A Learning Framework for Versatile STL Controller Synthesis

Peter Varnai and Dimos V. Dimarogonas[1]

Abstract— **In this paper, we aim towards providing a practical framework for learning to satisfy signal temporal logic (STL) task specifications for systems with partially unknown dynamics. We consider STL tasks whose satisfaction can be guaranteed by enforcing *a priori* known temporal specifications imposed on the atomic propositions that compose them. First, a neural network is trained offline as a control policy to satisfy such temporal specifications while also minimizing a target cost, such as the input energy of the system. The obtained controller then serves as a guide that aids exploration while learning to satisfy any specific STL task optimally using policy improvement, greatly increasing the sample efficiency of the procedure. The promise of the approach towards a versatile STL learning framework is demonstrated through simulations.**

I. INTRODUCTION

This paper considers synthesizing controllers for systems with partially unknown dynamics in order to minimize a target cost while satisfying time-dependent tasks. The tasks are given in the form of temporal logic (TL) specifications [1], which allow the expression of complex behaviors such as surveilling regions of interest and periodically to recharging. Control strategies for satisfying TL tasks in known environments have been looked into in detail (e.g., [2], [3]), while control synthesis in unknown and stochastic environments has also gained considerable attention recently [4]–[7]. The focus is generally placed on task satisfaction itself in a robust manner, though some works also consider real-time plan revision [8] or added target costs [9]. Here, we consider both aspects and aim to develop a sample efficient framework for the optimal policy synthesis of solving a variety of TL tasks.

In this work, the temporal tasks are formulated using STL [10], where the logical predicates are defined over functions of the system state. STL is particularly suited for learning methods as it is equipped with various robustness metrics that quantify how well the task is satisfied. This constitutes a reward that can be efficiently exploited, such as by using Q-learning [5], [11] or policy improvement techniques [12]. The former relies on trajectory histories of the system to generate an optimal policy, while the latter must restart the learning process at every initial state of the system. Both approaches solve a single STL task. To the authors' knowledge, this is the first work towards a sample-efficient learning framework capable of dealing with a variety of STL task specifications.

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, the Swedish Research Council (VR), the SSF COIN project, and the EU H2020 Co4Robots project.

[1]Both authors are with the Division of Decision and Control Systems, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, 114 28 Stockholm, Sweden. `varnai@kth.se` (P. Varnai), `dimos@kth.se` (D. V. Dimarogonas)

The proposed approach is based on policy improvement [12], where a parameterized controller is iteratively updated towards the optimum using a multitude of sampled system trajectories. Our previous work [13] has shown that the efficiency of this algorithm is greatly improved when augmented with prescribed performance control (PPC)-based control [3] to guide the learning. This controller aids STL task satisfaction, leading to more sample-efficient exploration towards minimizing the target cost. Here, this idea is further extended by training a guiding controller that already aims to minimize the target cost while satisfying the STL task, further increasing the effectiveness of exploration.

A secondary goal is for the learning framework to be readily adaptable to a variety of task specifications, allowing the system to appropriately react to real-time task changes. To this end, the trained guide controller does not focus on a particular STL task, but on satisfying any *robustness specification* imposed on the evolution of atomic propositions composing possible STL formulas. This allows an effective transfer of gathered offline experience to specific STL formulas online, provided that a good estimate of robustness specifications guaranteeing the satisfaction of the formula are known. We assume such specifications are available *a priori*, but give ideas for their construction in future work.

The rest of the paper is organized as follows. Section II gives a brief introduction to STL and policy improvement. The problem statement is formulated in Section III, followed by the details of our solution approach in Section IV and a simulation study in Section V. Finally, Section VI concludes with the main take-aways and considerations for future work.

II. PRELIMINARIES

A. Signal temporal logic (STL)

STL is a form of temporal logic defined over real-valued and real-time signals [14]. The underlying atomic predicates μ are either true($\top$) or false($\perp$) depending on the sign of a corresponding function $h^\mu(\boldsymbol{x})$; $\mu = \top$ if $h^\mu(\boldsymbol{x}) \geq 0$ and $\mu = \perp$ if $h^\mu(\boldsymbol{x}) < 0$. Boolean and temporal operators then allow the definition of more complex STL expressions ϕ:

$$\phi := \top \mid \mu \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 U_{[a,b]} \phi_2. \tag{1}$$

The notation $(\boldsymbol{x}, t) \vDash \phi$ expresses that a signal $\boldsymbol{x}(t)$ satisfies the task ϕ at time t. This can be determined in a recursive fashion using the following semantics [3]: $(\boldsymbol{x}, t) \vDash \mu$ iff $h^\mu(\boldsymbol{x}(t)) \geq 0$; $(\boldsymbol{x}, t) \vDash \neg\phi$ iff $\neg((\boldsymbol{x}, t) \vDash \phi)$; $(\boldsymbol{x}, t) \vDash \phi_1 \wedge \phi_2$ iff $(\boldsymbol{x}, t) \vDash \phi_1 \wedge (\boldsymbol{x}, t) \vDash \phi_2$; and finally, $(\boldsymbol{x}, t) \vDash \phi_1 U_{[a,b]} \phi_2$ iff $\exists t_1 \in [t+a, t+b]$ such that $(\boldsymbol{x}, t_1) \vDash \phi_2$ and $(\boldsymbol{x}, t_2) \vDash \phi_1$ for all $t_2 \in [t, t_1]$. The time bounds of the *until* operator U

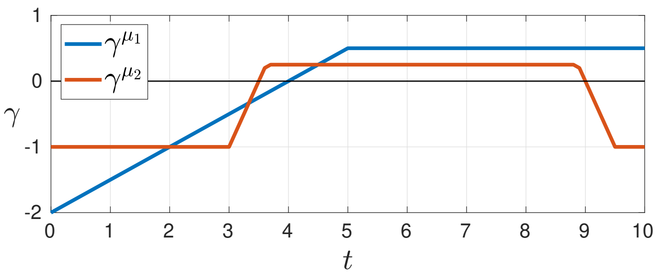


Fig. 1: Sample robustness specifications guaranteeing satisfaction of the STL task studied in Example 1.

satisfy $a \le b$; the operators *eventually* and *always* are then expressed by $F_{[a,b]}\phi = \top U_{[a,b]}\phi$ and $G_{[a,b]}\phi = \neg F_{[a,b]}\neg\phi$.

In STL, it is possible to measure how well a formula is satisfied using various notions of robustness metrics [10]. Here, we use the *spatial robustness* metric ρ whose definition is as follows for the formulas used herein: $\rho^{\mu}(\boldsymbol{x}, t) = h^{\mu}(\boldsymbol{x}(t))$, $\rho^{\neg\phi}(\boldsymbol{x}, t) = -\rho^{\phi}(\boldsymbol{x}, t)$, $\rho^{\phi_1 \wedge \phi_2}(\boldsymbol{x}, t) = \min\left(\rho^{\phi_1}(\boldsymbol{x}, t), \rho^{\phi_2}(\boldsymbol{x}, t)\right)$, $\rho^{F_{[a,b]}\phi}(\boldsymbol{x}, t) = \max_{t' \in [t+a, t+b]} \rho^{\phi}(\boldsymbol{x}, t')$, and for the always operator $\rho^{G_{[a,b]}\phi}(\boldsymbol{x}, t) = \min_{t' \in [t+a, t+b]} \rho^{\phi}(\boldsymbol{x}, t')$. A formula ϕ is satisfied by the signal $\boldsymbol{x}(t)$ at time t if the corresponding robustness metric $\rho^{\phi}(\boldsymbol{x}, t) \ge 0$.

The atomic predicates μ themselves are equipped with a robustness measure. A temporal specification ϕ can be made to be satisfied by properly controlling the evolution of these measures in time, which motivates the following definition.

Definition 1 (Robustness specification). The *robustness specification* of an atomic proposition μ is defined by a curve $\gamma^{\mu}(t)$ and is satisfied if $\rho^{\mu}(\boldsymbol{x}(t)) \ge \gamma^{\mu}(t)$ holds for all t.

Example 1. Consider the task $\phi = F_{[0,5]}\mu_1 \wedge G_{[0,5]}(\mu_1 \Rightarrow F_{[2,4]}G_{[0,1]}\mu_2)$. In words, the task describes that μ_1 must happen within 5s, and any time it does, μ_2 must hold for an entire second starting sometime between $2-4$s afterwards. A plausible robustness specification for how μ_1 and μ_2 should evolve in order to guarantee satisfaction of ϕ is depicted in Fig. 1. Note that there are many pairs of specification curves which provide such a guarantee.

B. *Policy improvement with path integrals (PI^2)*

Consider the following optimization problem:

$$\min_{\theta} J(\tau_{[0,T]}^{\pi_\theta}), \tag{2}$$

where π_θ is a control policy parameterized by $\theta \in \mathbb{R}^p$, $\tau_{[0,T]}^{\pi_\theta}$ is the generated system trajectory[1] obtained by following this policy, T is the problem horizon, and $J(\cdot)$ is a cost to be minimized. The PI^2 algorithm is a learning algorithm used to solve such problems by exploring the parameter space and updating θ until a (locally) optimal solution is found. Originally introduced in [15], PI^2 was proven to converge for specific objectives and is known to perform well for others.

[1]A system trajectory $\tau_{[0,T]}$ denotes the collection of the state and input signals $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$ during the time interval $t \in [0, T]$.

In our PI^2 framework, the policy is a function of the system state and time and is expressed in the form[2]:

$$\pi_\theta(\boldsymbol{x}_t, t) = \hat{\boldsymbol{u}}(\boldsymbol{x}_t, t) + \boldsymbol{k}_t(\theta). \tag{3}$$

The first term, $\hat{\boldsymbol{u}}(\boldsymbol{x}_t, t)$, is a base control action aiming to guide the exploration towards minimizing the given cost J. The second term, $\boldsymbol{k}_t(\theta)$, is a parameterized feedforward term which enables exploration towards the optimum. We assume this parametrization allows for degrees of freedom in every instance of time, i.e., $\theta = \{\theta_0, \dots, \theta_T\}$.

The PI^2 algorithm used in this paper to solve problems of form (2) and to handle STL task specifications is based on our previous work [13]; a concise description is as follows:

- The algorithm is initialized with parameter estimates $\theta_t^{(0)}$ for $t \in [0, T]$ and a Gaussian distribution defined by a covariance $\mathbf{C}_t^{(0)}$ around each mean $\theta_t^{(0)}$.
- In the (k)-th iteration, N parameters $\tilde{\theta}_{t,i}$, $i = 1, \dots, N$, are sampled from the distribution around each $\theta_t^{(k)}$, generating N trajectories by following each policy $\tilde{\theta}_{t,i}$. Each sample is assigned a weight w_i based on its corresponding cost $J_i = J(\tau^{\tilde{\theta}_{t,i}})$. Parameterizations leading to lower costs are given larger weights in order to move the solution towards the optimum.
- Finally, the policy parameters are updated using the weighted average $\theta_t^{(k+1)} = \sum_{i=1}^{N} w_i \tilde{\theta}_{t,i}$ and the previous steps are repeated a given K number of times.

In the context of STL, PI^2 is advantageous since it relies on a cost associated to entire trajectories, such as the robustness metric of a temporal task ϕ (which is determined at the end of the task's time horizon). Aiming to satisfy ϕ with an imposed minimal robustness $\rho^{\phi} \ge \rho_{\min} \ge 0$, the cost of interest $C(\tau)$ is augmented with a penalty term to obtain the objective of the optimization problem (2) as:

$$J(\tau) := J^{\lambda}(\tau, \rho^{\phi}) = C(\tau) + P^{\lambda}(\rho^{\phi}). \tag{4}$$

The penalty function P^{λ} is such that as $\lambda \to \infty$, we have $P^{\lambda}(\rho^{\phi}) \to \infty$ for $\rho^{\phi} < \rho_{\min}$ and $P^{\lambda}(\rho^{\phi}) \to 0$ for $\rho^{\phi} \ge \rho_{\min}$. The robustness constraint $\rho^{\phi} \ge \rho_{\min}$ is thus treated as a soft constraint and is moved towards a hard constraint by gradually increasing λ throughout the PI^2 iterations.

III. Problem Formulation

A. *System and task description*

The system under consideration is given by the nonlinear dynamics:

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u}, \qquad \boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{5}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{u} \in \mathcal{U} \subseteq \mathbb{R}^m$ are the system state and input, respectively. The functions $f(\boldsymbol{x})$ and $g(\boldsymbol{x})$ are assumed to be locally Lipschitz continuous; the latter is known and satisfies $g(\boldsymbol{x})g^{\mathrm{T}}(\boldsymbol{x})$ being positive definite for all $\boldsymbol{x} \in \mathbb{R}^n$. Furthermore, the term $g(\boldsymbol{x})$ is known and we assume an approximation of $f(\boldsymbol{x})$ is available in order to simulate the system. The initial state is given by $\boldsymbol{x}_0 \in \mathbb{R}^n$.

[2]The subscripts t denote signals at a given time slice t, e.g., $\boldsymbol{x}_t = \boldsymbol{x}(t)$, bridging notation between continuous time and its discretized setting.

B. Problem statement

The problem studied in this paper is formalized as follows.

Problem 1. Consider the dynamical system (5) tasked with an STL specification ϕ during a time horizon T. Compute a control policy $\pi^{\phi}(\boldsymbol{x}, t) : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^m$ under which the system evolves in a trajectory $\tau^{\pi^{\phi}}_{[0,T]}$ that satisfies ϕ with a given minimal robustness $\rho^{\phi} \geq \rho_{\min} \geq 0$ while minimizing a target cost $C(\tau)$.

Assumption 1. The task ϕ is composed of M atomic propositions μ_i with known robustness metrics $\rho^{\mu_i}(\boldsymbol{x})$, $i = 1, \dots, M$. Corresponding robustness specifications $\gamma_i(t)$ are known such that $\rho^{\mu_i}(\boldsymbol{x}(t)) \geq \gamma_i(t)$ for all $i = 1, \dots, M$ is feasible for the system and guarantees satisfaction of ϕ with the imposed minimal robustness $\rho^{\phi} \geq \rho_{\min}$.

The problem is formulated in a similar manner to the one examined in our previous work [13]. Therein, a PPC-based analytical law was used in (3) as a base law $\hat{\boldsymbol{u}}(\boldsymbol{x}_t, t)$ to guide the exploration in PI^2 towards satisfying the given STL task ϕ. This was done by aiming to enforce each $\rho^{\mu_i}(\boldsymbol{x}(t)) \geq \gamma_i(t)$ through proper control of the metric ρ^{μ_i}, which necessitates the presented assumptions on the system dynamics. While utilizing such a base law was shown to yield substantial improvement in the convergence rate of PI^2, the learning procedure essentially had to start anew from every initial condition of the system, because the base law was not tailored towards minimizing the specific target cost $C(\tau)$ in addition to satisfying the STL task. In this work, the emphasis is placed on an even more *sample efficient* solution, moving towards a practically feasible algorithm. The key idea is to train a neural network as a base law that also attempts to minimize the target cost of interest, thus transferring online computational loads to an offline training phase.

To the authors' knowledge, previous works employing machine learning for solving STL tasks focused on variations of Q-learning adapted to the STL setting, such as in robust satisfaction of STL tasks [5] or in an adversarial environment [16]. In case of STL formulas, the input to the policy neural network must include the entire trajectory history up until the time horizon of the formula ϕ in order to keep track of its progression. This presents a computational challenge for complex formulas; the examples provided in the mentioned papers have a time horizon of a few simulated time steps in a discrete time, space, and action setting, and it is not clear if the proposed methods scale well for longer horizons. Furthermore, the trained policy targets a *single* STL task, while in our problem formulation the focus is again on a sample-efficient online adaptation to *any* STL task at hand. This suggests that the offline gathered experience should be transferable to a variety of task specifications, motivating the proposed approach presented in the following section.

IV. Solution

Our proposed learning framework for efficiently synthesizing controllers subject to STL tasks while minimizing a target cost $C(\cdot)$ consists of two stages:

(1) A neural network is trained as a policy which aims to satisfy any set of robustness specifications $\rho^{\mu_i}(\boldsymbol{x}(t)) \geq \gamma_i(t)$ laid on the atomic propositions μ_i while minimizing the cost $C(\tau)$ of the resulting trajectories.

(2) The trained policy is used as the base law $\hat{\boldsymbol{u}}(\boldsymbol{x}_t, t)$ in (3) for the PI^2 algorithm in order to solve Problem 1, similarly as in the previous work [13], but offering much better sample efficiency (and thus potentially practical applicability) due to the previously gathered experience.

A. Offline stage - satisfying robustness specifications

In the first stage, a neural network is trained to aim at minimizing the target trajectory cost $C(\tau)$ while satisfying the robustness specification requirements imposed by the curves $\gamma_i(t)$. To this end, the trajectory cost is augmented with a penalty term as follows:

$$J(\tau) = C(\tau) + \sum_{i=1}^{M} \int_0^T P^{\lambda_i}(\rho^{\mu_i}(\boldsymbol{x}(t)), t)\mathrm{d}t. \qquad (6)$$

The penalty coefficients λ_i control the importance of satisfying the i-th robustness specification $\rho^{\mu_i}(\boldsymbol{x}(t)) \geq \gamma_i(t)$. Hence, similarly as in (4), P^{λ_i} has a form such that as $\lambda_i \to \infty$, $P^{\lambda_i}(\rho^{\mu_i}(\boldsymbol{x}(t)), t) \to \infty$ for $\rho^{\mu_i}(\boldsymbol{x}(t)) < \gamma_i(t)$ and $P^{\lambda_i}(\rho^{\mu_i}(\boldsymbol{x}(t)), t) \to 0$ otherwise. In this paper, we use functions composed of a linear and a sigmoid term parameterized by the scalars $\alpha, \beta > 0$ of the form:

$$P^{\lambda_i}(\rho^{\mu_i}(\boldsymbol{x}), t) = \alpha \xi_i(\rho^{\mu_i}(\boldsymbol{x}), t) + \frac{\lambda_i}{1 + e^{-\beta \lambda_i (\xi_i(\rho^{\mu_i}(\boldsymbol{x}), t) - 1)}}, \qquad (7)$$

where the transformation $\xi_i(\rho^{\mu_i}(\boldsymbol{x}), t) := \frac{\Gamma_i(t) - \rho^{\mu_i}(\boldsymbol{x})}{\Gamma_i(t) - \gamma_i(t)}$ using predefined curves $\Gamma_i(t) > \gamma_i(t)$ controls how close to the robustness specification boundary $\gamma_i(t)$ the penalization begins. Note that the value $\xi_i = 1$ corresponds to $\rho^{\mu_i}(\boldsymbol{x}(t)) = \gamma_i(t)$. In (7), the sigmoid term serves as a barrier, while the added (small) linear term provides gradient information to push solutions towards the desired robustness. The form of this penalty greatly impacts the behavior of the trained neural network, and it would be interesting to make connections between choices that perform well in this case and in the design of analytical base laws discussed in [13].

In order to generate data for training a neural network policy, we compute the solution to a multitude of sample problems with objective (6) for a given cost $C(\tau)$ from various initial conditions and robustness specification curves. The PI^2 algorithm is employed to find such solutions.

Each sample problem solution provides training data in the form of an input $\{\boldsymbol{x}(t), \boldsymbol{\gamma}_{[t,T]}\}$ and a corresponding output, the optimal control action $\boldsymbol{u}(t)$. The variable $\boldsymbol{\gamma}_{[t,T]}$ denotes the evolution of the assembled robustness specification curves $\boldsymbol{\gamma}(t) := [\gamma_1(t), \dots, \gamma_M(t)]$ during the time horizon $[t, T]$. The neural network serves as a universal function approximator for the optimal policy $\pi^*_{NN}(\boldsymbol{x}(t), \boldsymbol{\gamma}_{[t,T]})$, a function of both the system state and the entire future specification curve, and can be trained using well-established techniques in stochastic optimization [17].

From a practical perspective, the optimal control action $\boldsymbol{u}(t)$ at time t could mainly be influenced by the initial portion of $\boldsymbol{\gamma}(t)$. The policy can be thus treated as a function of a smaller horizon window, e.g., $\pi_{NN}(\boldsymbol{x}, \boldsymbol{\gamma}_{[t,t+\Delta T]})$, or using a lower-dimensional representation of the curve $\boldsymbol{\gamma}_{[t,t+\Delta T]}$. The idea is similar to receding horizon control [18] and substantially decreases the amount of training data required to train π_{NN} due to the dimensionality reduction of its input specification curve. Such a reduction is not possible in a trajectory-based training as in previous works [5], [11], since the entire state history is necessary to determine progress in satisfying an STL task. Note that it is not crucial for our trained policy to be fully optimal, since it is only used as a base law in PI^2 when solving Problem 1 in the second stage.

A further aspect to consider is selecting the penalty values λ_i for the sample problems used to generate training data. Higher penalties better accomplish the training goal of satisfying the robustness specifications while minimizing the target cost $C(\tau)$. On the other hand, there should be training data for cases when the robustness specifications are not satisfied. The trained policy must learn how to respond in such situations, which are encountered during exploration in PI^2 or if the given robustness specifications are infeasible. If $C(\tau)$ becomes negligible compared to the incurred penalties, optimal solutions may be difficult to find due to numerical reasons in these cases. This trade-off implies that the penalty values should be chosen such that the two cost terms of the objective (6) do not differ much in orders of magnitude.

B. Online stage - satisfying the STL task

In the second stage, the pre-trained policy π_{NN} is used as a base law in the PI^2 algorithm discussed in Section II-B in order to solve Problem 1. Depending on (i) the available simulation environment for training and thus the quality of the trained policy, and (ii) the supplied robustness specifications which guarantee satisfaction of ϕ with $\rho^\phi \geq \rho_{\min}$, the base law is expected to offer much better guided exploration towards minimizing the objective. This leads to a significantly improved convergence rate and thus sample efficiency of the PI^2 algorithm during this online stage.

C. Further improvements

The proposed learning framework relies on *a priori* known robustness specifications $\rho^{\mu_i}(\boldsymbol{x}(t)) \geq \gamma_i(t)$ which guarantee achieving the desired task satisfaction with minimal robustness $\rho^\phi \geq \rho_{\min}$ in exchange for the increased sample efficiency offered by offline training. Here we highlight two possible directions for future research towards generating such specifications from specific STL formulas:

- Having trained a base law π_{NN}, the curves $\gamma_i(t)$ themselves can be treated as the input to the system instead of the actual input $\boldsymbol{u}$. These curves then define the achieved costs $C(\tau)$ and STL task satisfaction robustness metrics ρ^ϕ, thus their initial estimate can be further improved using a PI^2 framework to optimize for their values before the online stage begins.
- During the online stage, the specification curves $\gamma_i(t)$ should be continuously adjusted to follow the evolution of each ρ^{μ_i} in time along the currently most optimal discovered policy in order for them to keep offering relevant guidance for exploration in the PI^2 algorithm.

Pursuing these outlined directions will allow handling complex STL formulas, where the necessary robustness specifications are not evident to design. The novelty of our presented control synthesis strategy lies in the advantage and versatility it promises by efficiently adapting to various STL formulas using such robustness specifications. This motivates the proposed learning framework's potential practical applicability compared to trajectory history-based learning approaches.

V. Simulation study

The improved sample efficiency of using an offline trained neural network policy to guide exploration in PI^2 for solving STL tasks is demonstrated by the following simulation study.

A. Scenario description

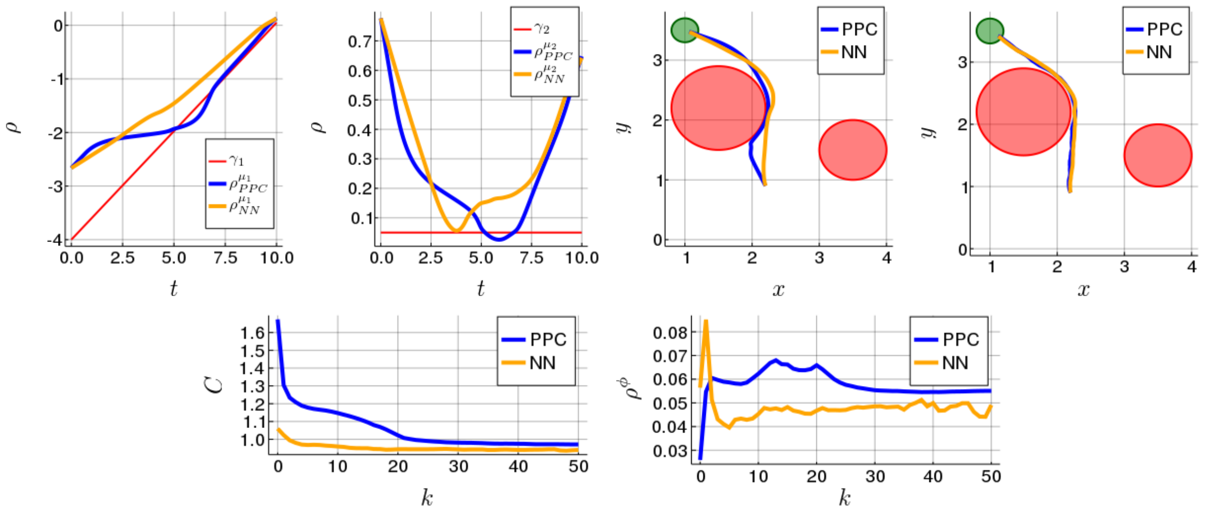
Consider an omnidirectional robot on a 2D plane with dynamics $\dot{\boldsymbol{x}} = \boldsymbol{u}$ with input constraint $\|\boldsymbol{u}\|_2 \leq 1$. The robot is presented with a navigational task of reaching a goal while avoiding two obstacles; these regions are all circular and centered at $\boldsymbol{x}_g = [1.0,\ 3.5]^{\mathrm{T}}$, $\boldsymbol{x}_{o1} = [1.5,\ 2.2]^{\mathrm{T}}$, $\boldsymbol{x}_{o2} = [3.5,\ 1.5]^{\mathrm{T}}$ with radii $r_g = 0.2$, $r_{o1} = 0.7$, and $r_{o2} = 0.5$, respectively. The corresponding atomic propositions μ_i are defined by their robustness metrics $h^{\mu_1}(\boldsymbol{x}) = r_g - \|\boldsymbol{x} - \boldsymbol{x}_g\|_2$, $h^{\mu_2}(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{x}_{o1}\|_2 - r_{o1}$, and $h^{\mu_3}(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{x}_{o2}\|_2 - r_{o2}$. The scenario is simulated for a time horizon $T = 10$s with a time step $\Delta t = 0.02$s.

While the obstacles must always be avoided, we examine two task descriptions regarding the goal region. The first requires the goal region to eventually be reached within 10s, whereas in the second one the agent must eventually always stay inside the goal region within 6s. Formally, the two tasks are defined as $\phi_1 = F_{[0,10]}\mu_1 \wedge G_{[0,\infty]}(\mu_2 \wedge \mu_3)$ and $\phi_2 = F_{[0,6]}G_{[0,\infty]}\mu_1 \wedge G_{[0,\infty]}(\mu_2 \wedge \mu_3)$ and the robot is required to satisfy either task with a minimal robustness degree $\rho_{\min} = 0.05$. To this end, valid robustness specifications $\gamma_1(t)$, $\gamma_2(t)$, and $\gamma_3(t)$ impose ρ^{μ_1} to become (and stay) above $\rho_{\min}$ within 10s (and 6s, respectively), as well as $\rho^{\mu_2}(t), \rho^{\mu_3}(t) \geq \rho_{\min}$ for $\forall t \geq 0$.

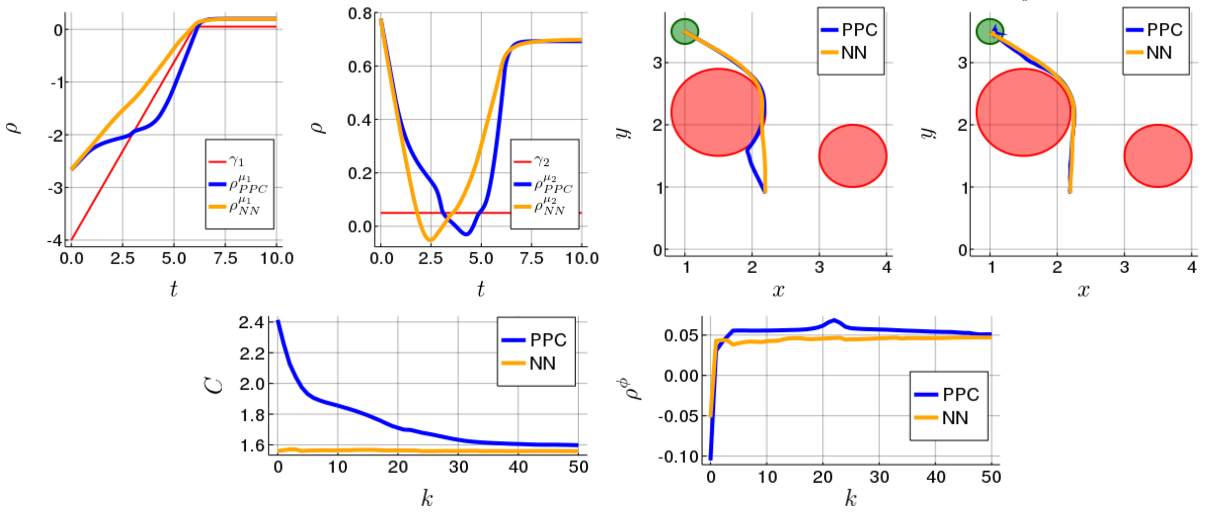
We also consider two costs of interest which the robot must minimize while satisfying its given STL task: $C_1(\tau) = \int_0^T \boldsymbol{u}(t)^{\mathrm{T}}\boldsymbol{u}(t)\mathrm{d}t$ and $C_2(\tau) = \int_0^T \min(0, \rho_{\min} - \rho^{\mu_1}(\boldsymbol{x}(t)))\mathrm{d}t$. The first minimizes the expended input energy, while the second the time spent outside the goal.

B. Training procedure

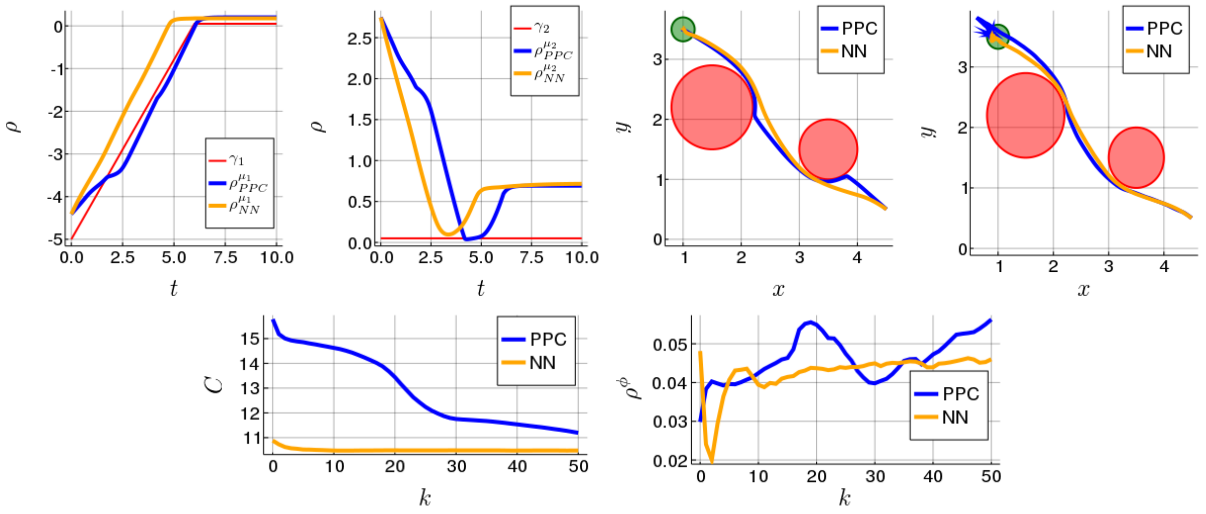
In the first offline stage, guiding base laws are trained to satisfy a variety of robustness specifications for the two separate cost functions by minimizing the objective (6) for either $C_1(\tau)$ or $C_2(\tau)$ in each case. The linear-sigmoid penalty function (7) was used to enforce the robustness specifications, parameterized by the values $\alpha = 0.05$, $\beta = 10$ and $\lambda = 10$; the curve $\Gamma_i(t)$ controlling where the



(a) $\boldsymbol{x}_0 = [2.2; 0.9]$, STL task $\phi_1 = F_{[0,10]}\mu_1 \wedge G_{[0,\infty]}(\mu_2 \wedge \mu_3)$, cost of interest $C_1(\tau) = \int_0^T \boldsymbol{u}^\mathrm{T}\boldsymbol{u}\mathrm{d}t$



(b) $\boldsymbol{x}_0 = [2.2; 0.9]$, STL task $\phi_2 = F_{[0,6]}G_{[0,\infty]}\mu_1 \wedge G_{[0,\infty]}(\mu_2 \wedge \mu_3)$, cost of interest $C_1(\tau) = \int_0^T \boldsymbol{u}^\mathrm{T}\boldsymbol{u}\mathrm{d}t$



(c) $\boldsymbol{x}_0 = [4.5; 0.5]$, STL task $\phi_2 = F_{[0,6]}G_{[0,\infty]}\mu_1 \wedge G_{[0,\infty]}(\mu_2 \wedge \mu_3)$, cost of interest $C_2(\tau) = \int_0^T \min(0, \rho_{\min} - \rho^{\mu_1}(\boldsymbol{x}(t)))\mathrm{d}t$

Fig. 2: Performance of PI^2 for solving STL tasks when using either a PPC-based ('PPC') or neural network based ('NN') base law for guiding exploration. Each scenario corresponds to a different initial condition or STL task; the figures from left to right show the initially imposed and achieved robustness specifications on μ_1 and μ_2, the initial and obtained system trajectories, and the convergence rates of the cost $C(\tau)$ and the STL task's robustness ρ^ϕ during the PI^2 iterations.

penalization begins was defined by $\Gamma_i(t) = \gamma_i(t) + 0.3$ for each atomic proposition μ_i. For both costs, the trained policy was given by a neural network with two fully dense, rectified linear unit (ReLU) activated hidden layers consisting of 64 neurons each. The inputs to each network are composed of the system state $\boldsymbol{x}$ and a low-dimensional representation of a 2s future horizon of the robustness specification curves $\boldsymbol{\gamma}(t)$ using its values at time instances t, $(t+1)$, and $(t+2)$. The output layers use $\tanh$ activations, aiming to structurally enforce the input bound $\|\boldsymbol{u}\|_2 \leq 1$.

To train the policies, first a multitude of sample problems were solved using PI^2 to minimize (6) from a grid of initial states spanning the intervals $x = [0, 5]$ and $y = [0, 4]$ in a step size of $1/3$. For each initial state $\boldsymbol{x}_0$, we defined 3 sample problems by choosing robustness specifications for μ_1 which attempt to drive the robot to reach the goal in a linear fashion. The 3 cases randomized the initial imposed distance to the goal and the imposed speed towards it, aiming not to deviate much from the actual initial distance and the maximum possible speed of the robot. The neural network was then trained on the obtained (and normalized) data using the machine-learning library Flux [17]. We employed stochastic gradient descent with restarts [19], in particular a cosine annealed learning rate restarted from 0.01 multiple times after 250 descent steps computed from batches of 1024 random data points. A dropout rate of 0.1 was used to further increase the effectiveness of the optimization procedure [20].

C. Results and discussion

The sample efficiency resulting from the pre-trained base laws is demonstrated by comparing the convergence rate of PI^2 when solving for the two STL task specifications ϕ_1 and ϕ_2 in the second, online stage. In each $K = 50$ iterations of PI^2, $N = 25$ trajectories were generated for exploration. The desired STL robustness $\rho_{\min} = 0.05$ was enforced with a penalty $P^\lambda(\rho^\phi) = \min(0, \lambda(\rho_{\min} - \rho^\phi)^3)$ where λ was increased from 200 to 5000 linearly throughout the iterations.

Fig. 2 presents a selection of results from the described simulation study. For each sample problem of examined cost $C_i(\tau)$ and task ϕ_i, we plot the imposed robustness specifications and corresponding initial trajectories, along with the obtained final trajectories and convergence rates of the PI^2 algorithm during the solution process. In each graph, results corresponding to the PPC-based ('PPC') and neural network ('NN') guiding laws are shown for comparison. The latter base law is seen to be significantly superior both in terms of the initially achieved costs and robustness metrics, as well as in terms of convergence rates and thus sample efficiency. The improvement depends on the training quality and can be expected to further increase for more complicated scenarios, provided that enough training data is used. We note that the presented (both PPC and NN) results could be further improved by adequate tuning of the PI^2 algorithm parameters or better training of the neural networks; however, the main take-away is to show the versatility and fundamental improvement in sample efficiency offered by our approach.

VI. CONCLUSIONS

A novel learning framework was presented for finding control policies for nonlinear systems which guarantee STL task satisfaction while minimizing a target cost of the trajectory. The approach relies on satisfying robustness specifications laid on the temporal evolution of atomic propositions composing the task. This allows offline training for a policy that is readily adaptable to other STL tasks real-time, as the gathered experience is not specific to a particular task. Compared to previous methods, this approach greatly improves sample efficiency during this adaptive online phase. Simulation results are promising and motivate further research into the proposed learning framework towards a practically feasible algorithm for STL task satisfaction.

REFERENCES

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

[2] M. Kloetzer and C. Belta, "LTL planning for groups of robots," in *IEEE International Conference on Networking, Sensing and Control*, 2006, pp. 578–583.

[3] L. Lindemann, C. K. Verginis, and D. V. Dimarogonas, "Prescribed performance control for signal temporal logic specifications," in *IEEE Conference on Decision and Control*, 2017, pp. 2997–3002.

[4] J. Fu and U. Topcu, "Probably approximately correct MDP learning and control with temporal logic constraints," *arXiv preprint arXiv:1404.7073*, 2014.

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

[6] D. Sadigh, E. S. Kim, S. Coogan, S. S. Sastry, and S. A. Seshia, "A learning based approach to control synthesis of MDPs for linear temporal logic specifications," in *IEEE CDC*, 2014, pp. 1091–1096.

[7] X. Li, C.-I. Vasile, and C. Belta, "Reinforcement learning with temporal logic rewards," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017, pp. 3834–3839.

[8] M. Guo and D. V. Dimarogonas, "Multi-agent plan reconfiguration under local LTL specifications," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 218–235, 2015.

[9] M. Wen, R. Ehlers, and U. Topcu, "Correct-by-synthesis reinforcement learning with temporal logic constraints," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015, pp. 4983–4990.

[10] 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.

[11] Q. Gao, D. Hajinezhad, Y. Zhang, Y. Kantaros, and M. M. Zavlanos, "Reduced variance deep reinforcement learning with temporal logic specifications," *International Conf. on Cyber-Physical Systems*, 2019.

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

[13] P. Varnai and D. V. Dimarogonas, "Prescribed performance control guided policy improvement for satisfying signal temporal logic tasks," in *IEEE American Control Conference*, 2019.

[14] 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.

[15] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *Journal of Machine Learning Research*, vol. 11(Nov), pp. 3137–3181, 2010.

[16] D. Muniraj, K. G. Vamvoudakis, and M. Farhood, "Enforcing signal temporal logic specifications in multi-agent adversarial environments: A deep Q-learning approach," in *IEEE CDC*, 2018, pp. 4141–4146.

[17] M. Innes, "Flux: Elegant machine learning with Julia," *Journal of Open Source Software*, 2018.

[18] H. Chen and F. Allgöwer, "A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability," *Automatica*, vol. 34, no. 10, pp. 1205–1217, 1998.

[19] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.