $\omega_1 > 0$, $\omega_2 < 0$. Note that other forms of functions are also possible. To make sure the HOCBF $b(\mathbf{x}, t) = h(\mathbf{x}) + \gamma(t, \boldsymbol{\omega})$ guarantees the satisfaction of μ, we add 3 constraints on γ:

$$\gamma(0, \boldsymbol{\omega}) \;\; > -h(\mathbf{x}_0), \tag{12a}$$

$$\gamma(t_b, \boldsymbol{\omega}) \leq 0, \tag{12b}$$

$$\gamma(t_a, \boldsymbol{\omega}) > -\sup_{\mathbf{x} \in \mathbb{R}^n} h(\mathbf{x}). \tag{12c}$$

Constraint (12a) ensures the HOCBF is positive at the initial time, i.e., $b(\mathbf{x}_0, 0) > 0$. Constraint (12b) ensures that $h(\mathbf{x}) \geq b(\mathbf{x}, t) \geq 0$ before time t_b. Given (12a) and (12b) the forward invariance of the superlevel set of $b(\mathbf{x}, t)$ enforces the satisfaction of $F_{[t_a,t_b]}\mu$. The third constraint (12c) ensures that the superlevel set of $b(\mathbf{x}, t)$ is nonempty when $t < t_a$. As it will be dicussed later, we delete the HOCBF once $h(\mathbf{x}) > 0$ when $t \geq t_a$, so we do not consider whether the superlevel set of $b(\mathbf{x}, t)$ is empty after t_a.

For a predicate μ in Category III that is wrapped with $G_{[t_a,t_b]}$, let γ be defined as:

$$\gamma(t, \boldsymbol{\omega}) = \omega_1 e^{-\omega_2 t} - c, \tag{13}$$

where $\boldsymbol{\omega} = (\omega_1, \omega_2)$, $\omega_1 > 0$, $\omega_2 > 0$. $c > 0$ is a small constant. Again, other forms of functions are possible. Similar to (12), we have two constraints on γ:

$$\gamma(0, \boldsymbol{\omega}) \;\; > -h(\mathbf{x}_0), \tag{14a}$$

$$\gamma(t_a, \boldsymbol{\omega}) \leq 0. \tag{14b}$$

The difference is that (14b) ensures $h(\mathbf{x}) \geq b(\mathbf{x}, t) \geq 0$ before time t_a so that $G_{[t_a,t_b]}\mu$ is enforced to be satisfied. When $c > 0$ is small enough, the superlevel set of $b(\mathbf{x}, t)$ is always nonempty under Assumption 1. We choose the exponential function (13) for *always* instead of a linear function because it satisfies:

$$0 \leq -\gamma(t, \boldsymbol{\omega}) < c, \; \forall t \in [t_a, t_b].$$

As a result, $b(\mathbf{x}, t) \geq 0$, i.e., $h(\mathbf{x}) \geq -\gamma(t, \boldsymbol{\omega})$, is not over-conservative for $t \in [t_a, t_b]$ when $c > 0$ is small enough. As it will be detailed below, the HOCBF is deleted when $t > t_b$, which further mitigates over-conservativeness.

We can construct an HOCBF b_j for each predicate μ_j in φ using (9) or (10). However, it is possible that the corresponding constraints $b(\mathbf{x}, t) \geq 0$ are conflicting with each other during some time periods. By making an additional assumption below, we addressed this issue in detail in [19].

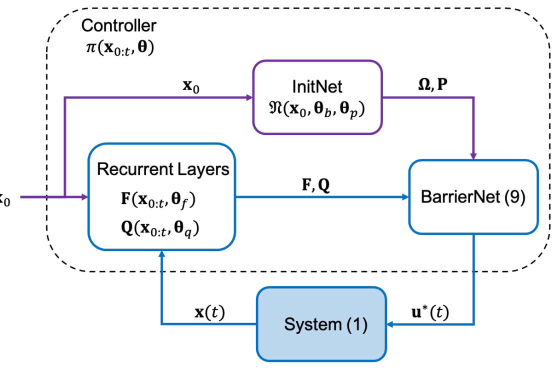

Controller
$\pi(\mathbf{x}_{0:t}, \boldsymbol{\theta})$
$\mathbf{x}_0$
InitNet
$\mathfrak{N}(\mathbf{x}_0, \boldsymbol{\theta}_b, \boldsymbol{\theta}_p)$
$\boldsymbol{\Omega}, \mathbf{P}$
$\mathbf{x}_0$
Recurrent Layers
$\mathbf{F}(\mathbf{x}_{0:t}, \boldsymbol{\theta}_f)$
$\mathbf{Q}(\mathbf{x}_{0:t}, \boldsymbol{\theta}_q)$
$\mathbf{F}, \mathbf{Q}$
BarrierNet (9)
$\mathbf{x}(t)$
System (1)
$\mathbf{u}^*(t)$

Fig. 1: Overall structure of the controller. Purple parts are only executed at $t = 0$, while blue parts are executed repeatedly. The dashed box indicates the controller $\pi(\mathbf{x}_{0:t}, \boldsymbol{\theta})$.

Assumption 2. Let all predicate functions in Category II and III be in the form of:

$$h(\mathbf{x}) = \pm\big(R - \|l(\mathbf{x}) - \mathbf{o}\|_2\big), \tag{15}$$

where $l : \mathbb{R}^n \to \mathbb{R}^o$ is a differentiable function shared by all predicates mapping state $\mathbf{x}$ to a vector that we care about, e.g., the location of a robot, and $R \in \mathbb{R}_+$, $\mathbf{o} \in \mathbb{R}^o$ are the radius and center of a circular region.

Theorem 2. Assume we have a STL formula φ satisfying Assumption 2, a system (1) satisfying Assumptions 1, a set of HOCBFs constructed by (9) and (10) that satisfy all constraints (12) and (14), and a sequence of functions ψ_i for each HOCBF as in (3), where $\psi_i(\mathbf{x}_0, 0) \geq 0$, $i = 1, \ldots, m$. Then a control law $\mathbf{u}(t)$ that satisfies (5) for all HOCBFs is guaranteed to satisfy specification φ.

The proof can be found in [19].

C. Learning Robust Controllers

Theorem 2 ensures the satisfaction of the STL specification when all HOCBFs constraints are satisfied. Then we can use BarrierNet (8) to obtain a controller that satisfies all HOCBFs constraints. In this subsection, we first explain why in (8), $b(\mathbf{x}, t, \boldsymbol{\theta}_b, \mathbf{x}_0)$ and $p_{i,j}(\mathbf{x}_0, \boldsymbol{\theta}_p)$ all depend on the initial condition $\mathbf{x}_0$. Then we describe the structure of the entire neural network controller $\pi(\mathbf{x}_{0:t}, \boldsymbol{\theta})$. Finally, we introduce the training process of the controller.

Parameters Depending on Initial Conditions. Consider a predicate belonging to Category II or III with corresponding HOCBF $b(\mathbf{x}, t) = h(\mathbf{x}) + \gamma(t, \boldsymbol{\omega})$. Since constraints (12a) and (14a) on parameters $\boldsymbol{\omega}$ contain the initial condition $\mathbf{x}_0$, different $\boldsymbol{\omega}$ should be used for different initial condition $\mathbf{x}_0$. Hence, we use a neural network whose input is $\mathbf{x}_0$ to provide $\boldsymbol{\omega}$, denoted as $\boldsymbol{\omega}(\mathbf{x}_0, \boldsymbol{\theta}_b)$. As a result, the HOCBF also depends on $\mathbf{x}_0$ and contains trainable parameters $\boldsymbol{\theta}_b$, denoted as $b(\mathbf{x}, t, \boldsymbol{\theta}_b, \mathbf{x}_0)$.

On the other hand, to use HOCBFs to guarantee set-invariance, we also need to make sure $\psi_i(\mathbf{x}_0, 0) \geq 0$ for all $i = 1, \ldots, m$. Since $b(\mathbf{x}_0, 0) > 0$, we can always find a large enough p_i such that $\psi_i(\mathbf{x}_0, 0) \geq 0$ according to (7). These constraints on p_i also depend on $\mathbf{x}_0$. Hence, we use a neural network with input $\mathbf{x}_0$ and parameters $\boldsymbol{\theta}_p$ to provide p_i, denoted as $p_i(\mathbf{x}_0, \boldsymbol{\theta}_p)$.

Neural Network Controller Structure. In practice, we use one neural network referred to as InitNet to provide all parameters depending on $\mathbf{x}_0$:

$$[\mathbf{\Omega}^\top \mathbf{P}^\top] = \mathfrak{N}(\mathbf{x}_0, \boldsymbol{\theta}_b, \boldsymbol{\theta}_p), \tag{16}$$

where $\mathbf{\Omega} = [\boldsymbol{\omega}_1^\top \dots \boldsymbol{\omega}_N^\top]^\top \in \mathbb{R}^{2N}$, $\mathbf{P}$ is the concatenation of all p_i in (7) for all HOCBFs, $\mathfrak{N}$ is the neural network parameterized by trainable parameters $\boldsymbol{\theta}_b$ and $\boldsymbol{\theta}_p$. We transform the constraints on γ into constraints on $\mathbf{\Omega}$. For constraints in the form of $\omega \in [\underline{\omega}, \overline{\omega}]$ we apply a Sigmoid function on the last layer of $\mathfrak{N}$ while for constraints in the form of $\omega \in [\underline{\omega}, \infty)$ or $\omega \in (-\infty, \overline{\omega}]$ we apply a Softplus function. In this way, $\mathbf{\Omega}$ satisfies all constraints in (12) and (14). Similarly, for p_i, $i = 1, \dots, m-1$, we add constraints $p_i > max\big[-\dot{\psi}_{i-1}(\mathbf{x}, 0)/\alpha_i\big(\psi_{i-1}(\mathbf{x}, 0)\big), 0\big]$ which are also implemented by Softplus functions.

InitNet is only used at time $t = 0$ to provide a set of HOCBFs and the corresponding class $\mathcal{K}$ functions, which are fixed after $t = 0$. Then we use another (recurrent) neural network parameterized by $\boldsymbol{\theta}_q$ and $\boldsymbol{\theta}_f$ to provide $\mathbf{Q}(\mathbf{x}_{0:t}, \boldsymbol{\theta}_q)$ and $\mathbf{F}(\mathbf{x}_{0:t}, \boldsymbol{\theta}_f)$ at each discrete time point. The whole controller $\pi(\mathbf{x}_{0:t}, \boldsymbol{\theta}) = \mathbf{u}^*$ contains $\mathbf{Q}(\mathbf{x}_{0:t}, \boldsymbol{\theta}_q)$, $\mathbf{F}(\mathbf{x}_{0:t}, \boldsymbol{\theta}_f)$, $\mathfrak{N}(\mathbf{x}_0, \boldsymbol{\theta}_b, \boldsymbol{\theta}_p)$ and the dQP (8) with $\boldsymbol{\theta} = (\boldsymbol{\theta}_q, \boldsymbol{\theta}_f, \boldsymbol{\theta}_b, \boldsymbol{\theta}_p)$. The overall structure of the controller is shown in Fig. 1

Training BarrierNet. Similar to [5], we randomly sample V initial conditions $\mathbf{x}_0^v$, $v = 1, \dots, V$. We apply the system dynamics (1) with the controller π until reaching the time horizon T to get V state and control trajectories. We evaluate their STL robustness and cost J, and then use the mean value to approximate the expectation. Formally, we rewrite (6) into:

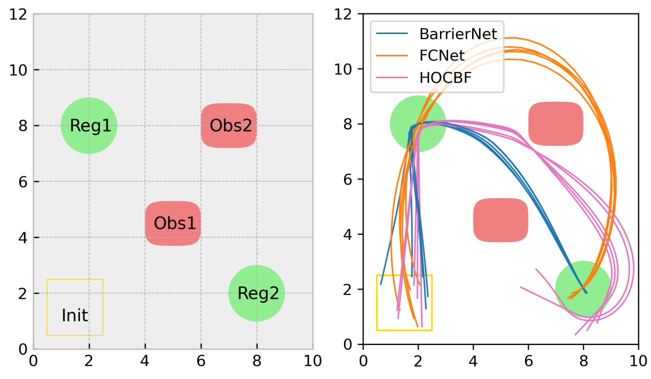
$$\begin{aligned} \boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}} \frac{1}{V} \sum_{v=1}^{V} \big[\rho(\varphi, \mathbf{x}^v, 0) - J(\mathbf{u}^v)\big] \\ \text{s.t. } \dot{\mathbf{x}}^v &= f(\mathbf{x}^v) + g(\mathbf{x}^v)\pi(\mathbf{x}^v_{0:t}, \boldsymbol{\theta}), \ v = 1, \dots, V, \end{aligned} \tag{17}$$

where the superscript v indicates the v^{th} sample. We substitute the constraint (dynamics) into the objective function to make it an unconstrained optimization problem. Since the QP (8) is differentiable with respect to its parameters using the technique in [18], we backpropagate the gradient of the objective funtion in (17) through the QP to all parameters $\boldsymbol{\theta}$. The gradients of the STL robustness are calculated analytically and automatically using an adapted version of STLCG [20] that use the robustness in [16]. Then we update the parameters using the gradient. Note that at each optimization step we randomly resample V initial conditions to have a better exploration of the initial set $\mathcal{X}_0$ and we use the stochastic optimizer Adam to train the parameters.

The following corollary from Theorem 2 states that our network controller is correct:

Corollary 1. Consider an STL formula φ and a system (1) satisfying Assumptions 1 and 2. Then any neural network controller with BarrierNet (8) as the last layer guarantees that the solution of system (1) starting from any initial condition $\mathbf{x}_0 \in \mathcal{X}_0$ satisfies the specification φ.

Proof. This follows immediately from Theorem 2. $\square$



(a) Environment (b) Trajectories

Fig. 2: (a) The 2D environment we consider. (b) Sampled trajectories with random initial conditions using three methods: BarrierNet (developed in this paper), FCNet, and HOCBFs.

An algorithm summarizing our solution is in [19].

V. Simulations

Environment and STL setup. Consider a 2D robot navigation problem. The dynamics of the robot is given as:

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{v}_x \\ \dot{v}_y \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix}, \tag{18}$$

where $\mathbf{x} = [p_x \ p_y \ v_x \ v_y]^\top$, $\mathbf{u} = [a_x \ a_y]^\top$, $[p_x \ p_y]^\top$ is the 2D position, $[v_x \ v_y]^\top$ is the velocity, and $[a_x \ a_y]^\top$ is the acceleration of the robot. We assume the control has no bounds in this case and use the L2 norm for the cost function in (17) with a coefficient of 0.003 to punish large accelerations. Consider the environment shown in Fig. 2a. $\mathbf{x}_0$ is uniformly sampled in the region $Init$ with zero velocity. We discretize the system with a time interval of $0.1s$. The task for the robot is given by an STL formula:

$$\varphi = F_{[0,2]}Reg_1 \wedge F_{[2,5]}Reg_2 \wedge G_{[0,5]}(\neg Obs_1 \wedge \neg Obs_2), \tag{19}$$

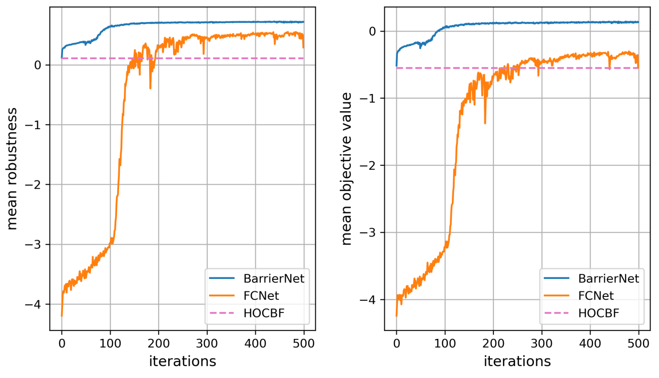
where Reg_i indicates $R_i - \|l(\mathbf{x}) - \mathbf{o}_i\|_2 \geq 0$, $i = 1, 2$, $l(\mathbf{x}) = [p_x \ p_y]^\top$. Obs_i is a superellipse:

$$1 - \sqrt[4]{(\frac{p_x - o_{x,i}}{a_i})^4 + (\frac{p_y - o_{y,i}}{b_i})^4} \geq 0, \tag{20}$$

$i = 1, 2$. Here, Reg_i belongs to Category II and Obs_i belongs to Category I, $i = 1, 2$. In plain English, the STL formula φ requires the robot to eventually visit Reg_1 within $[0, 2]$ and eventually visit Reg_2 within $[2, 5]$, while always avoid obstacles Obs_1 and Obs_2. The time horizon of φ is 5. For all 4 predicates, the corresponding HOCBFs have a relative degree of 2 with respect to system (18). In this example, we fixed $\mathbf{Q}(\mathbf{x}, \boldsymbol{\theta}_q)$ to an identical matrix, so the output of the previous layers at $t > 0$ is just $\mathbf{F}(\mathbf{x}, \boldsymbol{\theta}_f)$, which can be interpreted as a reference control. Since this task does not require back and forth motions, an RNN is not necessary for $\mathbf{F}$. Hence, both $\mathbf{F}$ and InitNet are implemented as neural networks with 3 fully connected layers. For the robustness function, we use the exponential robustness given in [16].

Comparison setup. We construct the HOCBFs and train the controller proposed in this paper. We compare the results with our previous work [5] where a neural network controller without BarrierNet, i.e., a Fully Connected Neural Network (FCNet) is trained for an STL task. It is equivalent to directly use the reference control $\mathbf{F}$. We refer to these two controllers as BarrierNet and FCNet respectively. To make the comparison fair, we assume that the system dynamics are known for [5]. We use the same objective function, optimizer, and the same neural network architectures, i.e., the FCNet has the same structure with $\mathbf{F}(\mathbf{x}, \boldsymbol{\theta}_f)$. Training curves are illustrated in Fig. 3. Meanwhile, we directly apply the approach in [10] (extended to HOCBF) without any learning, i.e., we construct a set of HOCBFs with fixed parameters and solve the QP (8) with $\mathbf{F} = \mathbf{0}$. The parameters are randomly chosen but satisfy all constraints (12) and (14). We refer to this approach as HOCBF. The resulting average values of the objective function and the robustness starting from random initial conditions are shown in Fig.3 with dashed lines. Sampled trajectories using the three approaches with random initial conditions are shown in Fig. 2b.

Analysis and Discussion. In Fig. 3a we can see that when using BarrierNet, the robustness is positive (the STL specification is satisfied) from the beginning of the training. This demonstrates the correctness of Corollary 1. As for FCNet, it takes about 150 iterations to get a positive mean robustness value. The results of directly applying HOCBFs with randomly chosen parameters are similar as using BarrierNet with an untrained neural network, i.e., at the first iteration during training. It also satisfies the specification but is less robust than using BarrierNet after training. As shown in Fig. 2b the robot reaches the center of Reg_2 with both BarrierNet and FCNet after training while only reaches the boundary of Reg_2 when directly using HOCBFs. The robot leaves Reg_2 after the corresponding HOCBF is deleted. The final robustness and objective function values of BarrierNet are both higher than FCNet. Without the guidance of HOCBFs, the FCNet controller only finds a sub-optimal solution which steers the robot further away to avoid the obstacles. Training of the BarrierNet and FCNet takes 14 min and 2 min respectively on our machine with 2.1/4.9 GHz Core i7 CPU and NVIDIA RTX 3050 GPU. Since QP can be solve very efficiently, all three methods can execute fast during testing which enables real-time control.



(a) Robustness (b) Objective

Fig. 3: Learning curves for the methods of BarrierNet and FCNet. Dashed lines show the result of directly using HOCBFs. (a) the mean robustness values. (b) the mean objective function values.

References

[1] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*. Springer, 2004, pp. 152–166.

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

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

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

[5] W. Liu, M. Nishioka, and C. Belta, "Safe model-based control from signal temporal logic specifications using recurrent neural networks," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 12 416–12 422.

[6] K. Leung and M. Pavone, "Semi-supervised trajectory-feedback controller synthesis for signal temporal logic specifications," in *2022 American Control Conference (ACC)*. IEEE, 2022, pp. 178–185.

[7] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3861–3876, 2016.

[8] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "Q-learning for robust satisfaction of signal temporal logic specifications," in *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 2016, pp. 6565–6570.

[9] K. C. Kalagarla, R. Jain, and P. Nuzzo, "Model-free reinforcement learning for optimal control of markov decision processes under signal temporal logic specifications," in *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, 2021, pp. 2252–2257.

[10] L. Lindemann and D. V. Dimarogonas, "Control barrier functions for signal temporal logic tasks," *IEEE control systems letters*, vol. 3, no. 1, pp. 96–101, 2018.

[11] W. Xiao, C. A. Belta, and C. G. Cassandras, "High order control lyapunov-barrier functions for temporal logic specifications," in *2021 American Control Conference (ACC)*. IEEE, 2021, pp. 4886–4891.

[12] W. Xiao, T.-H. Wang, R. Hasani, M. Chahine, A. Amini, X. Li, and D. Rus, "BarrierNet: Differentiable control barrier functions for learning of safe robot control," *IEEE Transactions on Robotics, DOI: 10.1109/TRO.2023.3249564*, 2023.

[13] W. Xiao and C. Belta, "Control barrier functions for systems with high relative degree," in *2019 IEEE 58th conference on decision and control (CDC)*. IEEE, 2019, pp. 474–479.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[15] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2010, pp. 92–106.

[16] W. Liu, K. Leahy, Z. Serlin, and C. Belta, "Robust multi-agent coordination from catl+ specifications," in *2023 American Control Conference (ACC)*. IEEE, 2023, pp. 3529–3534.

[17] W. Liu, N. Mehdipour, and C. Belta, "Recurrent neural network controllers for signal temporal logic specifications subject to safety constraints," *IEEE Control Systems Letters*, vol. 6, pp. 91–96, 2021.

[18] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 136–145.

[19] W. Liu, W. Xiao, and C. Belta, "Learning robust and correct controllers from signal temporal logic specifications using barriernet," *arXiv preprint arXiv:2304.06160*, 2023.

[20] K. Leung, N. Aréchiga, and M. Pavone, "Backpropagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods," *The International Journal of Robotics Research*, p. 02783649221082115, 2020.