Learning From Demonstrations Using Signal Temporal Logic in Stochastic and Continuous Domains

Aniruddh Gopinath Puranic, Jyotirmoy V. Deshmukh, and Stefanos Nikolaidis

***Abstract*—Learning control policies that are safe, robust and interpretable are prominent challenges in developing robotic systems. Learning-from-demonstrations with formal logic is an arising paradigm in reinforcement learning to estimate rewards and extract robot control policies that seek to overcome these challenges. In this approach, we assume that mission-level specifications for the robotic system are expressed in a suitable temporal logic such as Signal Temporal Logic (STL). The main idea is to automatically infer rewards from user demonstrations (that could be suboptimal or incomplete) by evaluating and ranking them w.r.t. the given STL specifications. In contrast to existing work that focuses on deterministic environments and discrete state spaces, in this letter, we propose significant extensions that tackle stochastic environments and continuous state spaces.**

***Index Terms*—Learning from demonstration, probabilistic inference, formal methods in robotics and automation.**

I. Introduction

LEARNING-FROM-DEMONSTRATIONS (LfD) is an exciting algorithmic paradigm in which control policies for robots can be extracted from human demonstrations [1], [2]. LfD can also address the issue of manually designing rewards or cost functions for robotic missions. The main areas of research in LfD are: behavior cloning via supervised learning [3] and inverse reinforcement learning (IRL) [4], [5]. Behavior cloning uses supervised learning to model/mimic the actions of a teacher by mapping states to actions. In IRL, a reward function is derived from a set of human demonstrations for reinforcement learning (RL) tasks. Apprenticeship learning (AL) aims to extract control policies from rewards inferred using IRL [6]. Designing rewards for RL is a non-trivial task and typically requires domain expert knowledge [7]. Learning noisy or incorrect reward functions can lead to the agent performing unintended or unsafe behaviors [8].

Practically, there are some limitations to the LfD paradigm: (i) a demonstration is seldom optimal and is always susceptible to noises or disturbances due to motions of the user or uncertainties in the environment. The control policy inferred from such a demonstration may thus perform unsafe or undesirable actions when the initial configuration is slightly perturbed [9]. Thus, LfD lacks robustness; (ii) demonstrations are not always equal: some are a better indicator of the desired behavior than others, and the expertise of the demonstrator determines the quality of the demonstration [7]; and (iii) safety conditions for the robot cannot be explicitly specified by demonstrations, and safely providing a demonstration requires skill [7], [9].

To address some of these challenges, previous work [10] proposed a framework to integrate formal logic with LfD. Demonstrations implicitly convey intended behaviors of the user and robot, i.e., demonstrations can be interpreted as partial specifications for the robot behavior, as well as a representation of the partial (possibly sub-optimal or incorrect) control policy. On the other hand, temporal logic specifications represent high-level mission objectives for the robot, but do not indicate *how* to achieve the objectives. They can also encode information about the environment such as rules to be obeyed by the agent. Our proposed approach seeks to use both, the user demonstrations and the specifications to learn rewards from which a control policy can be extracted via RL. In this framework, the user explicitly provides demonstrations and high-level specifications in a mathematically precise and unambiguous formal language - *Signal Temporal Logic* (STL). An STL formula evaluates a temporal behavior of a system (in our case, a demonstration or agent's policy) and provides a quantity that indicates how well this system satisfied the formula via a fitness quantity called $robustness$ [11], [12] that is then used to define rewards. In general, the STL specifications tell the agent "what to do," while the rewards obtained by evaluating the quality of demonstrations tell the agent "how to do it". STL does not define the entire reward function, but only some parts or sub-spaces of it and hence our framework uses demonstrations for learning rewards. Existing work focuses on deterministic agents and environments, however, in the real-world there may be uncertainties in the agent's motion/actions and environment. In this letter, we significantly extend the existing LfD-STL [10] framework to stochastic environments. Existing work also assumes discrete state spaces, which allows the use of tabular reward functions and tabular RL methods. In this letter, we extend our approach to continuous state spaces which necessitates continuous approximations for reward functions and the concomitant continuous-space RL algorithms.

Similar to [10], we use STL specifications (i) to evaluate and automatically rank demonstrations based on their $robustness$, and (ii) to infer rewards (considering environment stochasticity)

Manuscript received February 23, 2021; accepted June 1, 2021. Date of publication June 25, 2021; date of current version July 13, 2021. This letter was recommended for publication by Associate Editor L. Peternel and Editor D. Kulic upon evaluation of the reviewers' comments. This work was supported by the National Science Foundation under the CAREER Award SHF-2048094, Awards CCF-1837131 and CNS-1932620, and Toyota R&D. *(Corresponding author: Aniruddh Gopinath Puranic.)*

The authors are with the Computer Science Department, University of Southern California, Los Angeles, CA 90007 USA (e-mail: puranic@usc.edu; jdeshmuk@usc.edu; nikolaid@usc.edu).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2021.3092676, provided by the authors.

Digital Object Identifier 10.1109/LRA.2021.3092676

to be used in an RL procedure used to train the control policy. While we also use the directed acyclic graph-based ranked rewards from [10], we propose a new technique to handle stochasticity and continuous state spaces. The intuition is to create a "tube" around the trajectory represented by a demonstration. This tube represents the possible states the demonstrator could have been in, i.e., we devise a mechanism to propagate the rewards to nearby states. In this letter, our main goal is to learn a reward function from which an RL policy can be derived. The main contributions of our work are as follows:

1) We propose a novel mathematical way of inferring temporal/non-Markovian rewards for a system under probabilistic transition dynamics without the necessity for optimal or perfect demonstrations. We develop a reward approximation and prediction method applicable for continuous and higher dimensional spaces. These rewards can be used with appropriate RL methods such as policy gradients or actor-critic algorithms.
2) We show that this method can learn from a handful of demonstrations even in the presence of uncertainties in the environment.
3) We validate our method on several discrete-world environments and also on a custom 2D driving scenario.

II. Background and Problem Definition

A. Background

Definition 1 (Environment): An environment is a tuple $E = (S, \mathcal{A})$ consisting of the set of states S defined over $\mathbb{R}^n$ and set of actions $\mathcal{A}$.

Definition 2 (Demonstration): A finite sequence of state-action pairs is called a demonstration. Formally, a demonstration d of length $L \in \mathbb{N}$ is $d = \langle (s_1, a_1), (s_2, a_2), \ldots, (s_L, a_L) \rangle$, where $s_i \in S$ and $a_i \in \mathcal{A}$. That is, d is an element of $(S \times \mathcal{A})^L$.

Signal Temporal Logic (STL) is a real-time logic, generally interpreted over a dense-time domain for signals whose values are from a continuous metric space (such as $\mathbb{R}^n$). The basic primitive in STL is a *signal predicate* μ that is a formula of the form $f(\mathbf{x}(t)) > 0$, where $\mathbf{x}(t)$ is the tuple $(state, action)$ of the demonstration $\mathbf{x}$ at time t, and f maps the signal domain $\mathcal{D} = (S \times \mathcal{A})$ to $\mathbb{R}$. STL formulas are then defined recursively using Boolean combinations of sub-formulas, or by applying an interval-restricted temporal operator to a sub-formula. The syntax of STL is formally defined as follows: $\varphi ::= \mu \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathrm{G}_I\varphi \mid \mathrm{F}_I\varphi \mid \varphi \mathrm{U}_I \varphi$. Here, $I = [a, b]$ denotes an arbitrary time-interval, where $a, b \in \mathbb{R}^{\geq 0}$. The semantics of STL are defined over a discrete-time signal x defined over some time-domain $\mathbb{T}$. The Boolean satisfaction of a signal predicate is simply *True* ($\top$) if the predicate is satisfied and *False* ($\bot$) if it is not, the semantics for the propositional logic operators $\neg, \wedge$ (and thus $\vee, \rightarrow$) follow the obvious semantics. The following behaviors are represented by the temporal operators:

- At time t, if $G_I(\varphi)$ holds then φ holds $\forall t'$ in $t + I$.
- At time t, if $F_I(\varphi)$ holds then φ holds at some $t' \in t + I$.
- At time t, if $\varphi \mathrm{U}_I \psi$ holds then ψ holds at some time $t' \in t + I$, and $\forall t'' \in [t, t')$, φ holds.

Definition 3 (Quantitative Semantics for Signal Temporal Logic): Given an algebraic structure $(\oplus, \otimes, \top, \bot)$, we define the quantitative semantics for an arbitrary signal x against an STL formula φ at time t as in Table I .

TABLE I
Quantitative Semantics of STL

φ	$\rho(\varphi, \mathbf{x}, t)$
$true/false$	$\top/\bot$
μ	$f(\mathbf{x}(t))$
$\neg\varphi$	$-\rho(\varphi, \mathbf{x}, t)$
$\varphi_1 \wedge \varphi_2$	$\otimes(\rho(\varphi_1, \mathbf{x}, t), \rho(\varphi_2, \mathbf{x}, t))$
$\varphi_1 \vee \varphi_2$	$\oplus(\rho(\varphi_1, \mathbf{x}, t), \rho(\varphi_2, \mathbf{x}, t))$
$\mathbf{G}_I(\varphi)$	$\otimes_{\tau \in t+I}(\rho(\varphi, \mathbf{x}, \tau))$
$\mathbf{F}_I(\varphi)$	$\oplus_{\tau \in t+I}(\rho(\varphi, \mathbf{x}, \tau))$
$\varphi \mathbf{U}_I \psi$	$\oplus_{\tau_1 \in t+I}(\otimes(\rho(\psi, \mathbf{x}, \tau_1), \otimes_{\tau_2 \in [t, \tau_1)}(\rho(\varphi, \mathbf{x}, \tau_2))))$

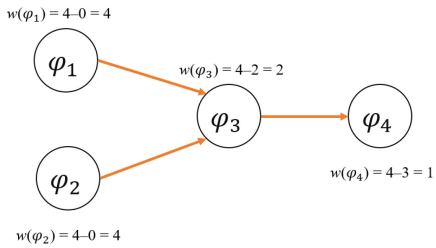


Fig. 1. Weights on nodes (specifications) in a DAG.

A signal satisfies an STL formula φ if it is satisfied at time $t = 0$. Intuitively, the quantitative semantics of STL represent the numerical distance of "how far" a signal is away from the signal predicate. For a given requirement φ, a demonstration or policy d that satisfies it is represented as $d \models \varphi$ and one that does not, is represented as $d \not\models \varphi$. In addition to the Boolean satisfaction semantics for STL, various researchers have proposed quantitative semantics for STL, [12], [13] that compute the degree of satisfaction (or *robust satisfaction values*) of STL properties by traces generated by a system. In this work, we use the following interpretations of the STL quantitative semantics: $\top = +\infty$, $\bot = -\infty$, and $\oplus = \max$, and $\otimes = \min$, as per the original definitions of robust satisfaction proposed in [12], [14].

Similar to [10], we make use of the two classes of temporal logic requirements: (i) hard requirements Φ_H which are particular properties of a system that are required to be invariant, such as requiring the system to follow the workspace rules or operate within its constraints at all times. These properties can be regarded as safety requirements for the system and they typically are of the form $\mathrm{G}(\varphi)$; and (ii) soft requirements Φ_S, that are generally concerned with the optimality of a system such as performance, efficiency, etc. Hard requirements always need to be satisfied by a system before being able to satisfy the soft requirements. These requirements are arranged using a directed acyclic graph (DAG) $G = (V, X)$, where each node in V represents a specification. Directed edges X in G correspond to the relative order/preferences of specifications and the weight on each node indicates the relative priority of its corresponding specification by analyzing number of nodes it is dependent on, i.e., for a node u in the DAG, its corresponding weight is given by $w(u) = |V| - |ancestors(u)|$ as shown in Fig. 1. The ancestors of a node u are the set of all nodes in V that have a path to u, i.e., u is dependent on its ancestors.

Demonstration Types: Based on the 2 classes of logic requirements, we obtain 2 types of demonstrations: (i) a demonstration is labeled *good* if it satisfies the specifications $\Phi = \Phi_H \cup \Phi_S$; (ii) a demonstration is considered *bad* if it violates any hard specification of Φ_H. A bad demonstration d consists of at least one state or state-action pair that violates a hard specification ψ, i.e., $s_{bad} = \{s_j \mid (s_j, a_j) \not\models \psi\}$.

DAG-based Rewards: As per the arrangement of specifications $\Phi = \Phi_H \cup \Phi_S$ in a DAG, we obtain the weight vector $\mathbf{w_\Phi} = [w(\varphi_1), \ldots, w(\varphi_{|\Phi|})]^T$; where, hard requirements are given by $\Phi_H = \{\varphi_1, \varphi_2, \ldots, \varphi_p\}$ and soft requirements are given by $\Phi_S = \{\varphi_{p+1}, \varphi_{p+2}, \ldots, \varphi_q\}$. For each demonstration d, we also obtain the corresponding robustness vector, $\vec{\boldsymbol{\rho_d}} = [\rho_1, \ldots, \rho_{|\Phi|}]^T$, where ρ_i is the robustness of that demonstration w.r.t. φ_i. Finally, the DAG-based robustness for d is given by the weighted-sum [1] $\hat{\rho}_d = \mathbf{w_\Phi}^T \cdot \vec{\boldsymbol{\rho_d}}$.

Definition 4 (Markov Decision Process (MDP)): An MDP is given by a tuple $M = (S, \mathcal{A}, T, R)$ where S is the state space of the system; $\mathcal{A}$ is the set of actions that can be performed on the system; $\mathcal{A} = \{a_1, a_2, \ldots, a_N\}$; T is the transition function, where $T(s, a, s') = Pr(s' \mid s, a)$; R is a reward function that typically maps either some $s \in S$ or some transition $\delta \in S \times \mathcal{A} \times S$ to $\mathbb{R}$.

In RL, the goal of the learning algorithm is to find a policy $\pi : S \rightarrow \mathcal{A}$ that maximizes the total (discounted) reward from performing actions on a MDP, i.e., the objective is to maximize $\sum_{t=0}^{\infty} \gamma^t r_t$, where r_t is the output of the reward function R for the sample at t and γ is the discount factor. In this letter, we assume full observation of the state space for agents operating in known environments.

B. Problem Formulation

We seek to infer rewards from user demonstrations and STL specifications while considering the environment's uncertainty. Given a reward-free MDP $M = \{S, \mathcal{A}, T\} \backslash R$, a finite set of high-level specifications in STL $\Phi = \Phi_H \cup \Phi_S$ and a finite dataset of human demonstrations $D = \{d_1, d_2, \ldots, d_m\}$ in an environment E, where each demonstration is defined as in Definition 2, the goal is to infer a reward function R for M such that the resulting robot policy π obtained by an RL algorithm, satisfies all the requirements of Φ[2].

Assumptions: Our main focus is on extracting rewards for agents operating in probabilistic and continuous domains and we assume that any appropriate existing RL algorithm would be able to learn an optimal policy from the inferred rewards. We require that at least one *good*, but not necessarily optimal demonstration is provided to the agent, while other demonstrations can be incomplete or *bad*. The states in the demonstration are expected to be unique/distinct, i.e., the demonstration does not contain repeated states. We will investigate demonstrations with repeated states and repeated task specifications as part of future work.

III. Methodology

A. STL-Based Rewards for Stochastic Dynamics

To address stochasticity in environments, we provide a stochastic reward definition for the agent and show that the rewards inferred for deterministic transitions as in [10] are a special case of the new reward function. Rationally, one would expect an agent to perform a given task correctly by following the *good* demonstrations and hence the rewards would be based on such demonstrations. Initially, we follow the DAG-based procedure to obtain the cumulative robustness $\hat{\rho}_d$ for a demonstration d as described in Section II-A. Given a demonstration $d = \langle s_i, a_i \rangle_{i=1}^{L}$ and the final DAG robustness $\hat{\rho}_d$, we derive a procedure to estimate the "true" reward R_d of the demonstration as if the transitions were deterministic. In other words, $R_d = \hat{\rho}_d$ is the reward that the agent would maximize if it were in a deterministic environment. When the environment is stochastic, R_d should increase along the demonstrations to prevent the agent from moving away from the states seen in such demonstrations, i.e., the rewards for a demonstration behave as $attractors$ because they persuade the agent to follow the *good* demonstration as much as possible. Hence, as the environment uncertainty increases, R_d also increases. Here, we consider the states and actions as observed in a demonstration d - the agent starts in state s_1 and executes the corresponding action a_1 as seen in d. Assuming Markovian nature of the environment's stochastic dynamics, for subsequent state-action tuples in d we have,

$$Pr(s_L \mid \tau = (s_i)_{i=1}^{L-1}, a_{L-1}) = \prod_{l=1}^{L-1} Pr(s_{l+1} \mid s_l, a_l) \tag{1}$$

where each a_i is the action indicated in the demonstration and τ is the (partial) trajectory/demonstration till a particular state[3]. Hence, the true reward R_d can be expressed as follows:

$$\begin{aligned} &Pr(s_L \mid \tau = (s_i)_{i=1}^{L-1}, a_{L-1}) \cdot R_d = \hat{\rho}_d \\ &R_d = \frac{\hat{\rho}_d}{\prod_{i=1}^{L-1} Pr(s_{l+1} \in d \mid s_l \in d, a_l \in d)} \end{aligned} \tag{2}$$

This equation reflects that R_d increases as uncertainty increases, i.e., as $Pr(s' \mid s, a) \rightarrow 0$ in the environment. In order to account for the stochasticity, we define $\mathcal{R}(s, a_s)$ as the set of all states that are reachable from a given state s in one step (since it is an MDP) by performing all actions other than its corresponding action a_s appearing in a demonstration.

B. Reward Assignments for Demonstrations

We first present the reward assignments for the discrete-state case and then describe the procedure for continuous state spaces. The rewards for all states are initially assigned to 0.

1) Good Demonstrations: For all state-action pairs occurring in a demonstration d, $r_d(s_l)$ describes the reward assigned to state $s_l \in d$. The reward function is given by Equation (3).

$$\begin{aligned} r_d(s_l) &= Pr(s_l \mid s_{l-1}, a_{l-1}) \cdot \frac{l}{L} \cdot R_d \\ &\quad \forall\, s_{l-1}, s_l, a_{l-1} \in d \\ r_d(s') &= Pr(s' \mid s_{l-1}, a) \cdot r_d(s_l) \\ &\quad s' \in \mathcal{R}(s_{l-1}, a_{l-1}) - \{s_l\}; a \in \mathcal{A} - \{a_{l-1}\} \end{aligned} \tag{3}$$

where $l \in [1, L]$. When, $l = 1$ (initial or base case), $Pr(s_1 \mid s_0, a_0)$ represents the probability of the agent starting in the same state as the demonstrations and (s_0, a_0) is introduced for notational convenience. Good demonstrations have strictly non-negative rewards as they obey all specifications. The rewards in such demonstrations behave as $attractors$ or *potential fields* to

[1] To ensure the robustness values of different specifications can be combined, they are normalized using `tanh` or piece-wise linear functions.

[2] The rewards and resulting policies are empirically verified.

[3] In the above derivation, $\tau = (s_i)$ implicitly means $\tau = (s_i, a_i)$ since it represents a trajectory/demonstration. The a_i notation was dropped for simplicity.

persuade the agent to follow the *good* demonstrations as much as possible. The shape of this reward function resembles a "bell curve".

2) Bad Demonstrations: A bad demonstration will have strictly negative robustness, that is amplified as the true reward R_d as per (Equation (2)).

$$\begin{aligned} r_d(s_l) &= Pr(s_l \mid s_{l-1}, a_{l-1}) \cdot R_d, \quad \text{if } s_l \in s_{bad} \\ r_d(s') &= Pr(s' \mid s_{l-1}, a) \cdot r_d(s_l), \\ & s' \in \mathcal{R}(s_{l-1}, a_{l-1}); a \in \mathcal{A} - \{a_{l-1}\} \end{aligned} \tag{4}$$

The rewards in other all states are zero. The rewards in such demonstrations behave as $repellers$ to deflect the agent from bad states. The shape of this reward function resembles an "inverted bell".

For a demonstration d, the *induced reward* $r_d(s)$ is the reward induced by demonstration d for any state s in the state space, computed via Equations (3) and (4). Let $r_d(s') \prec r_d(s)$ denote that $r_d(s) < r_d(s')$ if d is a bad demonstration and $r_d(s) > r_d(s')$ if d is a $good$ demonstration, for $s \in d$ and $s' \notin d$.

Lemma 1: For any demonstration $d, \forall s_l \in d, r_d(s') \prec r_d(s_l)$.

Proof Sketch: The sum of transition probabilities in a state over all actions is 1. Hence, the product of 2 of these probabilities (as for $r_d(s')$ in Equations (3) and (4)) is less than either of them and is a positive quantity. Therefore, in $good$ demonstrations, the neighbor states s' have lower rewards than the observed state s_l, thereby influencing the agent to not prefer states not seen in $good$ demonstrations and also that there is a possibility that the neighbors are bad states. For bad demonstrations, the neighbors s' have higher rewards than the bad states and are still negative, which influence the agent to move away from bad states and also that there is a chance these neighbors could be $good$ states. ■

Finally, once each demonstration is assigned rewards, they are ranked by their respective R_d values and a rank-based weighted sum is computed to obtain the final reward for the MDP, similar to [10].

C. Special Cases

In this section, we show how the deterministic rewards from the prior LfD-STL framework is a special case of our reward formulation. We also describe a stochastic model used in our experiments.

1) Deterministic: In the case of deterministic transitions, the agent follows the selected action (i.e., $Pr(s' \mid s, a) = 1$) while all other actions have probability of 0. As a result, the probability of transitioning to the neighbor states in 1-step via the other actions is 0. Therefore, this reduces to the same equations described in [10]. By Equation (1), $R_d = \hat{\rho_d}$. The rewards for each type of demonstration are as follows:

- $Good$ demonstration:

$$\begin{aligned} r_d(s_l) &= \frac{l}{L} \cdot R_d, \\ & \forall s_l, a_l \in d; l \in [1, L] \end{aligned} \tag{5}$$

- Bad demonstration:

$$r_d(s_l) = \begin{cases} R_d, & \text{if } s_l \in s_{bad} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

2) Stochastic: Let $p \in [0, 1)$ denote the uncertainty of the environment: the agent follows or executes a selected action $a \in \mathcal{A}$ with probability $Pr(s' \mid s, a) = 1 - p$ and due to the uncertainty, randomly follows/chooses one of the remaining $N - 1$ actions uniformly, i.e., with probability $p/(N-1)$. The sum of probabilities of all transitions or actions is 1. Thus, for a demonstration d, the agent follows d with probability $(1-p)^{L-1}$, by Equation (1). Substituting this in Equation (2), the true reward is:

$$R_d = \frac{\hat{\rho_d}}{(1-p)^{L-1}} \tag{7}$$

With regards to the "attractor-repeller" intuition stated earlier, as the uncertainty p increases, R_d also increases, influencing the agent to follow along the demonstrations. For each type of demonstration, the rewards are described below:

Good demonstrations

$$\begin{aligned} r_d(s_l) &= (1-p) \cdot \frac{l}{L} \cdot R_d; \quad \forall s_l \in d \\ r_d(s') &= \frac{p}{N-1} \cdot (1-p) \cdot \frac{l}{L} \cdot R_d, \\ & s' \in \mathcal{R}(s_{l-1}, a_{l-1}) - \{s_l\}; a \in \mathcal{A} - \{a_{l-1}\} \end{aligned} \tag{8}$$

where $l \in [1, L]$. For the initial state, $Pr(s_1 \mid s_0, a_0)$ could be $1 - p$ or simply 1 if the agent is known to always start from that state. From the above equations and Lemma 1, $r_d(s')$ is guaranteed to be lower than $r_d(s_l)$ since $0 < p < 1 \Rightarrow 0 < p/(N-1) < 1/(N-1) < 1$. By applying simple inequality rules, we can show that $(1-p) \cdot p/(N-1) < (1-p)$, which is the guarantee that reward is propagated in a decreasing manner to neighboring states not seen in the demonstrations.

Bad demonstrations.

$$\begin{aligned} r_d(s_l) &= (1-p) \cdot R_d; \quad \text{if } s_l \in s_{bad} \\ r_d(s') &= \frac{p}{N-1} \cdot (1-p) \cdot R_d, \\ & s' \in \mathcal{R}(s_{l-1}, a_{l-1}); a \in \mathcal{A} - \{a_{l-1}\} \end{aligned} \tag{9}$$

A similar guarantee for reachable states holds here as well. The rewards in other all states are zero. We use this model for all our stochastic discrete environment experiments.

The case of $p = 1$ or as $p \to 1$ represents that the agent is completely non-deterministic (i.e., it never transitions to the desired state or performs the action chosen). In this case, by computing the limits, we can see that the rewards for all states tend to either $+\infty$ in the case of $good$ and to $-\infty$ in the case of bad demonstrations. In such scenarios, the demonstrator may adapt and provide adversarial actions so that the agent performs the originally intended behavior. We will investigate such adversary-influenced demonstrations for future work.

We emphasize that our approach is generic to any $Pr(s' \mid s, a) \in [0, 1)$ and non-uniformity in transition probabilities. The description of all these cases shows that our reward mechanism is complete for stochastic environments. The probabilistic rewarding scheme described above can possibly assign positive rewards in the case of $good$ demonstrations (and negative in bad demonstrations) to bad (and $good$, respectively) states, leading to a reward discrepancy. However, this is compensated for when the STL-based RL algorithm [10] uses the robustness of the partially learned policy w.r.t. hard specifications while learning, to detect and rectify any violations. Alternatively, providing more demonstrations would also overcome discrepancies in rewards, but are not required.

D. Continuous Domain Rewards

For continuous state-spaces, defining rewards for states only encountered in a demonstration is very restrictive and due to the continuous nature of the state and/or action spaces, and numerical accuracy errors, the demonstrations observed will rarely have the same state and/or action values. Additionally, providing demonstrations in this space is already subject to uncertainties. We can first compute the demonstration rewards from DAG-specifications and then assign rewards to the demonstration states as described in the previous section. We then rank the demonstrations and scale the assigned rewards by the corresponding demonstration ranks. The next step is to show how rewards from different demonstrations are generalized and combined over the state space. Since the states in the demonstrations are not exactly the same, simply performing a rank-based weighted sum of state rewards would be tedious due to the large state space. To address this, we collect the rank-scaled state rewards in a data set and perform regression. For each demonstration, we have a collection of tuples in the form of $(state, reward)$ or $(state, action, reward)$ and we can then parameterize the rewards as $r(s, \theta)$ or $r(s, a, \theta)$ respectively. Finally, we organize these points in a dataset to learn a function approximation $f_\theta : S \rightarrow R$ or $f_\theta : S \times \mathcal{A} \rightarrow R$. Function approximations can be learnt via regression techniques like Gaussian Processes, neural networks (NN) such as feed-forward deep NN, convolutional NN, etc., that take as input, the features of a state or state-action pair and output a single/scalar reward.

For discrete actions, it is straight-forward to compute the reachable set. But for continuous actions, in order to compute the reachable set from a given observed state with bounded time and actions, we model each observed state using a (multi-variate) Gaussian distribution and generate samples. These samples correspond to the reachable set and we can compute the probability of each sample belonging to that distribution, which gives us the transition probabilities. Specifically, instead of using each of the tuples in their raw form as training data, we represent them as samples of (multi-variate) Gaussian distribution with mean s or (s, a) and having a scaled identity covariance matrix representing the noise in the observations. We then generate k samples from the distribution of each observed state to represent the reachable set. For each of the k samples, we can estimate the probability of that sample belonging to the distribution of the observed state, which is the transition function that can be used to assign rewards as described earlier.

IV. Experiments

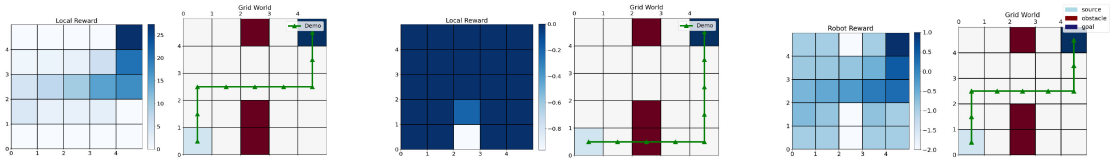
A. Stochastic Discrete Environment

We created a grid-world environment E consisting of a set of states $S = \{start, goals, obstacles\}$ of varying grid sizes such as: $5 \times 5, 8 \times 8$ and 15×15 and randomly choosing the obstacle locations. We experimented with different values of the environment's stochasticity $p \in [0.1, 0.8]$. We used *Manhattan* distance as the distance metric and formulated the STL specifications:

1) Avoid obstacles at all times (hard requirement): $\varphi_1 := \mathbf{G}_{[\mathbf{0},\mathbf{T}]}(\mathbf{d_{obs}[t]} \geq \mathbf{1})$, where T is the length of a demonstration and d_{obs} is the minimum distance of robot from obstacles computed at each step t.
2) Eventually, the robot reaches the goal state (soft requirement): $\varphi_2 := \mathbf{F}_{[\mathbf{0},\mathbf{T}]}(\mathbf{d_{goal}[t]} < \mathbf{1})$, where d_{goal} is the distance of robot from goal computed at each step. φ_2 depends on φ_1 in the DAG.
3) Reach the goal as fast as possible (soft requirement): $\varphi_3 := \mathbf{F}_{[\mathbf{0},\mathbf{T}]}(\mathbf{t} \leq \mathbf{T_{goal}})$, where T_{goal} is the upper bound of time required to each the goal, which is computed by running breadth-first search algorithm from start to goal state, since the shortest policy must take at least T_{goal} to reach the goal. φ_3 depends on both φ_1 and φ_2 in the DAG.

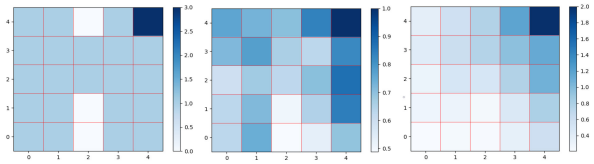
All environments are created using $PyGame$ library and we define and evaluate the STL specifications using Breach [15]. The users provide demonstrations in the $PyGame$ interface by clicking on their desired states with the task to reach the goal state from start without hitting any obstacles. Due to the stochasticity, *unaware to the users*, their clicked state may not always end up at the desired location. The user then proceeds to click from that unexpected state till they quit or reach the goal. 4 demonstrations from a single user were collected, of which 2 are shown in Fig. 2 along with the inferred rewards and resulting robot policy under 20% stochastic environment. We obtained similar results for the other larger grid sizes considered in this experiment. Details for the *OpenAI Gym [16] Frozenlake* 8×8 grid environment are shown in the supplemental video. In all cases, we used Double Q-Learning [17], which is appropriate for stochastic settings, with the modifications to the algorithm at 2 steps (reward update and termination) as described by [10]. The number of episodes varied based on the environment complexity (grid size, number and locations of obstacles) of the grid-world. The discount factor γ was set to 0.8 and ϵ-greedy strategy for action selection with decaying ϵ was used. A learning rate of $\alpha = 0.1$ was found to work reasonably well. For 100 trials, with 20% environment uncertainty, the policy was found to reach the goal on average $\approx 81\%$ of the time with the learned rewards.

We then compared our method with the state-of-the-art IRL algorithm - Maximum Causal Entropy IRL (MCE-IRL) [18] having unique features for each state, which required around 300 demonstrations in the 5×5 grid world under identical stochastic conditions and over 1000 demonstrations for the $Frozenlake - 8 \times 8$. Additionally, since MCE-IRL learns a reward for each state, it requires the demonstration set to cover all possible states, while ours does not require this criteria and hence can learn from few demonstrations. A ground-truth reward function for the 5×5 grid is shown in Fig. 3(a). Qualitatively, from Fig. 2(c) and Fig. 3, we can observe that the rewards using our method are more aligned with the ground truth, compared to the others. For instance, the state at the center of the top row is an obstacle and both the IRL methods infer positive rewards for that state, while ours correctly computes a negative reward for the same using fewer samples. We remark that there are other recent works [19], [20] that learn from suboptimal demonstrations. However, at the core, they build on the entropy-based IRL methods and can learn more accurate rewards compared to MCE-IRL at the cost of generating additional demonstrations [19] or manually defining an environment reward function in addition to the learned rewards [20]. MCE-IRL offers a good balance between performance and sample complexity compared to these methods and since they all build on MaxEntropy-IRL [5], we used MCE-IRL for comparison. Furthermore, MCE-IRL and our method have the common objective of inferring rewards from demonstrations and mainly differ in the manner features are utilized: in MCE-IRL, features are directly used in computing the solution, while ours indirectly accesses the features via specifications.



(a) Demo 1: Optimal and good (b) Demo 2: Bad (c) Learned reward and robot policy

Fig. 2. Results for 5×5 grid with 20% stochasticity: Inferred rewards are shown in left figures. Right figures show the grid-world with start state (light blue), goal (dark blue), obstacles (red) and demonstration/policy (green). Darker blues represent higher rewards.



(a) GT reward function (b) MaxEnt-IRL (c) MCE-IRL

Fig. 3. Ground-truth (GT) rewards and rewards extracted by ME-IRL and MCE-IRL, respectively, each using 300 optimal demonstrations.

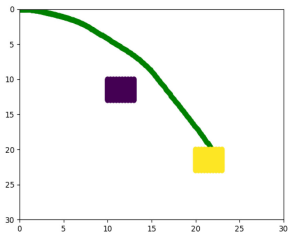


Fig. 4. The car starts at the top-left corner and the task is to navigate to the goal (in yellow) while possibly avoiding potholes or obstacles (purple). A sample demonstration is shown by the green trajectory.

B. *Continuous-Space Environment*

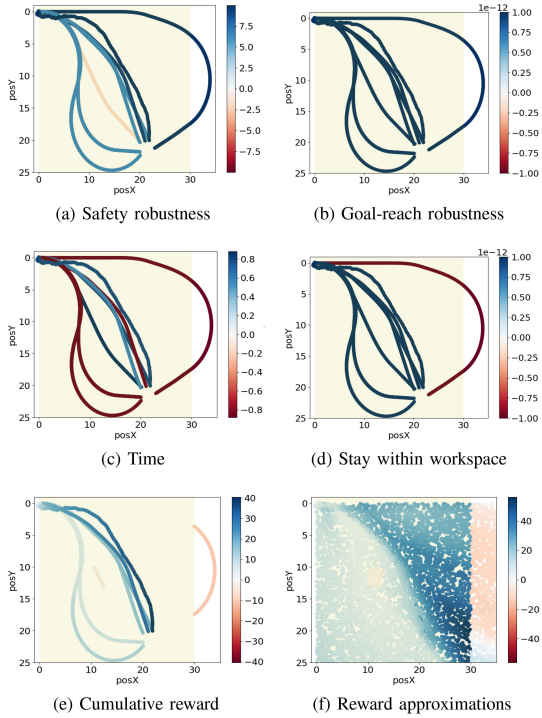
We used a simple car kinematic model that is governed by the following equations:

$$\dot{x} = v \cdot cos(\theta) + \mathcal{N}(0, \sigma^2);\ \dot{y} = v \cdot sin(\theta) + \mathcal{N}(0, \sigma^2)$$

$$\dot{v} = u_1 \cdot u_2;\ \dot{\theta} = v \cdot tan(\psi);\ \dot{\psi} = u_3$$

where x and y represent the XY position of the car; θ is the heading; v is the velocity; u_1 is the input acceleration; u_2 is the gear indicating forward (+1) or backward (-1); u_3 is the input to steering angle ψ. At any time instant t, the state of the car is given by $S_t = [x, y, \theta, v, \dot{x}, \dot{y}, \dot{\theta}, \dot{v}]^T$. Users can control the car using either an analog *Logitech G29* steering with pedal controller or via keyboard inputs. Alternatively, one could also use a similar setup for mobile robots using respective kinematics and a joystick controller for acute turns.

The driving layout with goal and obstacle areas, and a sample demonstration is shown in Fig. 4. The task is to drive from the top-left corner to the center of the goal while avoiding any hindrances (obstacles, potholes, etc.), denoted by $\mathcal{H}$. As in any driving scenario, the car must maintain a safe distance $\texttt{dSafe}$ from $\mathcal{H}$ and drive on the road/drivable surface. We collected 8 demonstrations (6 good and 2 bad) using a mixture of analog and keyboard inputs; one of the bad demonstrations passed through the pothole while another drove off the "road". The distance metric used in this space is $Euclidean$. The specifications for this scenario are as follows:



(a) Safety robustness (b) Goal-reach robustness
(c) Time (d) Stay within workspace
(e) Cumulative reward (f) Reward approximations

Fig. 5. (a)–(d): Robustness of demonstrations w.r.t. each STL specification. Blue trajectories indicate positive robustness and red indicate negative. (e): Final rewards based on cumulative robustness and demonstration ranking. (f): Reward approximation using neural networks. The yellow region represents the workspace of the agent, i.e., it is not allowed to leave that region.

1) Avoid obstacles at all times (hard requirement): $\varphi_1 := \mathbf{G}_{[0,T]}(\mathbf{d_{obs}[t]} \geq \mathbf{dSafe})$, where T is the length of a demonstration and d_{obs} is the minimum distance of the car from $\mathcal{H}$ computed at each step t. For our experiments, we used $\texttt{dSafe} = 3$ units.
2) Always stay within the workspace/drivable region (hard requirement): $\varphi_2 := \mathbf{G}_{[0,T]}((\mathbf{x}, \mathbf{y}) \in \mathbf{Box(30, 25)})$, where the workspace is defined by a rectangle of dimensions 30×25 square units.
3) Eventually, the robot reaches the goal state (soft requirement): $\varphi_3 := \mathbf{F}_{[0,T]}(\mathbf{d_{goal}[t]} < \delta)$, where d_{goal} is the distance between centers of car and goal computed at each step t and δ is a tolerance when the center of the car is "close enough" to the goal's center. φ_3 depends on φ_1 and φ_2 in the DAG.

Similar specifications can be used for manipulators and mobile robots. The collected trajectories along with their robustness for each STL specification and also for the time taken by them to reach the goal are shown in Fig. 5. One of the *bad*

demonstrations scraped the avoid-region and is shown in red in Fig. 5(a). All demonstrations reached the goal and are shown in blue in Fig. 5(b). In Fig. 5(c), 4 out of 8 demonstrations were slow to reach the goal compared to the others. One bad demonstration went "off track" and is shown in red in Fig. 5(d). All these individual robustness values are combined via the specification-based DAG for each trajectory and rewards are assigned by modeling the states s as samples of multi-variate Gaussian distribution $\mathcal{N}(\mu, \sigma^2 I)$ where $\mu = s$ and σ represents the deviations in noise levels. For each s, we generated $k = 20$ samples to represent the reachable set and assigned stochastic rewards as described in Section III. The final rewards are shown in Fig. 5(e) for low noise with $\sigma = 0.03$ (for additional details on analysis of noise, please refer to the supplemental video). In the continuous spaces, we can observe that there is a large area of states with reward 0 (in white) and may not be particularly helpful since the agent rarely encounters the same state as seen in demonstrations due to noise. To overcome this issue, we approximated rewards over the state space using neural network regression. We thus combine the precision of Gaussian distributions for reward inference and the scalability of neural networks for predictions.

Remark 1: In all experiments, the reward plots were normalized and the maximum reward was capped to a sufficiently large value R_{max} for the sake of practical/numerical implementation and visualization simplicity. Additionally, to combine robustness from semantically different STL specifications, we used `tanh` to normalize the robustness before combination. For the driving experiment, the state space has higher dimensions which becomes too convoluted to visualize. Instead, to show how the neural network regression would perform with smaller dimension inputs, we use the XY positions of the car along with the type of the XY state as inputs to the network. The type of the state is a one-hot encoding of whether the state represents an obstacle/avoid region, goal, outside-workspace or traversable region. We assume that a perception algorithm would provide the label of each state.

The neural network contained 2 hidden layers with 100 and 200 nodes respectively; and used the Adam optimizer [21] with batch training for 20 epochs and RMSE loss. It was trained using $PyTorch$ on a system with *AMD Ryzen 7 3700X 8-core* CPU and *Nvidia RTX 2070-Super* GPU. As we see in Fig. 5(f), the predictions are closely correlated with the locations of various map features (boundaries, avoid regions and goal). In all the experimental scenarios, the final rewards can be verified w.r.t. the specifications to detect violations.

V. Related Work

There is a vast literature on methods that utilize the quantitative semantics of temporal logics such as STL, Linear Temporal Logic (LTL), etc., to describe or shape reward functions in the RL domain [22]–[25]. [26] tries to extract a stochastic policy that satisfies the temporal logic specifications, which can be used once the rewards are inferred using our method. LfD with formal methods has been explored in [27] to learn complex tasks by designing a special logic, augmenting a finite-state automaton (FSA) with an MDP formulation and then performing behavioral cloning to initialize policies that are later trained via reinforcement LfD. However, this work relies on optimal/perfect demonstrations. [28] proposes a counterexample-guided approach using probabilistic computation tree logics for safety-aware apprenticeship learning and [10] develops a framework for providing high-level STL specifications that evaluate and rank the quality of demonstrations to infer rewards used by existing RL algorithms to learn a robust control policy; both these methods perform logic-checking during training to achieve verification-in-the-loop and can learn from counterexamples or demonstrations that violate hard requirements. In [29], the authors combine LfD with LTL by converting the LTL specifications into a loss function to learn a dynamic movement primitive that satisfies the specifications and tries to imitate the demonstrations. This work requires manually defining loss functions based on the quantitative semantics of LTL and it does not seek to learn a reward function which is the main objective of our letter. The authors in [30] integrate a learning method with model-predictive control (MPC) to design a controller that behaves similar to expert demonstrations while trying to decide the trade-offs on how well to follow each STL rule using slackness in robustness values. They assume that priorities among the STL rules are already given and also assume that the experts are aware of the priorities among the STL rules and provide demonstrations accordingly. However providing such demonstrations requires optimality and skill. LfD with high-level side information encoded in co-safe LTL has been explored in [31]. This method learns a reward function as well as an FSA that is augmented with the MDP states to learn from a handful of optimal demonstrations. Some issues with this approach are that rewards and hence the FSA rely on demonstrations being optimal; and the augmented state space increases exponentially as the number and length of specifications increase. Ours on the other hand, uses the STL specifications to evaluate and rank the demonstrations, without increasing the state space. To summarize, majority of the prior research uses formal logic to verify learned policies whereas we try to infer rewards that are consistent with the task specifications.

A survey of methods that learn from suboptimal/imperfect demonstrations is described in [7]. In many cases, these methods filter the sub-optimal or imperfect demonstrations or classify such demonstrations when majority of the other demonstrations are optimal. Recently, the authors of [19] proposed a framework to learn the latent reward function from suboptimal demonstrations by examining the relationships between the performance of a policy and the level of noise injected to synthesize optimality-ranked trajectories. In [20], the authors propose an approach in which nearly-optimal demonstrations are used to learn rewards via IRL, that are later utilized for reward shaping in RL tasks.

VI. Conclusion

We proposed a novel approach that significantly improved upon the existing LfD-STL framework [10] by considering the uncertainty in the environment to define temporal-based rewards from suboptimal demonstrations. We also proposed a method to learn and predict rewards in continuous and high-dimension spaces. These rewards can be used to extract robust and interpretable RL control policies. The experiments on several stochastic discrete-worlds and in the driving scenario (continuous domain) illustrate the effectiveness and scalability of our method. We believe this letter will provide new avenues to combine verification of rewards (such as this work) and verification of learned policies of agents [29]–[31] to develop safe and interpretable control policies for applications in autonomous driving, ground and aerial robot surveillance or patrol, household and medical assistants, etc.

REFERENCES

[1] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *Proc, Int. Conf. Mach. Learn.*, 1997, pp. 12–20.

[2] S. Schaal, "Learning from demonstration," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 1996, pp. 1040–1046.

[3] F. Torabi, G. Warnell, and P. Stone, "Behavioral cloning from observation," in *Proc. Int. Joint Conf. Artif. Intell.*, 2018, pp. 4950–4957.

[4] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proc, Int. Conf. Mach. Learn.*, 2000, pp. 663–670.

[5] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, 2008, pp. 1433–1438.

[6] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proc, Int. Conf. Mach. Learn.*, vol. 69, 2004.

[7] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration," *Annual Review of Control, Robotics, Auton. Syst.*, vol. 3, pp. 297–330, 2020.

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

[9] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Comput. Surv.*, vol. 50, no. 2, pp. 1–35, 2017.

[10] A. G. Puranic, J. V. Deshmukh, and S. Nikolaidis, "Learning from demonstrations using signal temporal logic," in *Proc. Conf. Robot Learn.*, 2020, *p. (To appear*).

[11] J. V. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, and S. A. Seshia, "Robust online monitoring of signal temporal logic," *Formal Methods System Des.*, vol. 51, no. 1, pp. 5–30, 2017.

[12] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theor. Comput. Sci.*, vol. 410, no. 42, pp. 4262–4291, 2009.

[13] S. Jakšić, E. Bartocci, R. Grosu, T. Nguyen, and D. Ničković, "Quantitative monitoring of STL with edit distance," *Formal Methods System Des.*, vol. 53, no. 1, pp. 83–112, Aug. 2018.

[14] 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.

[15] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems," in *Proc. Int. Conf. Comput. Aided Verification*, 2010, pp. 167–170.

[16] G. Brockman *et al.*, "Openai Gym," in *CoRR*, 2016, *arXiv:1606.01540*.

[17] H. Hasselt, "Double q-learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2010, pp. 2613–2621.

[18] B. D. Ziebart, "Modeling purposeful adaptive behavior with the principle of maximum causal entropy," Ph.D. dissertation, Carnegie Mellon University, USA, 2010.

[19] L. Chen, R. Paleja, and M. Gombolay, "Learning from suboptimal demonstration via self-supervised reward regression," 2020, *arXiv:2010.11723*.

[20] H. B. Suay, T. Brys, M. E. Taylor, and S. Chernova, "Learning from demonstration for shaping through inverse reinforcement learning," in *Proc. Int. Conf. Auton. Agents Multiagent Syst. AAMAS*, 2016, pp. 429–437.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, 2015.

[22] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "Q-learning for robust satisfaction of signal temporal logic specifications," in *Proc. Conf. Decis. Control*, 2016, pp. 6565–6570.

[23] X. Li, C. Vasile, and C. Belta, "Reinforcement learning with temporal logic rewards," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 3834–3839.

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

[25] Y. Jiang, S. Bharadwaj, B. Wu, R. Shah, U. Topcu, and P. Stone, "Temporal-logic-based reward shaping for continuing learning tasks," 2020, *arXiv:2007.01498*.

[26] X. Li, Y. Ma, and C. Belta, "A policy search method for temporal logic specified reinforcement learning tasks," in *Proc. Amer. Control Conf.*, 2018, pp. 240–245.

[27] X. Li, Y. Ma, and C. Belta, "Automata guided reinforcement learning with demonstrations," CoRR, 2018, *arXiv:1809.06305* .

[28] W. Zhou and W. Li, "Safety-aware apprenticeship learning," in *Proc. Int. Conf. Comput. Aided Verification*, 2018, pp. 662–680.

[29] C. Innes and S. Ramamoorthy, "Elaborating on learned demonstrations with temporal logic specifications," in *Robot: Sci. Syst.*, 2020.

[30] K. Cho and S. Oh, "Learning-based model predictive control under signal temporal logic specifications," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 7322–7329.

[31] M. Wen, I. Papusha, and U. Topcu, "Learning from demonstrations with high-level side information," in *Proc. Int. Joint Conf. Artif. Intell.*, 2017, pp. 3055–3061.