# Recurrent Neural Network Controllers for Signal Temporal Logic Specifications Subject to Safety Constraints

Wenliang Liu, *Graduate Student Member, IEEE*,
Noushin Mehdipour, *Graduate Student Member, IEEE*,
and Calin Belta, *Fellow, IEEE*

***Abstract*—We propose a framework based on Recurrent Neural Networks (RNNs) to determine an optimal control strategy for a discrete-time system that is required to satisfy specifications given as Signal Temporal Logic (STL) formulae. RNNs can store information of a system over time, thus, enable us to determine satisfaction of the dynamic temporal requirements specified in STL formulae. Given a STL formula, a dataset of satisfying system executions and corresponding control policies, we can use RNNs to predict a control policy at each time based on the current and previous states of system. We use Control Barrier Functions (CBFs) to guarantee the safety of the predicted control policy. We validate our theoretical formulation and demonstrate its performance in an optimal control problem subject to partially unknown safety constraints through simulations.**

***Index Terms*—Optimal control, neural networks, autonomous systems.**

## I. Introduction

DUE TO their expressivity and similarity to natural languages, temporal logics have been used to formalize specifications for cyber-physical systems. Control policies enforcing the satisfaction of such specifications have been derived [1], [2]. Our focus in this letter is Signal Temporal Logic (STL) [3], which is interpreted over real-valued signals. STL is equipped with quantitative semantics, known as robustness, that measures how strongly a signal satisfies a specification [4]. This allows to map the problem of controlling a system under a STL specification to an optimization problem with robustness as cost function [5], [6]. Optimizing the robustness, whether through a Mixed Integer Programming (MIP) encoding [5] or a gradient-based method [7], [8], [9], [10], [11], can be computationally expensive and might not meet real-time requirements in practice. Moreover, the optimization may converge to local optima, which might not satisfy the STL specification.

To address these limitations, we propose a Recurrent Neural Network (RNN) controller design for a dynamical system with specifications given as STL formulae. The input to the RNN is the current state of the system and the output is the control that is predicted to maximize the STL robustness at that state. The RNN is trained using imitation learning [12], in which the dataset consists of samples (system executions) generated by solving an optimization problem. A shallow RNN requires limited computations, and thus, it can be used for real-time control. Moreover, convergence can be improved by excluding samples with robustness scores less than a specified threshold.

Employing neural networks (NN) in temporal logic control was proposed recently. In [13], the authors used a feedforward NN as a feedback controller to satisfy STL specifications. The feedforward NN predicted the controller at each time only based on the current state of the system. However, in general, the satisfaction of a STL specification is *history-dependent*. For example, if a specification requires an agent to visit region A and then region B, it is not possible for the agent to know whether it should move towards B given only the current position - it needs to know whether it has visited *A* already. For Linear Temporal Logic (LTL), the history-dependence is addressed by translating the formulae into automata that contain history information [6]. The authors of [14] translated (truncated) LTL specifications into a finite-state automata and used reinforcement learning to train a feedforward NN for predicting satisfying control policies. However, STL is not equipped with such an automaton. In [15] and [16], the specification is restricted to a fragment of STL, such that the progress towards satisfaction can be checked with a partial trajectory. Control policies are inferred using Q-learning. Besides the restriction on the STL structure, these works also require the initial partial trajectory to be known. Similarly, the authors of [17] applied reinforcement learning methods to learn control policies enforcing the satisfaction of STL fragments. Most recently, [18] used a RNN-like recurrent computation graph to compute robustness of STL formulae. By

Manuscript received September 14, 2020; revised November 28, 2020; accepted December 22, 2020. Date of publication January 8, 2021; date of current version June 23, 2021. This work was supported by NSF under Grant IIS-1723995 and Grant IIS-2024606. Recommended by Senior Editor L. Menini. *(Corresponding author: Wenliang Liu.)*

Wenliang Liu is with the Department of Mechanical Engineering, Boston University, Boston, MA 02135 USA (e-mail: wliu97@bu.edu).

Noushin Mehdipour is with the Department of Systems Engineering, Boston University, Boston, MA 02446 USA (e-mail: noushinm@bu.edu).

Calin Belta is with the Department of Mechanical/Systems Engineering, Boston University, Boston, MA 02215 USA (e-mail: cbelta@bu.edu).

Digital Object Identifier 10.1109/LCSYS.2021.3049917

allowing back-propagation of robustness gradients, a controller was synthesized to satisfy a STL formula.

RNNs have internal states (memory) units that can store history. In this letter, we propose a feedback RNN controller, which predicts the control policy at each state based on the current state and the history of the system, to address the history-dependence of STL satisfaction. One important advantage of a feedback controller is its tolerance to disturbance. We demonstrate that the feedback structure of RNNs allows us to handle system disturbance and safety requirements that were not known previously (during training). These are enforced using Control Barrier Functions (CBF) [19].

This idea is related to [14], where CBFs were used as shields to guarantee safety for both training and execution phases of a reinforcement learning framework. The authors of [20] also trained a NN-based controller using imitation learning with CBF safety requirements. In contrast to our work, which uses RNN to accomplish STL specifications, [20] did not consider temporal logic specifications, and the NN was solely used to solve an optimization problem with CBF constraints in a reachability problem.

## II. Notation and Preliminaries

### A. Signal Temporal Logic (STL)

An $n$-dimensional real-valued signal is denoted as $S = s_0 s_1 \dots$, where $s_k \in \mathbb{R}^n$, $k \in \mathbb{Z}_{\geq 0}$. The STL *syntax* [3] is defined and interpreted over $S$:

$$\varphi := \top \mid \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathbf{U}_I \varphi_2, \tag{1}$$

where $\varphi$, $\varphi_1$, $\varphi_2$ are STL formulae, $\top$ is the logical *True*, $\mu$ is a *predicate* over signals, $\neg$ and $\wedge$ are the Boolean *negation* and *conjunction* operators. The Boolean constant $\bot$ (*False*) and *disjunction* $\vee$ can be defined from $\top$, $\neg$, and $\wedge$ in the usual way. $I = [a, b] = \{k \in \mathbb{Z}_{\geq 0} \mid a \leq k \leq b;\ a, b \in \mathbb{Z}_{\geq 0}\}$ denotes a bounded time interval and $\mathbf{U}$ is the temporal *until* operator. The temporal operators *eventually* and *always* are defined as $\mathbf{F}_I \varphi := \top \mathbf{U}_I \varphi$ and $\mathbf{G}_I \varphi := \neg \mathbf{F}_I \neg\varphi$, respectively. $\varphi_1 \mathbf{U}_I \varphi_2$ states that "$\varphi_2$ becomes true at some time point within $I$ and $\varphi_1$ must be always true prior to that." $\mathbf{F}_I \varphi$ is satisfied if "$\varphi$ becomes *True* at some time in $I$" while $\mathbf{G}_I \varphi$ is satisfied if "$\varphi$ is *True* at all times in $I$". Predicates are of the form $\mu := l(s_k) \geq 0$, where $l : \mathbb{R}^n \to \mathbb{R}$ is a Lipschitz continuous function.

The STL *qualitative semantics* determines *whether* a signal $S$ satisfies a given specification $\varphi$, i.e., $S \models \varphi$, or not, i.e., $S \not\models \varphi$. Its *quantitative semantics*, or *robustness*, assigns a real value to measure *how much* a signal satisfies $\varphi$. Multiple functionals have been proposed to capture the STL robustness [4], [8], [10], [11]. In this letter, we use the Arithmetic-Geometric Mean (AGM) robustness [9] which is a *sound* score, i.e., a strict positive robustness indicates satisfaction of the specification, and a strict negative robustness indicates violation. However, the frameworks presented in this letter are applicable to all robustness functionals in literature. As opposed to the traditional robustness [4], which only captures the most extreme satisfaction (or violation), AGM employs arithmetic and geometric means over all the satisfying (or violating) sub-formulae and time points in a formula and can highlight the level and frequency of satisfaction. We denote the AGM robustness of $\varphi$ at time $k$ with respect to signal $S$ by $\eta(\varphi, S, k)$. For brevity, we denote $\eta(\varphi, S, 0)$ by $\eta(\varphi, S)$. The time horizon of a STL formula $\varphi$ denoted by $hrz(\varphi)$ is the smallest time point in the future for which signal values are needed to compute the robustness at the current time [21].

### B. Discrete-Time Dynamics and Control Barrier Functions

Consider a discrete-time control system given by

$$q_{k+1} = f(q_k, u_k), \tag{2}$$

where $q_k \in \mathcal{Q} \subset \mathbb{R}^n$ is the state ($q_0$ is the initial state) and $u_k \in \mathcal{U} \subset \mathbb{R}^m$ is the control input at time $k$, and $f : \mathcal{Q} \times \mathcal{U} \to \mathcal{Q}$ is a Lipschitz continuous function. Let $u_{0:K-1}$ denote the control sequence $u_0 \dots u_{K-1}$. The system trajectory $q_0 q_1 \dots q_K$ generated by applying $u_{0:K-1}$ starting at $q_0$ is denoted by $\boldsymbol{q}(q_0, u_{0:K-1})$.

Let $b : \mathbb{R}^n \to \mathbb{R}$. The set $\boldsymbol{C} = \{q \in \mathbb{R}^n \mid b(q) \geq 0\}$ is called (forward) *invariant* for system (2) if all its trajectories remain in $\boldsymbol{C}$ for all times, if they originate in $\boldsymbol{C}$.

The function $b$ is a (discrete-time, exponential) *Control Barrier Function* (CBF) [22] for system (2) if there exist an $\alpha \in [0, 1]$, and for each $q_k$ there exists a $u_k \in \mathcal{U}$ such that:

$$\begin{aligned} b(q_0) &\geq 0 \\ b(q_{k+1}) + (\alpha - 1) b(q_k) &\geq 0, \quad \forall k \in \mathbb{Z}_{\geq 0}, \end{aligned} \tag{3}$$

where $q_{k+1}$, $q_k$, and $u_k$ are related by (2). The set $\boldsymbol{C}$ is invariant for system (2) if there exists a CBF $b$ as (14). This invariance property is usually referred to as *safety*. In other words, the system is safe if it stays inside the set $\boldsymbol{C}$.

## III. Problem Statement and Approach

In this section, we formally state the STL control synthesis problem and its direct solution, which is later used to generate the dataset for training RNN controllers (detailed in Section IV).

Consider system (2) starting at $q_0 \in \mathbb{R}^n$ and a differentiable cost function $J(u_k, q_{k+1})$ representing the cost of ending up at state $q_{k+1}$ by applying control input $u_k$ at time $k$. Assume that temporal logic requirements are given by a STL formula $\varphi$ interpreted over the system states $q_0 \dots q_K$ where $K$ is the final planning horizon. For simplicity, we assume that $K = hrz(\varphi)$. However, $K$ could be any integer greater than or equal to $hrz(\varphi)$. Suppose there are $N$ safety requirements given as CBF constraints $b_i(q_k) > 0$ (see Section II-B), where $i = 1, \dots, N$, $k = 0, \dots, K$. Let $\mathbf{b} : \mathbb{R}^n \to \mathbb{R}^N$, where $\mathbf{b} = (b_1, \dots, b_N)$, and $\mathbf{b}(q_k) > 0$ is interpreted componentwise.

Given system dynamics (2), cost function $J$, STL formula $\varphi$, initial state $q_0$ and safety requirement $\mathbf{b}(q_k) > 0$, we want to find a control sequence $u_{0:K-1}$ that maximizes the STL robustness $\eta(\varphi, \boldsymbol{q}(q_0, u_{0:K-1}))$ as well as minimizing the cost function $\sum_{k=0}^{K-1} J(u_k, q_{k+1})$ and satisfying the safety requirements $\mathbf{b}(q_k) > 0,\ k = 0, \dots, K$.

Synthesizing the control sequence $u_{0:K-1}$ in one shot and applying it to the entire planning horizon forms an open loop controller. However, this formulation would fail to satisfy the specifications if the actual system trajectory deviates from the synthesized one due to the existence of disturbances in the system dynamics or changes in the safety constraints (e.g., moving obstacles). Instead, we propose to find the optimal control at each time based on the current and past[1] states

[1]State history is necessary to decide STL satisfaction, see Sections I and II-A

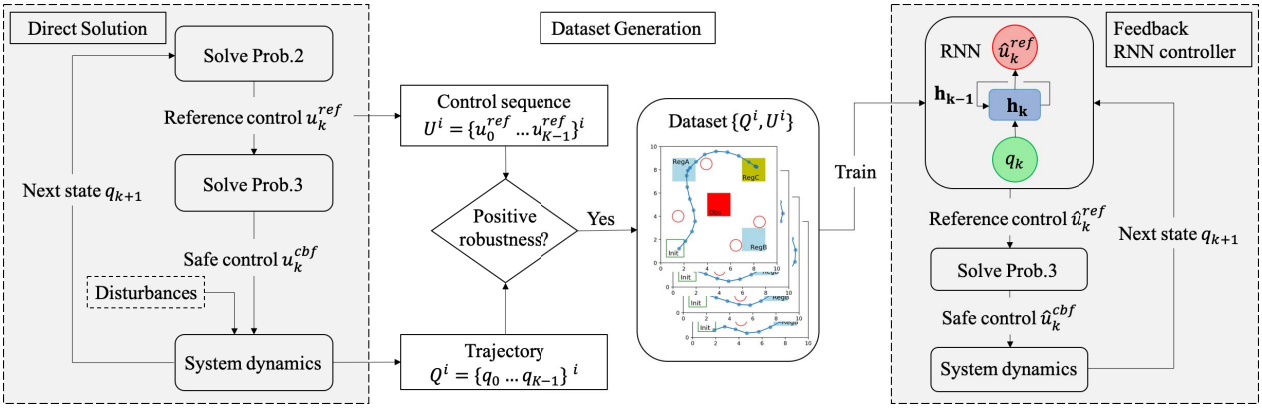



Fig. 1. Overall approach: **Left:** Safe trajectories are generated using gradient-based optimization and CBF; **Middle:** Safe and satisfying trajectories (with positive robustness) and the corresponding reference controls are added to a state-control dataset; **Right:** a RNN is trained on the dataset to predict reference controls for STL satisfaction. A safe feedback RNN controller is synthesized using CBF.

of the system, which gives a history-dependent state feedback controller. Specifically, at each time $k$, the optimization variable $u_{k:K-1}$ covers the rest of the time and the feedback information includes the current state $q_k$ and the *history trajectory* $q_0 \dots q_{k-1}$ (this property is called *history-dependence* of STL). However, solving the optimization problem at each time is time-consuming, which is a problem for real-time implementations. Moreover, the optimization may converge to a local optimum (negative robustness). We address these limitations by training a RNN to predict the control policy at each time. NNs execute very fast. They can take a long time to train, but this computation is performed off-line (before deployment). Our goal is to make the RNN controller flexible, i.e., we want the trajectories generated from the predicted RNN control input to be able to meet the STL specifications under various safety constraints (e.g., unforeseen or dynamic safety constraints), without a need to re-train the RNN when the safety constraints change.

In order to generate a dataset for a flexible RNN, we decompose the optimization problem at each time into two problems: Pb. 1 and Pb. 2. The solution to Pb. 1 provides a *reference control* sequence that gives the "direction" towards the satisfaction of the STL formula but does not consider the safety constraints. In Pb. 2, the first control input (input at the current time) in the reference control sequence is modified (if needed) using CBFs to provide a *safe control* which is applied to the system to move to the next state. Pb. 1 and Pb. 2 are recursively solved at each time until the final time is reached, as shown in Fig. 1 (left). At each time, the current (safe) system state and the (possibly unsafe) reference control are added to ordered sequences of previous states and previous reference controls, respectively. At the final time, the two sequences are combined as a data pair to generate a state-control dataset, on which the RNN is trained (middle of Fig. 1). This framework enables the RNN to predict the reference control at each time based on the current state and the history trajectory. The reference control drives the next state of the system towards STL satisfaction, and is modified by solving Pb. 2 to ensure it is safe as shown in Fig. 1 (right).

There are two main advantages of training the RNN on the reference control (instead of the safe control) and using CBF to guarantee safety. First, we can accommodate safety constraints different from those in the dataset. Otherwise, if the RNN was trained on the safe control, it would assume the safety constraints in the dataset used for training always exist. Second, the final trajectory is guaranteed to be safe independent of the performance of the RNN. Even though safety of the predicted control input is guaranteed after RNN, we still solve Pb. 2 during dataset generation to enlarge the search space (i.e., explore more states that might appear due to various safety constraints and include more data in the dataset).

We propose two versions of Pb. 1 - either can be used depending on the structure and length of the STL formula.

*Problem 1.A (Reference Control):* Given system dynamics (2), cost function $J$, STL formula $\varphi$, current state $q_k$ and history trajectory $q_0 \dots q_{k-1}$, reference control $u^{\text{ref}}_{k:K-1}$ at time $k \in [0, K-1]$ is found by:

$$\begin{aligned} u^{\text{ref}}_{k:K-1} = \arg\max_{u_{k:K-1}} \quad & \eta(\varphi, q_0 \dots q_{k-1}\boldsymbol{q}(q_k, u_{k:K-1})) \\ & - \lambda \sum_{j=k}^{K-1} J(u_j, q_{j+1}) \\ \text{s.t.} \quad & u_j \in \mathcal{U} \subset \mathbb{R}^m, j = k, \dots, K-1 \\ & q_{j+1} = f(q_j, u_j), j = k, \dots, K-1 \end{aligned} \tag{4}$$

By solving Pb. 1.A at time $k$, we find a reference trajectory $\boldsymbol{q}(q_k, u_{k:K-1})$ which along with the history trajectory satisfies the STL formula, i.e., $q_0 \dots q_{k-1}\boldsymbol{q}(q_k, u_{k:K-1}) \models \varphi$.

*Example 1:* Consider a robot in a 2-dimensional workspace in Fig. 2(a). The specification is to "*eventually* visit *RegA or RegB* within [1,10] *and eventually* visit *RegC* within [11,20] *and always* avoid *Obs*", written as a STL formula:

$$\begin{aligned} \varphi_1 = (\mathrm{F}_{[1,10]}(RegA \vee RegB)) \wedge (\mathrm{F}_{[11,20]}RegC) \\ \wedge (\mathrm{G}_{[0,20]}\neg Obs), \end{aligned} \tag{5}$$

with $hrz(\varphi_1) = 20$. Consider the trajectory from Fig. 2(a), and (current) state $q_9$ at time $k = 9$. The blue trajectory $q_0 \dots, q_8$ is the history trajectory, and the red trajectory $q_{10} \dots q_{20}$ is the synthesized trajectory from the solution of Pb. 1.A.

If the horizon of $\varphi$ is large, Pb. 1.A may become prohibitively expensive. If $\varphi = \mathrm{G}_{[0,k_1]}\phi$, we can use a model predictive control (MPC) approach [23] to shorten the optimization (planning) horizon. Let $h^\phi = hrz(\phi)$ and let $h_p$ denote the (shorter) prediction horizon. Instead of optimizing the entire trajectory over $K = k_1 + h^\phi$ steps, in

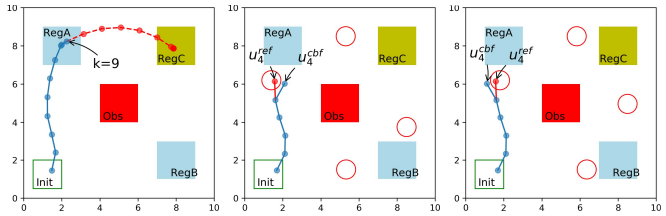



Fig. 2. (a): History trajectory (blue) and trajectory to be optimized (red) at time $k = 9$. (b) and (c): Reference control $u_4^{ref}$ at time $k = 4$ steering the robot to the red point and safe control $u_4^{cbf}$ steering the robot to the blue point. The history trajectory, current state $q_4$, and reference control $u_4^{ref}$ are the same in (b) and (c). The positions of the obstacles are different, which result in different $u_4^{cbf}$.

a MPC framework, we optimize the trajectory for the next $H = h_p + h^\phi$ steps by recursively maximizing the robustness of $\mathrm{G}_{[0,h_p]}\phi$ with respect to the partial trajectory $\boldsymbol{q}(q_k, u_{k:k+H-1})$, $k = 0, 1, \ldots, K-H$. For example, at time $k = 0$, we maximize the robustness of $\mathrm{G}_{[0,h_p]}\phi$ with respect to $q_0, q_1, \ldots, q_H$; at $k = 1$, we maximize the robustness of $\mathrm{G}_{[0,h_p]}\phi$ with respect to $q_1, q_2, \ldots, q_{H+1}$, etc. We need to ensure that, when moving forward, the satisfaction of $\phi$ that was obtained during the previous optimizations still holds. Therefore, when maximizing the robustness of $\mathrm{G}_{[0,h_p]}\phi$ with respect to the partial trajectory starting from time $k$, we need to enforce the robustness of $\phi$ to remain positive at the previous $h^\phi - 1$ steps [23]. Formally, we have:

*Problem 1.B (Reference Control using MPC):* At time $k \in [h^\phi - 1, K - H]$, given system dynamics (2), cost function $J$, STL formula $\varphi = \mathrm{G}_{[0,k_1]}\phi$, current state $q_k$ and history trajectory $q_{k-h^\phi+1} \ldots q_{k-1}$, find reference control $u^{\mathrm{ref}}_{k:k+H-1}$[2]:

$$
\begin{aligned}
u^{\mathrm{ref}}_{k:k+H-1} = \ & \arg\max_{u_{k:k+H-1}} \eta(\mathrm{G}_{[0,h_p]}\phi, \boldsymbol{q}(q_k, u_{k:k+H-1})) \\
& - \lambda \sum_{j=k}^{k+H-1} J(u_j, q_{j+1}) \\
\text{s.t.} \quad & u_j \in \mathcal{U} \subset \mathbb{R}^m, \ j = k, \ldots, k+H-1 \\
& q_{j+1} = f(q_j, u_j), \ j = k, \ldots, k+H-1 \\
& \eta(\phi, q_{k-h^\phi+1+i}, \ldots, q_{k-1}\boldsymbol{q}(q_k, u_{k:k+i})) > 0, \\
& \quad i = 0, \ldots, h^\phi - 2.
\end{aligned} \tag{6}
$$

The solution to Pb. 1.A or Pb. 1.B is the *reference control* without considering safety constraints. The reference control at the current time $u^{\mathrm{ref}}_k$ will be added to the sequence of reference controls for dataset generation, and subsequently modified to satisfy the safety constraints:

*Problem 2 (Safe Control):* At time $k \in [0, K-1]$, given system dynamics (2), current state $q_k$, safety constraints $\mathbf{b}(q_k) > 0$, and reference control $u^{\mathrm{ref}}_k$ (possibly unsafe), safe control policy $u^{\mathrm{cbf}}_k$ is found by:

$$
\begin{aligned}
u^{\mathrm{cbf}}_k = \arg\min_{u_k} \quad & \|u_k - u^{\mathrm{ref}}_k\|^2 \\
\text{s.t.} \quad & \mathbf{b}(f(q_k, u_k)) + (\alpha - 1)\mathbf{b}(q_k) > 0, \\
& u_k \in \mathcal{U} \subset \mathbb{R}^m
\end{aligned} \tag{7}
$$

*Remark 1:* We assume CBF parameters are tuned such that control inputs that satisfy all CBF constraints always exist.

[2]Note that, when $k < h^\phi - 1$ or $k > K - H$, the corresponding horizons in (6) need to be modified [23].

*Remark 2:* The safe control might violate the STL specification. In this letter, we prioritize safety over specification satisfaction.

*Example 2:* At time $k = 4$, the reference control $u_4^{ref}$, which steers the robot from Ex. 1 to satisfy $\varphi_1$ (go to *RegA*), is computed from Pb. 1. Assume that there are 4 circular obstacles appearing at time $k = 4$, as shown in Fig. 2(b) and Fig. 2(c), under the reference control $u_4^{ref}$, the robot will collide with one of the obstacles. However, by solving Pb. 2, we can modify the reference control to $u_4^{cbf}$ to avoid collision. With the same STL formula and current state and history trajectory, the reference control $u_4^{ref}$ is determined, while the safe control $u_4^{cbf}$ depends on the different positions of obstacles (Fig. 2(b) and 2(c)). Since the positions of obstacles when testing (deploying) the RNN are unforeseen, we save the current state $q_4$ and the reference control $u_4^{ref}$ into the dataset to teach the RNN the reference control towards STL satisfaction. When testing the RNN, we modify its output depending on the positions of obstacles at that moment.

**Direct solution** The method used to generate the dataset, which we refer to as the *direct solution*, is summarized below. At each time $k$, we solve Pb. 1.A or Pb. 1.B, depending on the structure of $\varphi$, to get a reference control sequence $u^{\mathrm{ref}}_{k:K-1}$ or $u^{\mathrm{ref}}_{k:k+H-1}$. We take $u^{\mathrm{ref}}_k$ and modify it by solving Pb. 2 to get the safe control input $u^{\mathrm{cbf}}_k$. By applying $u^{\mathrm{cbf}}_k$ to the system dynamics (also adding a disturbance $w \in \mathcal{W} \subset \mathbb{R}^n$ such that $q_{k+1} = f(q_k, u_k^{cbf}) + w$ to further enlarge the exploration space), we find the next state $q_{k+1}$, and Pb. 1 and Pb. 2 are recursively solved until the final time is reached. Both Pb. 1 and Pb. 2 are solved using gradient based optimization methods.

## IV. RNN Controller Synthesis

In this section, we describe our approach in using the direct solution for dataset generation and training RNN controllers.

**Dataset Generation** In order to create a dataset for RNN, we generate a set of $M$ random initial states $q_0^i$, $i = 1, \ldots, M$, and corresponding safety constraints $\mathbf{b}^i(q_k) > 0, k = 0, \ldots, K$, $i = 1, \ldots, M$. For each $q_0^i$ and associated $\mathbf{b}^i$, we can use the direct solution to generate a safe trajectory denoted by $Q^i = \{q_0 \ldots q_K\}^i$ and the corresponding reference control sequence denoted by $U^i = \{u_0^{ref} \ldots u_{K-1}^{ref}\}^i$. Together, $(Q^i, U^i)$ is considered as a paired state-control data. If $Q^i$ has positive robustness, i.e., $\eta(\varphi, Q^i) > 0$, the state-control pair $(Q^i, U^i)$ is added to the dataset $\mathbf{D}$ (as illustrated in Fig. 1).

**Feedback RNN Controller** Due to the *history-dependence* of STL, the control at each time depends on the current state and the history trajectory. Formally, at each time $k$, $u_k^{ref} = g(q_0, \ldots, q_k)$. Since neural networks are known to be universal function approximators [24], the feedback function $g$ can be approximated by a RNN with weights $(\boldsymbol{W}_1, \boldsymbol{W}_2)$:

$$
\begin{aligned}
\mathbf{h}_k &= \mathcal{R}(q_k, \mathbf{h}_{k-1}, \boldsymbol{W}_1) \\
\hat{u}_k^{ref} &= \mathcal{N}(\mathbf{h}_k, \boldsymbol{W}_2),
\end{aligned} \tag{8}
$$

where $\mathbf{h}_k$ is the RNN hidden state at time $k$, which encodes the history trajectory, and $\hat{u}_k^{ref}$ is the RNN output, which is the predicted control policy. By passing the history trajectory as the hidden state, as shown in Fig. 3, RNN can manage the history-dependence of the STL satisfaction.

The RNN formulated in (8) is trained on the state-control dataset $\mathbf{D}$ such that the prediction error between the reference

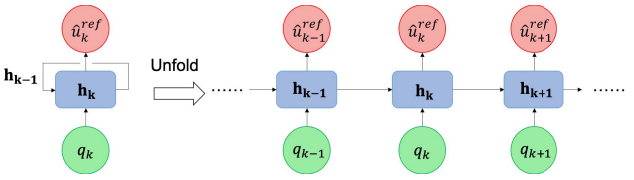
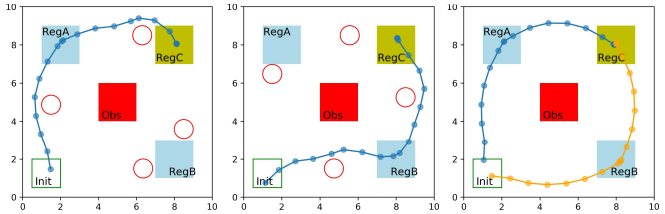



Fig. 3. Illustration of the feedback RNN controller.




(a) (b) (c)

Fig. 4. Trajectories generated using our RNN-CBF framework. Only the solid obstacle (rectangle) was known during the RNN training. CBF guarantees safety against the random unknown obstacles (circles) if they exist.

control $u_k^{ref}$ (from the dataset) and the predicted control $\hat{u}_k^{ref}$ at all times $k = 0, 1, \dots K-1$ is minimized:

$$\min_{\boldsymbol{W}_1, \boldsymbol{W}_2} \sum_{\mathbf{D}} \sum_{k=0}^{K-1} \|\mathcal{N}(\mathcal{R}(q_k, \mathbf{h}_{k-1}, \boldsymbol{W}_1), \boldsymbol{W}_2) - u_k^{ref}\|^2. \quad (9)$$

To implement the RNN, we use a Long Short Term Memory (LSTM) network [25]. Similar to [13], we also apply a hyperbolic tangent function on the RNN outputs (i.e., the predicted control inputs at each time) in order to meet the control constraints $u_k \in \mathcal{U}$.

To guarantee the safety of the trajectory, Pb. 2 is solved to adjust $\hat{u}_k^{ref}$ and obtain a safe control $\hat{u}_k^{cbf}$. This safe control $\hat{u}_k^{cbf}$ is applied to the system to steer it to the next state $q_{k+1}$, and the process is repeated until reaching the final time.

## V. Case Studies

In this section, we show the efficacy of our proposed RNN framework and compare our results with the direct solution. All algorithms were implemented in Python running on a Mac with a 2.6GHz Core i7 CPU and 16GB of RAM. We used Sequential Quadratic Programming (SQP) [26] to solve Pb. 1 and Pb. 2. The RNN was implemented using Pytorch [27].

We present two case studies, which illustrate the proposed framework using Pb. 1.A (Case Study 1) and Pb. 1.B (Case Study 2), respectively. For both, the cost function is defined as $J = \frac{1}{2}\sum_{k=0}^{K-1} \|u_k\|^2$. The RNN structure consists of a LSTM network with 2 hidden layers and 64 nodes in each layer. The dataset $\mathbf{D}$ contains state-control pairs $(Q, U)$ with random initial states in a fixed region. The trained RNN controller is tested on 1000 random initial states (in the same fixed region) with random safety constraints.

*Case Study 1:* Consider the scenario from Ex. 1, and assume the discrete-time dynamics of the robot is given by:

$$\begin{aligned} x_{k+1} &= x_k + \frac{v_k}{\omega_k}(\sin(\theta_k + \omega_k) - \sin\theta_k), \\ y_{k+1} &= y_k + \frac{v_k}{\omega_k}(\cos\theta_k - \cos(\theta_k + \omega_k)), \\ \theta_{k+1} &= \theta_k + \omega_k. \end{aligned} \quad (10)$$

$q = (x, y, \theta)$ is the state vector with position and orientation of the robot, and the control input $u = (v, \omega)$ contains the forward and angular speeds, where $v \in [0, 1]$, $\omega \in [-0.5, 0.5]$.

Besides the fixed obstacle specified in Eq. (5), we assume random circular obstacles emerge in the environment (see Fig. 4). These obstacles are considered as additional safety constraints that can be enforced by CBFs $b_i$ (from Eq. (3)):

$$b_i(q) = (x - x_{o,i})^2 + (y - y_{o,i})^2 - r_{o,i}^2, \quad i = 1, 2, 3, 4 \quad (11)$$

where $(x_{o,i}, y_{o,i})$ is the center of the $i^{th}$ circular obstacle and $r_{o,i}$ is its radius.

TABLE I
Computation Times for the Direct and RNN Solutions

| Time \ Methods | Direct solution | RNN solution |
|---|---|---|
| Solve Pb.1.A (single time) | $0.635s$ | $0.000417s$ |
| Generate entire trajectory | $12.8s$ | $0.0582s$ |

The procedure described in the direct solution (with Pb. 1.A) is applied to generate a dataset, considering $\lambda = 0$ in (4) and $\alpha = 0.7$ in (7). The norm in (7) is also modified to $(v_k - v_k^{ref})^2 + \gamma(\omega_k - \omega_k^{ref})^2$ where $\gamma = 0.03$ in order to encourage the robot to turn instead of slowing down when approaching an obstacle. Generating a dataset of 500 (satisfying) trajectories takes about 2 hours, and training the RNN on this dataset for 300 epochs takes about 2 minutes.

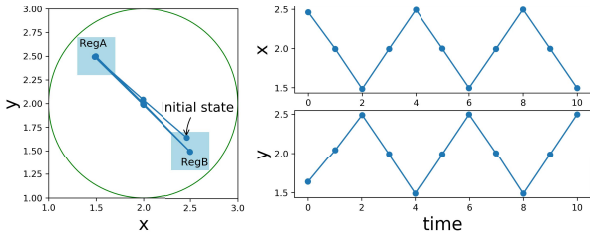
The success rate (obtaining safe and satisfying trajectories) for the RNN solution is 99.5%. Fig. 4 shows sample trajectories for random initial conditions and safety constraints (circular obstacles in Fig. 4a and 4b) obtained by applying the safe control $\hat{u}^{cbf}$. As illustrated, by separating the CBF from the RNN controller, safety constraints are guaranteed to be satisfied, even for previously unknown safety constraints, and independent of the performance of the RNN (Fig. 4a and Fig. 4b). Moreover, since the RNN is trained on the reference control inputs, the trajectory generated from the predicted control inputs avoids unnecessary re-directions when no additional safety constraints exist (Fig. 4c).

The average normalized robustness for the trajectories generated by the RNN solution is 0.0425, and for the trajectories in dataset $\mathbf{D}$ from the direct solution (all of which are trajectories with positive robustness) is 0.0423. Since the random obstacles serve as disturbances during dataset generation, no additional disturbances are added, hence the robustness comparison of both solutions is fair. This suggests that the performance of the RNN controller is as good as the direct solution. Computation times for the direct solution and the RNN solution are shown in Tab. I. The comparison confirms that the proposed RNN controller is much faster and suitable for real-time synthesis and planning applications.

*Case Study 2:* Consider a discrete-time system given by:

$$\begin{aligned} x_{k+1} &= x_k + u_{x,k}, \\ y_{k+1} &= y_k + u_{y,k}, \end{aligned} \quad (12)$$

in a configuration shown in Fig. 5a. $q = (x, y)$ is the state vector, and $u = (u_x, u_y)$ is the control input with $\mathcal{U} = [-0.6, 0.6]^2$. The specification is "for all times in $[0, 7]$, *eventually* visit *RegA* every 3 steps *and eventually* visit *RegB*




(a) (b)

Fig. 5. A trajectory generated using our RNN-CBF framework satisfies $\varphi_2$ while remaining in the safe region (the green circle).

every 3 steps", which translates to the STL formula:

$$\varphi_2 = \mathrm{G}_{[0,7]}(\mathrm{F}_{[0,3]}RegA \wedge \mathrm{F}_{[0,3]}RegB). \tag{13}$$

With $\phi = \mathrm{F}_{[0,3]}RegA \wedge \mathrm{F}_{[0,3]}RegB$, we have $h^{\phi} = 3$. Let $q_0$ be a random position inside $RegB$. We use Pb. 1.B to find reference control inputs and generate a dataset $\mathbf{D}$ based on the direct solution procedure. In this example, we set $h_p = 0$, $\lambda = 10^{-6}$, and $\alpha = 0.8$. We also add a random disturbance $w \in [-0.05, 0.05]^2$ to the system dynamics when generating the dataset. Safety is specified as a circular region (Fig. 5):

$$b(q) = -(x - x_{safe})^2 - (y - y_{safe})^2 + r_{safe}^2, \tag{14}$$

with $(x_{safe}, y_{safe})$ and $r_{safe}$ being its center and radius.

Generating a dataset of 1000 satisfying trajectories takes about 40 minutes and training the RNN for 300 epochs takes about 2 minutes. Fig. 5a shows a sample trajectory obtained by applying the safe control $\hat{u}^{cbf}$. As illustrated in Fig. 5b, the system periodically visits $RegA$ and $RegB$ every 3 steps. In this example, the RNN controller produces satisfying trajectories with a success rate of 100%. The computation times for the direct solution and RNN solution are $2.252s$ and $0.00885s$, respectively, which also illustrates the advantages of the RNN controller for real-time applications.

## VI. Conclusion and Future Work

We proposed a RNN framework to synthesize feedback control policies for a system under STL specifications. We used CBF to modify the control policies predicted by the RNN to guarantee safety, even in cases where safety constraints were unknown during the RNN training phase. We showed that our proposed RNN-CBF solution can be executed in real-time, while guaranteeing safety and achieving high success rate for STL satisfaction. Future research investigates utilizing the proposed RNN framework in model-free reinforcement learning approaches for control synthesis under STL specifications.

## References

[1] P. Tabuada, *Verification and Control of Hybrid Systems: A Symbolic Approach*, Boston, MA, USA: Springer, 2009.

[2] C. Belta, B. Yordanov, and E. A. Gol, *Formal Methods for Discrete-Time Dynamical Systems*, vol. 89. Cham, Switzerland: Springer, 2017.

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

[4] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Proc. Int. Conf. Formal Model. Anal. Timed Syst.*, 2010, pp. 92–106.

[5] 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 *Proc. 53rd IEEE Conf. Decis. Control*, Los Angeles, CA, USA, 2014, pp. 81–87.

[6] C. Belta and S. Sadraddini, "Formal methods for control synthesis: An optimization perspective," *Annu. Rev. Control Robot. Auton. Syst.*, vol. 2, pp. 115–140, May 2019.

[7] Y. V. Pant, H. Abbas, and R. Mangharam, "Smooth operator: Control using the smooth robustness of temporal logic," in *Proc. IEEE Conf. Control Technol. Appl. (CCTA)*, Mauna Lani, HI, USA, 2017, pp. 1235–1240.

[8] I. Haghighi, N. Mehdipour, E. Bartocci, and C. Belta, "Control from signal temporal logic specifications with smooth cumulative quantitative semantics," in *Proc. IEEE 58th Conf. Decis. Control (CDC)*, Nice, France, 2019, pp. 4361–4366.

[9] N. Mehdipour, C.-I. Vasile, and C. Belta, "Arithmetic-geometric mean robustness for control from signal temporal logic specifications," in *Proc. IEEE Amer. Control Conf. (ACC)*, Philadelphia, PA, USA, 2019, pp. 1690–1695.

[10] P. Varnai and D. V. Dimarogonas, "On robustness metrics for learning STL tasks," in *Proc. Amer. Control Conf. (ACC)*, Denver, CO, USA, 2020, pp. 5394–5399.

[11] Y. Gilpin, V. Kurtz, and H. Lin, "A smooth robustness measure of signal temporal logic for symbolic control," *IEEE Control Syst. Lett.*, vol. 5, no. 1, pp. 241–246, Jan. 2021.

[12] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robot. Auton. Syst.*, vol. 57, no. 5, pp. 469–483, 2009.

[13] S. Yaghoubi and G. Fainekos, "Worst-case satisfaction of STL specifications using feedforward neural network controllers: A lagrange multipliers approach," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 5s, pp. 1–20, 2019.

[14] X. Li, Z. Serlin, G. Yang, and C. Belta, "A formal methods approach to interpretable reinforcement learning for robotic planning," *Sci. Robot.*, vol. 4, no. 37, p. eaay6276, 2019.

[15] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "Q-learning for robust satisfaction of signal temporal logic specifications," in *Proc. IEEE 55th Conf. Decis. Control (CDC)*, Las Vegas, NV, USA, 2016, pp. 6565–6570.

[16] H. Venkataraman, D. Aksaray, and P. Seiler, "Tractable reinforcement learning of signal temporal logic objectives," 2020. [Online]. Available: arXiv:2001.09467.

[17] A. Balakrishnan and J. V. Deshmukh, "Structured reward shaping using signal temporal logic specifications," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Macau, China, 2019, pp. 3481–3486.

[18] K. Leung, N. Aréchiga, and M. Pavone, "Back-propagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods," in *Proc. 14th Int. Workshop Algorithmic Found. Robot.*, 2020, pp. 1–16.

[19] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *Proc. IEEE 18th Eur. Control Conf. (ECC)*, 2019, pp. 3420–3431.

[20] S. Yaghoubi, G. Fainekos, and S. Sankaranarayanan, "Training neural network controllers using control barrier functions in the presence of disturbances," 2020. [Online]. Available: arXiv:2001.08088.

[21] A. Dokhanchi, B. Hoxha, and G. Fainekos, "On-line monitoring for temporal logic robustness," in *Proc. Int. Conf. Runtime Verification*, 2014, pp. 231–246.

[22] A. Agrawal and K. Sreenath, "Discrete control barrier functions for safety-critical control of discrete systems with application to bipedal robot navigation," in *Robotics: Science and Systems*, Cambridge, MA, USA, 2017.

[23] S. Sadraddini and C. Belta, "Robust temporal logic model predictive control," in *Proc. 53rd Annu. Allerton Conf. Commun. Control Comput. (Allerton)*, Monticello, IL, USA, 2015, pp. 772–779.

[24] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[26] D. P. Bertsekas, "Nonlinear programming," *J. Oper. Res. Soc.*, vol. 48, no. 3, p. 334, 1997.

[27] A. Paszke *et al.*, "Automatic differentiation in pytorch," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran Assoc., Inc., 2017.