2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Safe Model-based Control from Signal Temporal Logic Specifications Using Recurrent Neural Networks

Wenliang Liu[1], Mirai Nishioka[2], Calin Belta[1]

Abstract— **We propose a policy search approach to learn controllers from specifications given as Signal Temporal Logic (STL) formulae. The system model, which is unknown but assumed to be an affine control system, is learned together with the control policy. The model is implemented as two feedforward neural networks (FNNs) - one for the drift, and one for the control directions. To capture the history dependency of STL specifications, we use a recurrent neural network (RNN) to implement the control policy. In contrast to prevalent model-free methods, the learning approach proposed here takes advantage of the learned model and is more efficient. We use control barrier functions (CBFs) with the learned model to improve the safety of the system. We validate our algorithm via simulations and experiments. The results show that our approach can satisfy the given specification within very few system runs, and can be used for on-line control.**

I. INTRODUCTION

Temporal logics, such as Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) [1], have been increasingly used as specification languages in robotics, biology, and autonomous driving applications. In this paper, we use Signal Temporal Logic (STL) [2], which was originally developed to monitor temporal properties of real-valued signals. STL is equipped with both qualitative semantics, which measures whether a signal satisfies a formula, and quantitative semantics, known as robustness, which assigns a real value to measure how strongly a signal satisfies a formula [3]. Higher robustness scores indicate stronger satisfactions. Using robustness, controlling a system from an STL specification can be formulated as an optimization problem with robustness as a constraint, as a term in the objective function, or both. Such a problem can be solved using either Mixed Integer Programming (MIP) [4], [5], or gradient-based methods [6]–[9]. However, both MIP and gradient-based methods are computationally expensive, especially for high dimensional systems and long-horizon planning, which prevents real-time implementation.

Neural Network (NN) - based controllers have been proposed to address the computational complexity of the methods described above. NNs are trained offline and executed online, which enables real-time implementation. In [10], the control policy is parameterized as a Feedforward Neural Network (FNN) to maximize the robustness of an STL formula defined over system trajectories. FNNs are memoryless controllers. However, the controller to satisfy an STL formula is, in general, *history-dependent*. For instance, if a specification requires an agent to eventually visit two regions, the agent needs to know which region it has already visited so that it can go towards the other one. A Recurrent Neural Network (RNN) controller, which has memory, was proposed in [11] to satisfy STL specifications. The RNN was trained on a dataset containing satisfying trajectories. However, such a dataset is not always available. [11] used the optimization methods mentioned above to generate the dataset, which is again computationally expensive. The authors of [12] proposed a semi-supervised approach to train an RNN controller such that the resulting trajectory maximizes the STL robustness as well as minimizes the distance from a given dataset of expert demonstrations. Compared with [11], the size of the dataset is largely reduced. All the above approaches assume the system dynamics are known, which is sometimes not the case in practice. RNNs were also used in sequential prediction tasks with STL constraints [13], where neither system dynamics nor controls are considered.

Reinforcement Learning (RL) can solve control problems for systems with unknown dynamics. Recently, RL approaches have been investigated to synthesize controllers satisfying STL specifications. The advantage of combining RL and STL is that the STL robustness can be used to generate RL rewards, which avoids reward hacking [14]. The authors of [15] and [16] used Q-learning to learn control policies that maximize the robustness of STL fragments for systems with discrete state and control spaces. Similar ideas were used for continuous states and controls using deep RL [17], [18]. These methods restrict the specification to be a fragment of STL. Path integral was used for learning control policies from Truncated Linear Temporal Logic (TLTL) [19] and STL [20]. In [21]–[23], automaton-based approaches were used with FNN controllers to satisfy LTL and TLTL specfications, where history information is contained in the automaton state. All the above methods are completely model-free, and are therefore data inefficient, i.e., they require a large number of trials to learn the policy [24].

Model-based policy search methods (e.g., [24], [25]) learn a dynamic model of the system and use it to guide policy search. They are more data-efficient than model-free methods. In this paper, we propose a model-based policy search method that learns both the system model and the control policy from an STL specification. The model is unknown but assumed to be an affine control system. It is implemented as two FNNs with dropout (one for the drift and one for the control directions), which are approximations

This work was partially supported by the National Science Foundation under grants IIS-2024606 and IIS-1723995.

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

[2]Mirai Nishioka is with Commonwealth School, Boston, MA, USA mnishioka@commschool.org

2023 IEEE International Conference on Robotics and Automation (ICRA) | 979-8-3503-2365-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICRA48891.2023.10161201

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 26,2023 at 05:34:14 UTC from IEEE Xplore. Restrictions apply.

of Bayesian neural networks [26] and can reduce model bias [24]. The control policy is an RNN, which outputs the control depending on current state and past states, and therefore captures the memory necessary satisfy arbitrary STL formulas. Only a few system executions are needed for training the model, and no extra executions are needed to train the RNN controller. After training, the RNN controller can be executed very fast, which enables real time control.

Another critical concern for RL is safety: unsafe states should never be reached during training and testing. Control Barrier Functions (CBF) [27] have been proposed to obtain safe control. In [21], CBFs are incorporated with formal methods to guarantee the safety of RL. Typically, using CBF requires the system dynamics to be known. The authors of [28] use Gaussian process to model an uncertain system and use CBFs to guide a safe exploration. Similarly, we consider an unknown system. We propose a method that applies CBF to the learned probabilistic model, which takes into account the model uncertainty. Applying CBF when training the model can also benefit the exploration, because without CBF, we have to often stop the system due to proximity to unsafe states. Another related work considering unknown dynamics is [29], in which FNN controllers are trained, but an open-loop trajectory satisfying the STL specification needs to be computed before each run.

The contributions of this paper are summarized as follows. (1) We propose a novel model-based policy search method to learn a policy that maximizes the expected robustness of a given STL specification with unknown system dynamics. Our approach considers full STL formulae, requires no dataset, and involves only few system executions. (2) We incorporate CBFs into the learning process with the consideration of model uncertainty, which improves the safety and exploration performance. (3) Simulations and experiments are implemented, illustrating the efficacy and scalability of our approach, and the ability of real-time control.

II. Notations and Preliminaries

We consider a discrete-time affine control system:

$$x_{t+1} = f(x_t) + g(x_t)u_t,\ t = 0, 1, \dots \tag{1}$$

where $x_t \in \mathcal{X} \subseteq \mathbb{R}^n$ is the system state at time t, $u_t \in \mathcal{U} \subseteq \mathbb{R}^m$ is the control input at time t, and $f : \mathcal{X} \to \mathcal{X}$, $g : \mathcal{X} \to \mathbb{R}^{n\times m}$ are locally Lipschitz continuous functions modeling the drift and the control directions. Let $x_{t_1:t_2}$ denote the partial state trajectory $x_{t_1}, x_{t_1+1}, \dots, x_{t_2}$ with $t_2 > t_1$.

A. Signal Temporal Logic (STL)

An n-dimensional real-valued signal is denoted as $X = x_0x_1\dots$, where $x_t \in \mathbb{R}^n$, $t \in \mathbb{Z}_{\geq 0}$. The STL *syntax* [2] is defined as:

$$\varphi := \top | \ \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathbf{U}_I \varphi_2, \tag{2}$$

where $\varphi, \varphi_1, \varphi_2$ are STL formulae, $\top$ is the Boolean constant *True*, $\neg$ and $\wedge$ are the Boolean *negation* and *conjunction* operators, respectively. μ is a *predicate* over signals of the form $\mu := l(x_t) \geq 0$, where $l : \mathbb{R}^n \to \mathbb{R}$ is a Lipschitz continuous function. $I = [a, b] = \{t \in \mathbb{Z}_{\geq 0} \mid a \leq t \leq b;\ a, b \in \mathbb{Z}_{\geq 0}\}$ denotes a bounded time interval and $\mathbf{U}$ is the temporal *until* operator. The semantics of STL formulas is defined over signals. $\varphi_1 \mathbf{U}_I \varphi_2$ states that "φ_2 becomes true at some time point within I and φ_1 must be always true prior to that." The Boolean constant $\perp$ (*False*) and *disjunction* $\vee$ can be defined from $\top$, $\neg$, and $\wedge$ in the usual way. Additional 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. $\mathbf{F}_I\varphi$ is satisfied if "φ becomes *True* at some time in I" while $\mathbf{G}_I\varphi$ is satisfied if "φ is *True* at all times in I".

The STL *qualitative semantics* [2] determines *whether* a signal X satisfies a formula φ (written as $X \models \varphi$) or not (written as $X \not\models \varphi$). Its *quantitative semantics*, or *robustness*, assigns a real value to measure *how much* a signal satisfies φ. For instance, let $\varphi = \mathbf{F}_{[0,2]}x \geq 2$. A signal $X_1 = 0, 2, 4, \dots$ should have a higher robustness than $X_2 = 0, 1, 2.01, \dots$ because X_1 satisfies φ stronger. Multiple functionals have been proposed to capture the STL robustness [3], [7]–[9]. In this paper, we use the robustness function proposed in [8], which is a *sound* score, i.e., positive robustness indicates satisfaction of the formula, and negative robustness indicates a violation. As opposed to the traditional robustness [3], the robustness in [8] is differentiable so it is suitable for gradient-based optimization methods. We denote the robustness of φ at time t with respect to signal X by $\rho(\varphi, X, t)$. For brevity, we denote $\rho(\varphi, X, 0)$ by $\rho(\varphi, X)$. The time horizon of an STL formula φ denoted by $hrz(\varphi)$ is the closest time point in the future for which signal values are needed to compute the robustness at the current time [30].

B. Recurrent Neural Network (RNN) Controller

Due to the *history-dependence* property of STL, a feedback controller for system (1) required to satisfy an STL formula over its state at time t, should be a function of the current state and all the history states, i.e., $u_t = \pi(x_{0:t})$. In this paper, we will learn a controller in the form of an RNN [31], which can be formulated as follows:

$$\begin{aligned} \mathbf{h}_t &= \mathcal{R}(x_t, \mathbf{h}_{t-1}, W_1), \\ u_t &= \mathcal{N}(\mathbf{h}_t, W_2), \end{aligned} \tag{3}$$

where $\mathbf{h}_t$ is the RNN hidden state, $t = 0, 1, \dots, T-1$ (T is the final time of interest)At time t, the input of the RNN is the current state x_t and the output is the control u_t for system (1). $\mathcal{R}$ represents fully connected layers with parameters W_1, which update the hidden state based on the input and the previous hidden state. $\mathcal{N}$ denotes layers with parameters W_2, which predict the output based on the hidden state. The output layer is applied a hyperbolic tangent function to satisfy the constraint $u \in \mathcal{U}$ as in [10]. The hidden state $\mathbf{h}_t$ encodes all the previous inputs. $\mathbf{h}_{-1}$ is a zero vector.

C. Discrete Time Control Barrier Function

Consider system (1) and let $b : \mathbb{R}^n \to \mathbb{R}$. The set $\boldsymbol{C} = \{x \in \mathbb{R}^n \mid b(x) \geq 0\}$ is (forward) *invariant* for system (1) if all its trajectories remain in $\boldsymbol{C}$ for all times, if they originate in $\boldsymbol{C}$. Function b is a (discrete-time, exponential) *Control*

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 26,2023 at 05:34:14 UTC from IEEE Xplore. Restrictions apply.

Barrier Function (CBF) [32] for system (1) if there exists $\alpha \in [0,1]$, and for each x_t there exists $u_t \in \mathcal{U}$ such that:

$$\begin{aligned} b(x_0) &\geq 0 \\ b(x_{t+1}) + (\alpha - 1)b(x_t) &\geq 0, \quad \forall t \in \mathbb{Z}_{\geq 0}, \end{aligned} \tag{4}$$

where x_{t+1}, x_t, and u_t are related by (1). The set $\boldsymbol{C}$ is invariant for system (1) if there exists a CBF b as in (4) [32]. 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

Consider system (1) with unknown dynamics f, g and fully observable state. We assume that the initial state x_0 is randomly located in a set $\mathcal{X}_0 \subseteq \mathcal{X}$ with the probability density function $p : \mathcal{X}_0 \to \mathbb{R}$. Consider an STL formula $\varphi = \varphi_{task} \wedge \varphi_{safe}$ with predicates interpreted over x, where φ_{task} is a general full STL formula that specifies the task, while $\varphi_{safe} = \mathbf{G}_{I_1}\phi_1 \wedge \ldots \wedge \mathbf{G}_{I_N}\phi_N$ represents N safety requirements. We assume that $\phi_1, \ldots, \phi_N$ are affine or quadratic predicates, so they can be converted into N affine or quadratic CBFs $b_1, \ldots, b_N$. For example, the satisfaction of $\phi = (a^\top x_t \leq c)$ is equivalent to $b = -a^\top x_t + c \geq 0$, where $a \in \mathbb{R}^n$ and $c \in \mathbb{R}$. We also assume that all CBFs have a relative degree of 1. High-order CBFs will be investigated in future work. Given a finite time horizon $T \geq hrz(\varphi)$ (for simplicity we assume that $T = hrz(\varphi)$), our goal is to find a control policy $u_t = \pi(x_{0:t}),\ t = 0, \ldots, T-1$ that makes trajectories $x_{0:T}$ starting from $\mathcal{X}_0$ satisfy φ. Since the robustness is sound, we can use it as a reward, and find a controller that maximize its expectation.

In this paper we use a parameterized policy, which is implemented by an RNN. The control u_t is computed by using (3) recursively. We denote $u_t = \pi(x_{0:t}; W)$, where $W = (W_1, W_2)$ captures the RNN parameters. We now formally state the STL control synthesis problem as follows:

Problem 1. Given system (1) with unknown dynamics f and g, fully observable state, initial state x_0 with distribution $p : \mathcal{X}_0 \to \mathbb{R}$, and given an STL formula φ over x with $T = hrz(\varphi)$, find the optimal policy parameters W^* that maximize the expected STL robustness, i.e.,

$$\begin{aligned} &W^* = \arg\max_W E_{p(x_0)}[\rho(\varphi, x_{0:T})] \\ &\text{s.t. } x_{t+1} = f(x_t) + g(x_t)\pi(x_{0:t}; W), t = 0, \ldots, T-1 \end{aligned} \tag{5}$$

Since the system model is unknown, Pb. 1 cannot be solved directly. We first execute the system with random controls and collect the resulting system state transitions to form a dataset. We train two FNNs on this dataset to approximate the model (one for f and one for g). Then we solve Pb. 1 using the learned model to improve the control policy. We alternately improve the model using the data collected by applying the new policy, and improve the policy using the new model until convergence. CBFs are applied at each time the system is executed to improve safety except for generating the initial dataset. We stop the system and terminate this trial if it gets too close to unsafe states. The overall model-based safe policy search framework is shown in Fig. 1. The technical details are presented in Sec. IV.

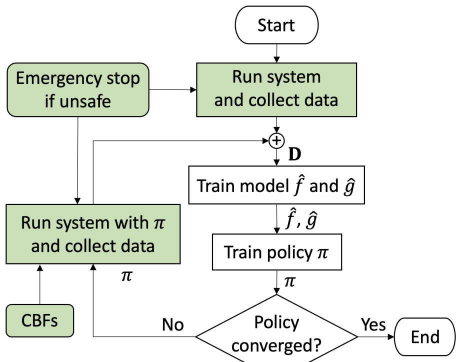

Fig. 1: Our overall approach to Pb. 1: We first generate an initial dataset $\mathbf{D}$, and then we train the system model and the control policy alternately. Safety is improved by CBFs and ensured by emergency stops. The iteration terminates after convergence.

Model-based learning is much more data-efficient than model-free RL methods, i.e., it requires fewer trials to learn the policy [24]. Alternately training the system model and the control policy benefits both. With a better policy, more states that the system may reach in order to finish the task can be explored, which results in a better model. With a more precise model, the policy can be better tuned. On the other hand, using CBF can not only improve safety but also make data collection more efficient, because the system can evolve for more steps in each trial without emergency stops.

IV. Model-Based safe policy search

In this section, we introduce the model-based safe policy search approach. The system model learning, the control policy improvement, and the overall algorithm are presented in Secs. IV-A, IV-B and IV-C, respectively.

A. System Model Learning

Although the true system dynamics (1) are assumed to be deterministic, learning a deterministic model to train the policy suffers from model bias, because an inaccurate model is given full confidence [24]. Hence, we use a probabilistic model implemented by two feedforward neural networks (FNNs) $\mathcal{F}$ and $\mathcal{G}$ with dropout to approximate the dynamics (1) given the current state x_t and the control input u_t:

$$\hat{x}_{t+1} = \mathcal{F}(x_t; W_f, Z_f) + \mathcal{G}(x_t; W_g, Z_g)u_t \tag{6}$$

where W_f and W_g are FNN parameters, Z_f and Z_g are random dropout masks that randomly deactivate the nodes in the dropout layers with probability p_d. As an approximation of a Bayesian Neural Network, an FNN with dropout can not only reduce model bias (detailed in Sec. IV-B), but also represents the uncertainty of the learned model via moment-matching [26]. To do this, we evaluate the FNNs with the input (x_t, u_t) and randomly sampled dropout masks $(Z_f^1, Z_g^1) \ldots, (Z_f^{N_z}, Z_g^{N_z})$, and obtain the outputs $\hat{x}_{t+1}^1, \ldots, \hat{x}_{t+1}^{N_z}$. The model uncertainty at (x_t, u_t) is represented by the covariance Σ_t of $\hat{x}_{t+1}^1, \ldots, \hat{x}_{t+1}^{N_z}$. The FNNs can be easily transformed into a deterministic form by applying masks Z_f^{det} and Z_g^{det} that activate all nodes and scaling the outputs of dropout layers by a factor of $1 - p_d$.

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 26,2023 at 05:34:14 UTC from IEEE Xplore. Restrictions apply.

To learn the model, we need to execute the system and record the system transitions. The model learning algorithm is summarized in Alg. 1 (values in brackets are for initial dataset generation). At the beginning, we sample N_0 random initial states $\{x_0^1, \ldots, x_0^{N_0}\}$ from the probability distribution p (step 2). From each initial state, we apply random controls $u_t \sim \text{Uniform}(\mathcal{U})$ and collect all the system transitions $\big((x_t^i, u_t^i), x_{t+1}^i\big)$ with $i = 1, \ldots, N_0$ and $t = 0, \ldots, T_0$ to create the initial dataset $\mathbf{D}$, where $T_0 \geq T$ is the time horizon to apply random controls (step 5). Then two FNNs are trained on $\mathbf{D}$ simultaneously (step 11) to minimize the loss C:

$$C = \sum_{\mathbf{D}} \sum_{t=0}^{T-1} \|x_{t+1} - \hat{x}_{t+1}\|^2. \tag{7}$$

After training, the FNN parameters are denoted by W_f^*, W_g^*.

Similarly to generating the initial dataset, each time after the control policy is improved, we sample N_s initial states (step 2), starting from which the safe controls modified from the control policy π using CBFs (discussed later) are applied until arriving at the time horizon T (steps 7-8). We add all the system transition data to the dataset $\mathbf{D}$. Then the FNNs are retrained on the new dataset to minimize the loss C (step 11). In practice, during all the system executions, if one of the values of $b_1, \ldots, b_N$ is less than 0, which means that the CBFs fail due to model error, we stop the system and terminate the trial to protect the system (steps 9-10).

As mentioned earlier, after the initial training of the model, we use CBFs with the learned model to modify the control policy. Here we transform the FNN model (6) into the form:

$$\hat{x}_{t+1} = \mathcal{F}(x_t; W_f, Z_f^{det}) + \mathcal{G}(x_t; W_g, Z_g^{det})u_t + \delta_t, \tag{8}$$

where δ_t is a zero-mean random vector representing the uncertainty of the learned model. Its covariance Σ_t is estimated via moment-matching. For each CBF b_j, $j = 1, \ldots, N$, the uncertainty of x_{t+1} is passed to $b_j(x_{t+1})$ as an error $\epsilon_{t,j}$:

$$b_j(x_{t+1}) = b_j(\hat{x}_{t+1}) + \epsilon_{t,j}. \tag{9}$$

We use the first order derivative to approximate the error:

$$\epsilon_{t,j} = \frac{\partial b_j}{\partial x}\Big|_{x=\mathcal{F}(x_t;W_f,Z_f^{det})+\mathcal{G}(x_t;W_g,Z_g^{det})u_t} \cdot \delta_t. \tag{10}$$

For brevity, in what follows, we omit the point that the derivative is evaluated at. The error $\epsilon_{t,j}$ is a zero-mean random variable with variance $\sigma_{t,j}^2$, which is given by:

$$\sigma_{t,j}^2 = \Big(\frac{\partial b_j}{\partial x}\Big)\Sigma_t\Big(\frac{\partial b_j}{\partial x}\Big)^\top. \tag{11}$$

By choosing a lower limit $-\lambda\sigma_{t,j}$ of the error $\epsilon_{t,j}$ in (9) to compute $b_j(x_{t+1})$, $\lambda > 0$, we can guarantee the safety for all $\epsilon_{t,j} \geq -\lambda\sigma_{t,j}$. At time t and state x_t, we obtain the modified control u_t^s that is safe for errors above $-\lambda\sigma_{t,j}$:

$$\begin{aligned} u_t^s = &\arg\min_{u_t} \ \|u_t - \pi(x_{0:t}, W)\|^2 \\ \text{s.t.} \quad & b_j\big(\mathcal{F}(x_t; W_f, Z_f^{det}) + \mathcal{G}(x_t; W_g, Z_g^{det})u_t\big) \\ & + (\alpha - 1)b_j(x_t) \geq \lambda\sigma_{t,j}, \quad j = 1, \ldots, N \end{aligned} \tag{12}$$

where $\alpha \in [0, 1]$. In (12), $\sigma_{t,j}$ is supposed to be a function of u_t (the derivative $\partial b_j/\partial x$ depends on u_t), but we assume a constant $\sigma_{t,j}$ that is evaluated at $u_t = \pi(x_{0:t}, W)$ to make (12) simpler. Since b_j is affine or quadratic, (12) is a (quadratically constrained) quadratic program, which can be solved efficiently using solvers such as Gurobi [33].

Remark 1. The safe control u_t^s can only improve safety, rather than fully guarantee it, because: (1) estimate bias exists; (2) the error $\epsilon_{t,j}$ in $b_j(x_{t+1})$ is approximated by the first order derivative and with a fixed control $\pi(x_{0:t}, W)$; (3) the error may exceed the lower limit. However, according to our simulation results (Section V), this method can largely reduce the chance of the system reaching unsafe regions.

Algorithm 1: System model learning

```
Input: A control policy $\pi$, a dataset $\mathbf{D}$
Output: Optimal FNN parameters $W_f^*$, $W_g^*$
1  for $i \in 1, \ldots, N_s(N_0)$ do
2      Sample initial state $x_0^i$;
3      for $t = 1, 2, \ldots, T(T_0)$ do
4          if No policy and model was trained then
5              Apply $u_t^i \sim \text{Uniform}(\mathcal{U})$ to the system
                and add the data $\big((x_t^i, u_t^i), x_{t+1}^i\big)$ to $\mathbf{D}$;
6          else
7              Adjust $\pi(x_{0:t}^i, W)$ to obtain $u_t^{s,i}$ by (12);
8              Apply $u_t^{s,i}$ to the system and add the data
                $\big((x_t^i, u_t^{s,i}), x_{t+1}^i\big)$ to $\mathbf{D}$;
9          if $\min(b1, \ldots, b_N) \leq 0$ then
10             Break
11 Train the FNN on $\mathbf{D}$, return $W_0^*$;
```

B. Control Policy Improvement

Now we can solve (5) in Pb. 1. We substitute the unknown system dynamics with the learned one from Sec. IV-A. Also, we estimate the expected robustness over initial conditions by sampling M initial states $x_0^1, \ldots, x_0^M$ and averaging the resulting robustness values. For different initial states, different dropout masks $(Z_f^1, Z_g^1) \ldots, (Z_f^M, Z_g^M)$ in the learned model are sampled to estimate the trajectories (no CBFs applied). The average over initial states is also an average over system models, hence it can reduce model-bias [24]. For brevity, let $\hat{f}^i(x_t) = \mathcal{F}(x_t; W_f, Z_f^i)$ and $\hat{g}^i(x_t) = \mathcal{G}(x_t; W_g, Z_g^i)$ be the sampled system model. Now the problem becomes:

$$\begin{aligned} W^* = &\arg\max_W \frac{1}{M}\sum_{i=1}^M \rho(\varphi, x_{0:T}^i) \\ \text{s.t.} \quad & x_{t+1}^i = \hat{f}^i(x_t^i) + \hat{g}^i(x_t^i)\pi(x_{0:t}^i, W), \\ & t = 0, 1, \ldots, T-1, \ i = 1, \ldots, M. \end{aligned} \tag{13}$$

We solve (13) by substituting the constraints into the objective function and solving the unconstrained optimization problem. We use gradient ascent to iteratively update the RNN parameters W. At each iteration step, we randomly resample a new set of M initial states and M dropout masks to evaluate the objective function and its gradient. Doing so gives us an unbiased estimator of the objective [25]. We then implement the optimization using the stochastic optimizer

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 26,2023 at 05:34:14 UTC from IEEE Xplore. Restrictions apply.

Adam [34]. The optimization terminates after convergence. The policy improvement process is shown in Alg. 2.

```
Algorithm 2: Control policy improvement
  Input: Learned model $\hat{f}$, $\hat{g}$, an STL formula $\varphi$
  Output: Optimal policy parameters $W^*$
1 repeat
2     Sample initial states $\{x_0^1, \dots, x_0^M\}$;
3     Sample system models $\{(\hat{f}^1, \hat{g}^1), \dots, (\hat{f}^M, \hat{g}^M)\}$;
4     Compute the objective in (13);
5     Back-propagate the gradient $\Delta W$ of (13);
6     Update $W$ with $\Delta W$ using the Adam optimizer;
7 until Convergence; return $W^*$;
```

Remark 2. All the derivatives involving the FNN system model and the RNN controller can be computed using the auto-differentiation tools designed for neural networks (e.g., PyTorch [35]). When computing the STL robustness ρ, we applied the method STLCG [36] which uses a computation graph similar to a neural network. So the auto-differentiation tools can also be applied. Hence, the gradient of the objective function $\frac{\partial \rho}{\partial W}$ in (13) can be accessed easily and analytically.

C. Model-based policy search

We alternately use Alg. 1 and Alg. 2 to train the model and the policy as in Fig. 1. We call the completion of one model learning and one policy improvement a *cycle*. The algorithm will terminate and return the final control policy with parameter W^* when the policy converges.

Remark 3. Although the objective of our algorithm is to maximize the expected STL robustness over trajectories starting from random initial states, there is no guarantee that the learned policy can satisfy the STL from all initial states. Moreover, using CBFs might change the original control given by the RNN, which might decrease the robustness or even violate the STL specification. This also causes that in model learning phase the robot explores different regions from those induced by the learned policy. Hence, the model may not be properly learned in some regions explored by the learned policy. However, by also including the safety constraints into the STL formula, the RNN controller is aware of these constraints, which avoids dramatic changes when using CBFs. This mitigates the above problems. It is also possible to include CBFs in the policy learning phase by differentiating through (12) using technique in [37]. We will investigate this in future work. The simulation and experiment results show that our approach can reach a very high success rate of satisfying the STL specification.

Remark 4. The advantages of using neural networks (RNN and FNN) includes: (1) fast execution; (2) good scalability; (3) easy propagation of gradients. These benefits will be shown in Sec. V. An additional advantage of the RNN is the ability to satisfy general full STL formulae. Another choice is using a memoryless controller but augmenting the system state [15], [16], which makes the state space too large for long horizon control and is restricted to STL fractions.

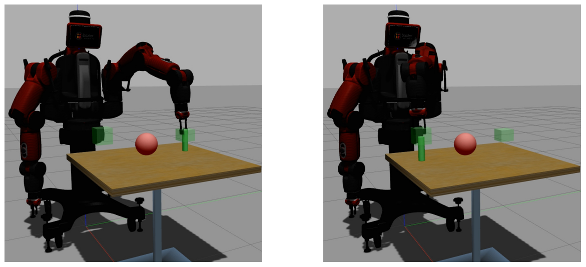
(a) Pick (b) Place

Fig. 2: The Gazebo simulation environment. The green cylinder is the object to be picked, and the red ball is the obstacle. The red, green, blue lines are the x, y, z axes respectively. Transparent boxes are R_{pick} and R_{place}. Video is at https://youtu.be/UmuJdVcxY14.

V. SIMULATIONS

We consider a pick-and-place task for a Baxter robot in Gazebo [38] simulation environment (shown in Fig. 2). The algorithms were implemented in Python. The NNs were implemented using Pytorch [35]. We used a computer with a 3.50GHz Core i7 CPU and 16GB RAM for the simulation.

Baxter is a 7 DOF dual arm robot. We only use its left arm in this case study. The system state $x_t \in \mathbb{R}^{14}$ is defined as the concatenation of joint angles $\theta_t \in \mathbb{R}^7$ and joint velocities $\dot{\theta}_t \in \mathbb{R}^7$. We use the torques applied to the joints as the control input $u_t \in [-8, 8]^4 \times [-3, 3]^3 \subseteq \mathbb{R}^7$. Note that u_t is applied in addition to the known gravity compensation torques. We control the system with a time interval $\Delta t = 1/6s$. The initial joint angles are $[-0.6, -0.75, 0, 0, 75, 0, 1.46, 0]$ rad with a uniformly sampled disturbance $d \in [-0.1, 0.1]^7$.

Our specification is defined over the end effector pose $[\mathbf{p}_t, \mathbf{q}_t]$ where $\mathbf{p}_t = [p_x, p_y, p_z]$ is the 3D position and $\mathbf{q}_t = [q_w, q_x, q_y, q_z]$ is the quaternion representing the orientation. Assume the forward kinematics $[\mathbf{p}_t, \mathbf{q}_t] = K(\theta_t)$ are known and the motion primitives for *open* and *close* of the gripper are already available. Both *open* and *close* can be finished within $0.5s$. Then the STL task is: stay in the *pick* region (R_{pick}) for $0.5s$ between $[0s, 3s]$, and stay in the *place* region (R_{place}) for $0.5s$ between $[3s, 6s]$. We also require the rotation angle of the end-effector from the vertical pose be always less than 30° ($\mathbf{q} \in V_{30}$) so that the gripper is downward when picking up the object. The safety constraints are to always avoid collision with a spherical obstacle ($\mathbf{p} \notin Obs$), its body ($p_x \geq x_{body}$), and the table ($p_z \geq h_{tb}$). The task can be formulated as φ_1 with $hrz(\varphi_1) = 36$:

$$
\begin{aligned}
\varphi_1 =& \big(\mathbf{F}_{[0, \frac{2.5}{\Delta t}]}(\mathbf{G}_{[0, \frac{0.5}{\Delta t}]}\mathbf{p} \in R_{pick})\big) \\
& \wedge \big(\mathbf{F}_{[\frac{3}{\Delta t}, \frac{5.5}{\Delta t}]}(\mathbf{G}_{[0, \frac{0.5}{\Delta t}]}\mathbf{p} \in R_{place})\big) \wedge (\mathbf{G}_{[0, \frac{6}{\Delta t}]}\mathbf{q} \in V_{30}) \\
& \wedge (\mathbf{G}_{[0, \frac{6}{\Delta t}]} p_x > x_{body}) \wedge (\mathbf{G}_{[0, \frac{6}{\Delta t}]} p_z > h_{tb}) \\
& \wedge (\mathbf{G}_{[0, \frac{6}{\Delta t}]}\mathbf{p} \notin Obs)
\end{aligned}
$$

Although the CBFs representing the safety constraints are affine or quadratic with respect to the end-effector position, they are nonquadratic with respect to the system state due to the complexity of the forward kinematics. Hence, we linearize the forward kinematics at each time step, which makes the CBFs meet the affine or quadratic assumption.

Let $N_0 = 100$, $N_s = 20$, $M = 200$, $N_z = 20$, $\lambda = 3$, $\alpha = 0.8$. The RNN applies an LSTM [39] with 3 hidden layers

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 26,2023 at 05:34:14 UTC from IEEE Xplore. Restrictions apply.

TABLE I: Success Rate and Average Robustness for φ_1

	w/ CBF			w/o CBF		
	γ	β	$\bar{\rho}$	γ	β	$\bar{\rho}$
Cycle 1	0%	10%	-0.4807	0%	24%	-0.7224
Cycle 2	2%	0%	-0.0973	14%	5%	-0.0880
Cycle 3	14%	0%	-0.0235	22%	0%	-0.0071
Cycle 4	83%	0%	0.0169	87%	0%	0.0192
Cycle 5	100%	0%	0.0401	100%	0%	0.0419

each with 64 nodes. Both FNNs have 2 hidden layers each with 128 nodes and the dropout probability $p_d = 0.05$. After 5 cycles of training which takes 50min, the robot can finish the task successfully as shown in Fig. 2. This is much faster than our previous work [11] in which training the controller for a much easier task takes 2 hours. We test the policy (trained with CBFs) after each cycle on 100 random initial states. The success rate γ (getting positive robustness), unsafe rate β (violating safety constraints) and average robustness $\bar{\rho}$, with or without CBFs, are shown in Table I.

After 5 cycles of training the success rate reaches 100%. From Table I, we can see that after updating the model, the policy can be improved to get a higher robustness and a higher success rate. The unsafe rate β is largely reduced by using CBFs. Note that sometimes the robustness and success rate without CBFs are higher. This is because CBFs can change the policy. The system runs for a total of 180 times, which is a relatively small number. In [15] a model-free Q-learning approach runs 2000 episodes to learn the policy for a much easier STL task and a much simpler system. After training, execution of the RNN controller and solving (12) to obtain the safe control take $0.0004s$ and $0.03s$ (in average) respectively. When the discrete time interval is large enough compared with the computation time for the safe control, the controller can be implemented in real time. The results show that the algorithm proposed in this paper is capable to control a robot arm directly from joint torques to end-effector pose without knowing the complex system dynamics. The planning horizon, system dimension and number of predicates would affect the training time linearly, while they have little effect on the execution time after training, hence our approach has a good scalabilty.

VI. Experiment

We consider a system containing 3 iRobot Create2 ground robots in an indoor motion capture environment (as shown in Fig. 3). The position and orientation of the robots can be obtained using the Optitrack motion capture system. Let $x^k = [p_x^k \ p_y^k \ \sin\theta^k \ \cos\theta^k]^\top$ and $u^k = [v^k \ \omega^k]^\top$ be the state and the control of the k^{th} robot, where p_x^k, p_y^k, θ^k give the position and orientation and $u^k = [v^k \ \omega^k]^\top$ captures the speed and angular speed, with $v^k \in [0, 0.75]$ and $\omega^k \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, $k \in \{1, 2, 3\}$. We normalize the terms corresponding to $\sin\theta$ and $\cos\theta$ in the output of the learned model after each step. The state and control of the whole system are the concatenations of the states and controls of the 3 robots. Hence the system state $x \in \mathbb{R}^{12}$ and the control $u \in \mathbb{R}^6$. Controls are sent to the robots in a rate of $3HZ$. The algorithm is implemented on a computer with a

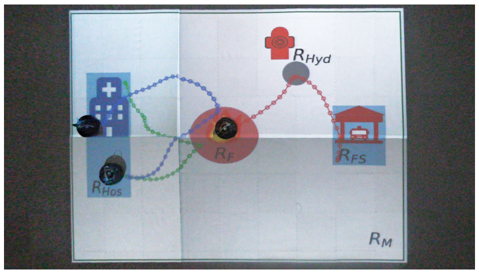

Fig. 3: An execution of the system with the learned policy. The trajectories of the robots are projected on to the ground.

4.20GHz Core i7 CPU and 32GB RAM. All these platforms communicate using the Robot Operating System (ROS) [40].

Consider a fire-fighting scenario as shown in Fig. 3, where two ambulances ($k = 1, 2$) randomly located in the hospital R_{Hos} are supposed to go to the fire R_F in $8s$, and then return to the hospital in $[8s, 15s]$, while a fire engine ($k = 3$) randomly located in a fire station R_{FS} should go to the fire in $10s$ and stay there in $[10s, 15s]$, and reaches a Hydrant R_{Hyd} before that. The safety constraints are (1) never go out of bounds R_M; (2) the distances between the robots are always larger than d. Let $p^k = [p_x^k \ p_y^k]$. The specification can be formulated as φ_2 with $hrz(\varphi_2) = 15/\Delta t = 45$:

$$\begin{aligned}\varphi_2 = \wedge_{k=1}^{2} \big(&\mathbf{F}_{[0, \frac{8}{\Delta t}]}(p^k \in R_F) \wedge \mathbf{F}_{[\frac{8}{\Delta t}, \frac{15}{\Delta t}]}(p^k \in R_{Hos})\big) \\ &\wedge \mathbf{F}_{[0, \frac{10}{\Delta t}]}(p^3 \in R_{Hyd}) \wedge \mathbf{G}_{[\frac{10}{\Delta t}, \frac{15}{\Delta t}]}(p^3 \in R_F) \\ &\wedge_{i,j \in \{1,2,3\}, i \neq j} \mathbf{G}_{[0, \frac{15}{\Delta t}]}(\|p^i - p^j\|^2 \geq d^2) \\ &\wedge_{k=1}^{3} \mathbf{G}_{[0, \frac{15}{\Delta t}]}(p^k \in R_M).\end{aligned}$$

Note that the safety predicates (and the corresponding CBFs) are all affine and quadratic functions.

Let $N_0 = 50$, $\alpha = 0.5$. The other settings are same with the simulation. When computing the norm in (12) we assign a weight of 0.03 to the angular speed in order to encourage the robot to turn instead of slowing down when approaching an obstacle. The computation time for the RNN policy and the safe control ($0.0004s$ and $0.01s$ respectively) is negligible compared with $\Delta t = 1/3s$, which enables real time control. The robots can learn a good policy that satisfies the specification after 4 cycles of training. In this process, we collected 2372 data points to train the system model, with the system running for about 13 minutes (not counting the time it takes the robots to return to the initial states). The total training time is about 20 minutes. Fig. 3 shows an execution of the system with the learned policy. We tested the policy 20 times and the success rate was 100%.

VII. Conclusion

In this paper, we proposed a model-based policy search approach to maximize STL robustness. The system model and the control policy are learned alternately. The policy is implemented as an RNN, which can deal with the history-dependence of STL. Simulation and experiment results show that our approach can learn the policy within relatively few system executions and achieves high success rate. Safety can be improved by using CBFs. After training, the RNN policy can be implemented in real time.

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 26,2023 at 05:34:14 UTC from IEEE Xplore. Restrictions apply.

References

[1] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] I. Haghighi, N. Mehdipour, E. Bartocci, and C. Belta, "Control from signal temporal logic specifications with smooth cumulative quantitative semantics," in *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, 2019, pp. 4361–4366.

[8] P. Varnai and D. V. Dimarogonas, "On robustness metrics for learning stl tasks," in *2020 American Control Conference (ACC)*. IEEE, 2020, pp. 5394–5399.

[9] Y. Gilpin, V. Kurtz, and H. Lin, "A smooth robustness measure of signal temporal logic for symbolic control," *IEEE Control Systems Letters*, vol. 5, no. 1, pp. 241–246, 2020.

[10] S. Yaghoubi and G. Fainekos, "Worst-case satisfaction of stl specifications using feedforward neural network controllers: a lagrange multipliers approach," in *2020 Information Theory and Applications Workshop (ITA)*. IEEE, 2020, pp. 1–20.

[11] W. Liu, N. Mehdipour, and C. Belta, "Recurrent neural network controllers for signal temporal logic specifications subject to safety constraints," *IEEE Control Systems Letters*, 2021.

[12] K. Leung and M. Pavone, "Semi-supervised trajectory-feedback controller synthesis for signal temporal logic specifications," *arXiv preprint arXiv:2202.01997*, 2022.

[13] M. Ma, J. Gao, L. Feng, and J. Stankovic, "Stlnet: Signal temporal logic enforced multivariate recurrent neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 14 604–14 614, 2020.

[14] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," *arXiv preprint arXiv:1606.06565*, 2016.

[15] 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.

[16] H. Venkataraman, D. Aksaray, and P. Seiler, "Tractable reinforcement learning of signal temporal logic objectives," in *Learning for Dynamics and Control*. PMLR, 2020, pp. 308–317.

[17] A. Balakrishnan and J. V. Deshmukh, "Structured reward shaping using signal temporal logic specifications," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 3481–3486.

[18] J. Ikemoto and T. Ushio, "Deep reinforcement learning based networked control with network delays for signal temporal logic specifications," *arXiv preprint arXiv:2108.01317*, 2021.

[19] X. Li, Y. Ma, and C. Belta, "A policy search method for temporal logic specified reinforcement learning tasks," in *2018 Annual American Control Conference (ACC)*. IEEE, 2018, pp. 240–245.

[20] P. Varnai and D. V. Dimarogonas, "Prescribed performance control guided policy improvement for satisfying signal temporal logic tasks," in *2019 American Control Conference (ACC)*. IEEE, 2019, pp. 286–291.

[21] X. Li, Z. Serlin, G. Yang, and C. Belta, "A formal methods approach to interpretable reinforcement learning for robotic planning," *Science Robotics*, vol. 4, no. 37, 2019.

[22] C. Wang, Y. Li, S. L. Smith, and J. Liu, "Continuous motion planning with temporal logic specifications using deep neural networks," *arXiv preprint arXiv:2004.02610*, 2020.

[23] M. Cai, M. Hasanbeig, S. Xiao, A. Abate, and Z. Kan, "Modular deep reinforcement learning for continuous motion planning with temporal logic," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7973–7980, 2021.

[24] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*. Citeseer, 2011, pp. 465–472.

[25] Y. Gal, R. McAllister, and C. E. Rasmussen, "Improving pilco with bayesian neural network dynamics models," in *Data-Efficient Machine Learning workshop, ICML*, vol. 4, no. 34, 2016, p. 25.

[26] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*. PMLR, 2016, pp. 1050–1059.

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

[28] M. Cai and C.-I. Vasile, "Safety-critical learning of robot control with temporal logic specifications," *arXiv preprint arXiv:2109.02791*, 2021.

[29] C. K. Verginis, Z. Xu, and U. Topcu, "Non-parametric neuro-adaptive control subject to task specifications," *arXiv preprint arXiv:2106.13498*, 2021.

[30] A. Dokhanchi, B. Hoxha, and G. Fainekos, "On-line monitoring for temporal logic robustness," in *International Conference on Runtime Verification*. Springer, 2014, pp. 231–246.

[31] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, 2016, vol. 1, no. 2.

[32] 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*, 2017.

[33] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2021. [Online]. Available: https://www.gurobi.com

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

[35] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[36] K. Leung, N. Aréchiga, and M. Pavone, "Back-propagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods," in *International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2020, pp. 432–449.

[37] 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.

[38] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. IEEE, 2004, pp. 2149–2154.

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

[40] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 26,2023 at 05:34:14 UTC from IEEE Xplore. Restrictions apply.