

Event-Based Signal Temporal Logic Synthesis for Single and Multi-Robot Tasks

David Gundana and Hadas Kress-Gazit

Abstract—We propose a new specification language and control synthesis technique for single and multi-robot high-level tasks; these tasks include timing constraints and reaction to environmental events. Specifically, we define Event-based Signal Temporal Logic (STL) and use it to encode tasks that are reactive to uncontrolled environment events. Our control synthesis approach to Event-based STL tasks combines automata and control barrier functions to produce robot behaviors that satisfy the specification when possible. Our method automatically provides feedback to the user if an Event-based STL task can not be achieved. We demonstrate the effectiveness of the framework through simulations and physical demonstrations of multi-robot tasks.

I. INTRODUCTION

High-level specifications have been used to describe complex robotics behaviors such as search and rescue missions and other planning and coordination tasks. Researchers have used control synthesis approaches to automatically generate controllers that satisfy high-level specifications described by temporal logic. Temporal logics such as Linear Temporal Logic (LTL) are synthesized into controllers [1] for single-robot systems, multi-robot systems (e.g. [2]–[4]), and swarms (e.g. [5], [6]). In other work, robot controllers have been synthesized for discrete-time continuous systems from Signal Temporal logic (STL) [7] and Metric Temporal Logic (MTL) specifications [8]. These specification languages can capture timing constraints associated with complex tasks [9]. In this paper we propose a specification formalism and associated control synthesis algorithm that combines the continuous (timing) properties of STL with the event-based nature of discrete logics, such as LTL, to enable users to specify tasks that have both timing constraints and desired reaction to external events.

Synthesizing STL: Authors of [10]–[12] present methods to design controllers for STL tasks. Work in [10] provides a framework for solving a fragment of STL for multi-robot tasks. This method is robust to robot attrition and used for large teams of robots; however, the control is calculated before execution therefore it is not robust to disturbances encountered at runtime. The control synthesis approaches of [11], [12] provide robustness to disturbances. These methods rely on solving computationally expensive mixed-integer linear programs. The computation complexity makes

it challenging to implement in real time, especially in the presence of dynamic obstacles.

The authors of [13] create control barrier functions (CBFs) and provide feedback control laws for a robot navigating in an environment with obstacles. These CBFs ensure that a system remains inside of a pre-defined set of allowable states, the safe-set, for all trajectories. [6], [14] leverage the work in [13] to create safe control for multi-robot systems and swarms.

The work in [15] uses time-varying CBFs to create a feedback control law that satisfies STL tasks for robotic systems in order to reduce the computational burden associated with solving mixed-integer linear programs. [16] extends [15] for multi-robot systems and introduces variables that relax CBFs and find a least violating solution when tasks conflict. Further, [17] creates a systematic procedure for constructing these CBFs to satisfy given STL tasks for multi-robot systems. In later work, [18] proposes a framework for satisfying STL tasks through automata based planning and timed signal transducers that represent temporal and Boolean operators [18]. We leverage [15]–[18] in our work and extend its capabilities to include tasks that require the robot to react to events in the environment.

Reactive STL: Researchers have investigated satisfying STL tasks that are reactive to external disturbances from the environment in order to encompass a larger set of complex tasks [19]. These reactive STL tasks have been satisfied using model predictive control solved through mixed-integer linear programs. Disturbances are bounded and the authors make assumptions about the behaviour of the environment and adversaries in [19]. In this paper, we propose a framework that considers these environment inputs to be discrete external events such as alarms and signals that have uncontrolled timings. To capture such tasks we create an extension of STL – Event-based STL – which can encode tasks where the robot must react to external events.

Assumptions: In this paper, we assume that the initial state of the robot and the initial state of the environment, that is whether environment events are triggered, do not violate the specification. In essence, we require that the system does not violate the specification before it starts executing. We also assume that all robots in the system are holonomic, meaning that the number of controllable degrees of freedom is equal to the total degrees of freedom. In multi-robot tasks we assume all robots can detect the state of the other robots, and that each robot computes its own control signal.

Contributions: We propose a framework for encoding tasks that contain timing constraints and reaction to environ-

This paper was recommended for publication by Editor Lucia Pallottino upon evaluation of the Associate Editor and Reviewers' comments.

D. Gundana and H. Kress-Gazit are with Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY, 14853 USA. {dog4,hadaskg}@cornell.edu. This work is supported by the National GEM Consortium, Cornell Sloan Fellowship, and NSF IIS-1830471.

Digital Object Identifier (DOI): see top of this page.

mental events, creating a control strategy to satisfy the task using CBFs, and providing feedback on the feasibility of these tasks. We present three main contributions: 1) a novel specification formalism, Event-based STL, that can capture tasks that cannot be expressed in current STL synthesis techniques, 2) an automata-based synthesis framework for generating decentralized controllers for multi-robot systems under an Event-based STL specification using time-varying CBFs, thus reducing the computational burden of current reactive STL approaches and 3) automated feedback to the user on the feasibility of Event-based STL tasks a-priori and at runtime for robots with bounded control inputs. Our approach is sound but not complete in that we may not find a control strategy even if one exists. We provide feedback to the user in such cases.

II. PRELIMINARIES

A. Signal Temporal Logic (STL)

Consider a continuous time dynamical system representing robot motion:

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

Where $\mathbf{x} \in \mathbb{R}^n$ is the state of the system, $\mathbf{u} \in \mathbf{U} \subseteq \mathbb{R}^m$ is the bounded control input of the system, and f and g are locally Lipschitz continuous functions.

Let $\mu \in \{True, False\}$ represent a predicate whose truth value is defined by the evaluation of a predicate function $h(\mathbf{x}_t)$ where \mathbf{x}_t is the state of the system at time t .

$$\mu ::= \begin{cases} False & \Rightarrow h(\mathbf{x}_t) < 0 \\ True & \Rightarrow h(\mathbf{x}_t) \geq 0 \end{cases} \quad (2)$$

Syntax: An STL formula ϕ is defined recursively as

$$\phi ::= True \mid \mu \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid F_{[a,b]}\phi \mid G_{[a,b]}\phi \mid \phi_1 U_{[a,b]}\phi_2 \quad (3)$$

where ϕ is an STL formula, $a, b \in \mathbb{R}^+$ are timing bounds, \neg is “not”, \wedge is “and”, F is “eventually”, G is “always”, and U is “Until” [9].

Semantics: The semantics of STL are evaluated over trajectories of the dynamical system in eqn. 1:

$$\begin{aligned} \mathbf{x}_t \models \mu & \Leftrightarrow h(\mathbf{x}_t) \geq 0 \\ \mathbf{x}_t \models \neg\phi & \Leftrightarrow \mathbf{x}_t \not\models \phi \\ \mathbf{x}_t \models \phi_1 \wedge \phi_2 & \Leftrightarrow \mathbf{x}_t \models \phi_1 \text{ and } \mathbf{x}_t \models \phi_2 \\ \mathbf{x}_t \models F_{[a,b]}\phi & \Leftrightarrow \exists t_1 \in [t+a, t+b] \text{ s.t. } \mathbf{x}_{t_1} \models \phi \\ \mathbf{x}_t \models G_{[a,b]}\phi & \Leftrightarrow \forall t_1 \in [t+a, t+b], \mathbf{x}_{t_1} \models \phi \\ \mathbf{x}_t \models \phi_1 U_{[a,b]}\phi_2 & \Leftrightarrow \exists t_2 \in [t+a, t+b] \text{ s.t. } \mathbf{x}_{t_2} \models \phi_2 \\ & \text{and } \forall t_1 \in [t+a, t_2], \mathbf{x}_{t_1} \models \phi_1 \end{aligned}$$

Intuitively, $F_{[a,b]}\phi$ is *True* if there exists a time between a and b where ϕ is *True*, $G_{[a,b]}\phi$ is *True* if ϕ is *True* for all time between a and b , and $\phi_1 U_{[a,b]}\phi_2$ is *True* if ϕ_1 is *True* from time a until ϕ_2 becomes *True*, and ϕ_2 becomes *True* sometime in the interval $[a, b]$.

B. CBFs for STL specifications

CBFs were proposed in [20] and used to define safe-sets for a system and ensure that the safe-set is forward invariant: if a system starts in the set it will always stay in that set. CBFs ensure forward invariance without determining the entire reachable set of system. Lindemann and Dimarogonas [15] propose a process to generate control for robotic systems to satisfy STL formulas using CBFs. To ensure a task is satisfied given the timing constraints of an STL formula, [15] creates CBFs $cbf(\mathbf{x}_t)$ that are time varying and forward invariant. These CBFs are constructed using predicate functions of the STL formula, $h(\mathbf{x}_t)$. Given a CBF, if eqn. 4 holds for all \mathbf{x}_t , then the system is forward invariant, i.e. if $cbf(\mathbf{x}_0) \geq 0$, then $cbf(\mathbf{x}_t) \geq 0$ for all t .

$$\sup_{\mathbf{u} \in \mathbf{U}} \frac{\partial cbf(\mathbf{x}_t)^T}{\partial \mathbf{x}} (f(\mathbf{x}_t) + g(\mathbf{x}_t)\mathbf{u}_t) + \frac{\partial cbf(\mathbf{x}_t)}{\partial t} \geq -\nu(cbf(\mathbf{x}_t)) \quad (4)$$

where $\nu : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ is a locally Lipschitz continuous class K function.

In [15] III.A, the authors describe in detail the process for generating CBFs such that if we can find a control input u that satisfies eqn. 4, the system is guaranteed to satisfy the associated STL formulas with timing constraints. Intuitively, we require (i) the CBF to be valid, i.e. satisfy $cbf(\mathbf{x}_t) \geq 0, \forall t$. To ensure the timing constraints in the STL formulas are met, we require (ii) $cbf(\mathbf{x}_t) \leq h(\mathbf{x}_t)$ based on predicate μ and the associated temporal operator; at some t in the interval for $F_{[a,b]}$, for all t in the interval for $G_{[a,b]}$, etc. The combination of these constraints guarantees $h(\mathbf{x}_t) \geq 0$ at the required times thus satisfying the STL formula.

We can combine STL formulas and create controllers that do not violate any of the individual CBFs in order to express more complex tasks. This is done using an approximation for the minimum of the barrier functions for each task. One can design a single CBF, cbf_ϕ , such that if $cbf_\phi \geq 0$, then $cbf_i \geq 0 \forall i$ [15]:

$$cbf_\phi = -\ln \left(\sum_{i=1}^I \exp(-cbf_i(\mathbf{x}_t)) \right) \quad (5)$$

where I is the number of CBFs in a given specification.

C. Linear Temporal Logic (LTL) and Büchi Automata

An LTL formula γ is constructed from a set of atomic propositions AP using the following grammar

$$\gamma ::= \pi \mid \neg\gamma \mid \gamma_1 \wedge \gamma_2 \mid X\gamma \mid \gamma_1 U \gamma_2 \quad (6)$$

where $\pi \in AP$, \neg and \wedge are the Boolean operators “not” and “and”, X is the temporal operator “next”, and U is the temporal operator “Until”. From these operators we can define the temporal operators “eventually” ($F\gamma = True U \gamma$) and “always” ($G\gamma = \neg F \neg \gamma$). The semantics of LTL are defined over an infinite sequence $\sigma = \sigma_1, \sigma_2, \dots$, where $\sigma_i \subseteq AP$ represents the set of propositions that are *True* in position i of the sequence.

Intuitively, $X\gamma$ is *True* if for every execution γ is *True* in the next position of the sequence, $F\gamma$ is *True* if for every

execution γ is *True* at some position in the sequence, $G\gamma$ is *True* if for every execution γ is *True* at all positions of the sequence, and $\gamma_1 U \gamma_2$ is *True* if for every execution γ_1 is *True* until γ_2 becomes *True*.

A nondeterministic Büchi automaton is a tuple

$$B = (S, s_0, \Sigma, \delta, F) \quad (7)$$

where S is a finite set of states, s_0 is the initial state, Σ is a finite input alphabet, $\delta \subseteq S \times \Sigma \times S$ is the transition relation, and $F \subseteq S$ is a set of accepting states. A run of a Büchi automaton on input word $\omega = \omega_1, \omega_2, \dots$, $\omega_j \in \Sigma$ is an infinite sequence of states s_0, s_1, s_2, \dots s.t. $\forall j \geq 1, (s_{j-1}, \omega_j, s_j) \in \delta$. We define $\text{inf}(\omega)$ as a set of states that are visited infinitely often on the input word ω . A run is accepting iff $\text{inf}(\omega) \cap F \neq \emptyset$.

Given an LTL formula γ , one can construct a nondeterministic Büchi automaton B_γ such that B_γ only accepts input words that satisfy γ [21], [22]. In the following we use the LTL to Büchi automaton tool Spot [23].

III. EVENT-BASED STL

We define a new specification formalism, Event-based STL, to describe tasks that have not been previously addressed by STL synthesis techniques. This formalism can capture tasks where the system needs to react to uncontrolled environmental events that may or may not occur during execution. Examples of these events are fire alarms in an evacuation scenario, a person entering in a room in a workspace environment, or a command from a user.

A. System Representation

The system model is defined by eqn. 1. In addition to the system model, we consider discrete environmental events. These environmental events are uncontrolled by the system and are represented as Boolean propositions $\pi \in AP$. We define $\sigma_t \subseteq AP$ as the set of atomic propositions that are *True* at time t .

B. Syntax of Event-Based STL

We define Event-based STL formulas Ψ as follows:

$$\mu ::= \mu \mid \neg \mu \mid \varphi_1 \wedge \varphi_2 \quad (8)$$

$$\alpha ::= \pi \mid \neg \pi \mid \alpha_1 \wedge \alpha_2 \quad (9)$$

$$\Psi ::= G_{[a,b]} \varphi \mid F_{[a,b]} \varphi \mid \varphi_1 U_{[a,b]} \varphi_2 \mid G(\alpha \Rightarrow \Psi) \mid G(\varphi \Rightarrow \Psi) \mid \Psi_1 \wedge \Psi_2 \quad (10)$$

where μ is a predicate representing $h(\mathbf{x}_t)$ as described in eqn. 2, α is a Boolean formula over environment propositions $\pi \in AP$, \Rightarrow is the implication operator, and the temporal operators follow the conventions of STL, as defined in Sec. II-A. If the "always" operator G does not contain a timing bound $[a, b]$, we assume the timing bound is $[0, \infty]$. We interpret the interval $[a, b]$ in continuous time similar to the fragment of STL in [15]. When executing the control on a simulated or physical system with a clock, we evaluate the dynamics of the system and the environment at a set sampling rate. For demonstrations we assume that the interval $[a, b]$ is a multiple of this sampling rate and environment events last longer than the sampling period.

C. Semantics of Event-Based STL

We define the semantics of Event-based STL over (\mathbf{x}_t, σ_t) where \mathbf{x}_t is the state of the system at time t and σ_t is a set of environment propositions that are *True* at time t .

$$\begin{aligned} (\mathbf{x}_t, \sigma_t) \models \mu & \Leftrightarrow h(\mathbf{x}_t) \geq 0 \\ (\mathbf{x}_t, \sigma_t) \models \neg \mu & \Leftrightarrow h(\mathbf{x}_t) < 0 \\ (\mathbf{x}_t, \sigma_t) \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow (\mathbf{x}_t, \sigma_t) \models \varphi_1 \text{ and } (\mathbf{x}_t, \sigma_t) \models \varphi_2 \end{aligned}$$

$$\begin{aligned} (\mathbf{x}_t, \sigma_t) \models \pi & \Leftrightarrow \pi \in \sigma_t \\ (\mathbf{x}_t, \sigma_t) \models \neg \alpha & \Leftrightarrow (\mathbf{x}_t, \sigma_t) \not\models \alpha \\ (\mathbf{x}_t, \sigma_t) \models \alpha_1 \wedge \alpha_2 & \Leftrightarrow (\mathbf{x}_t, \sigma_t) \models \alpha_1 \text{ and } (\mathbf{x}_t, \sigma_t) \models \alpha_2 \end{aligned}$$

$$\begin{aligned} (\mathbf{x}_t, \sigma_t) \models F_{[a,b]} \varphi & \Leftrightarrow \exists t_1 \in [t+a, t+b] \text{ s.t. } (\mathbf{x}_{t_1}, \sigma_{t_1}) \models \varphi \\ (\mathbf{x}_t, \sigma_t) \models G_{[a,b]} \varphi & \Leftrightarrow \forall t_1 \in [t+a, t+b], (\mathbf{x}_{t_1}, \sigma_{t_1}) \models \varphi \\ (\mathbf{x}_t, \sigma_t) \models \varphi_1 U_{[a,b]} \varphi_2 & \Leftrightarrow \exists t_2 \in [t+a, t+b] \text{ s.t. } (\mathbf{x}_{t_2}, \sigma_{t_2}) \models \varphi_2 \text{ and } \forall t_1 \in [t+a, t_2], (\mathbf{x}_{t_1}, \sigma_{t_1}) \models \varphi_1 \\ (\mathbf{x}_t, \sigma_t) \models G(\alpha \Rightarrow \Psi) & \Leftrightarrow \forall t, (\mathbf{x}_t, \sigma_t) \not\models \alpha \text{ or } (\mathbf{x}_t, \sigma_t) \models \Psi \\ (\mathbf{x}_t, \sigma_t) \models G(\varphi \Rightarrow \Psi) & \Leftrightarrow \forall t, (\mathbf{x}_t, \sigma_t) \not\models \varphi \text{ or } (\mathbf{x}_t, \sigma_t) \models \Psi \\ (\mathbf{x}_t, \sigma_t) \models \Psi_1 \wedge \Psi_2 & \Leftrightarrow (\mathbf{x}_t, \sigma_t) \models \Psi_1 \text{ and } (\mathbf{x}_t, \sigma_t) \models \Psi_2 \end{aligned}$$

IV. PROBLEM FORMULATION

Problem: Given a dynamical system (eqn. 1) and its state \mathbf{x} , environment events AP , and an Event-based STL formula Ψ , find control u such that $(\mathbf{x}_0, \sigma_0) \models \Psi$, if possible.

We describe our approach to synthesizing the control in Section V, and discuss feedback and guarantees in Section VI. For multi-robot tasks, we propose a decentralized control strategy where each robot satisfies the Event-based STL formulas that affect them. We do this in order to distribute the computational burden associated with finding control inputs for each robot. In this framework each robot knows the position of the other robots in the system, but not their control inputs or goals.

A. Examples

Single-Robot Example: We consider a holonomic robot operating in an obstacle-free workspace. The robot's motion is described by eqn. 1 where $\mathbf{x}_t \in \mathbb{R}^2$ is the state of the robot $[x_t, y_t]^T$ at time t , $f(\mathbf{x}_t) = 0$, $g(\mathbf{x}_t) = I_2$, and \mathbf{u}_t is the control input $[u_{x_t}, u_{y_t}]^T$. We define $AP = \{\text{alarm}\}$ as the set of environment events. The robot's task is whenever it senses the alarm, to arrive, within 10 time steps, at a point within 1 unit from $[5, 5]$. The task is captured by the Event-based STL formula

$$\Psi = G(\text{alarm} \Rightarrow F_{[0,10]}(\|\mathbf{x} - [5, 5]^T\| < 1))$$

Here, $h(\mathbf{x}_t) = (1 - \|\mathbf{x}_t - [5, 5]^T\|)$.

Multi-Robot Example: We consider four holonomic robots operating in an obstacle-free workspace. The dynamics of the robots are described by eqn. 1, where \mathbf{x} describes the state of the robots $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4]$, $\mathbf{x}_i = [x_{i,t}, y_{i,t}, \theta_{i,t}]$, $f(\mathbf{x}_i) = 0$, $g(\mathbf{x}_i) = I_3$, $\mathbf{u} = [\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4]$, and $\mathbf{u}_i = [u_{i,x_t}, u_{i,y_t}, u_{i,\theta_t}]$ for each robot i .

We define $AP = \{\text{approach}, \text{align}\}$. The multi-robot task is captured by the following Event-based STL formula $\Psi =$

$\Psi_1 \wedge \Psi_2 \wedge \Psi_3 \wedge \Psi_4 \wedge \Psi_{collision} \wedge \Psi_{approach} \wedge \Psi_{align}$ where the subformulas are

- $\Psi_1 = F_{[0,10]}(\| \mathbf{x}_1 - [3, 1]^T \| < 0.5)$
- $\Psi_2 = F_{[5,15]}(\| \mathbf{x}_2 - [3, 2]^T \| < 0.5)$
- $\Psi_3 = F_{[0,10]}(\| \mathbf{x}_3 - [3, 0]^T \| < 0.5)$
- $\Psi_4 = F_{[0,10]}(\| \mathbf{x}_4 - [3, 2]^T \| < 0.5)$
- $\Psi_{collision_{ij}} = G_{[0,30]}(\| \mathbf{x}_i - \mathbf{x}_j \| > 0.3), \forall i \neq j$
- $\Psi_{approach_i} = G(approach \Rightarrow F_{[0,10]}(\| \mathbf{x}_i - [6, 2]^T \| < 1)), i = 1, 3$
- $\Psi_{align_i} = G(align \Rightarrow F_{[0,10]}(\| \theta_i - \pi \| < 0.1)), i = 2, 4$

Subformulas $\Psi_{1,2,3,4}$ describe when the robots should be in a certain region. $\Psi_{collision_{ij}}$ describes six subformulas for collision avoidance which states that each robot must maintain a distance of at least 0.3 units from every other robot. $\Psi_{approach_i}$ states that, for robots 1 and 3, if the environment event *approach* is sensed, then they should arrive close to $[6, 2]$ (no more than 1 away) within 10 time units. Ψ_{align_i} states that, for robots 2 and 4, if the environment event *align* is sensed, they both should, within 10 time units, be facing the -x direction of the global reference frame.

V. SYNTHESIS FOR EVENT-BASED STL

Algo. 1 describes our approach to automatically synthesizing control given a high-level task encoded in Event-based STL. The inputs to this algorithm are an Event-based STL formula Ψ_{STL} , the number of robots n , σ_t , \mathbf{x}_t , and the functions $h_i(\mathbf{x}_t)$. The outputs are the control inputs $\mathbf{u}_i \in \mathbf{U}_i$ for each robot, that satisfy Ψ_{STL} .

Algo. 1 has two phases; first, before execution, we create template CBFs based on the predicates in Ψ and create a Büchi automaton that we use to temporally compose CBFs based on environmental events (Section V-A). Then, during execution, we choose a transition in the Büchi automaton that corresponds to the currently sensed events in the environment (Section V-B) and create the control from the CBFs that correspond to that transition (Section V-C.) The Büchi automaton acts as a centralized symbolic planner. Given a transition in the Büchi automaton, each robot determines their own control strategy in a decentralized manner.

A. CBFs and Abstracted Automaton

Given an Event-based STL formula Ψ_{STL} we first create template CBFs corresponding to the predicates in Ψ_{STL} . We then abstract the formula into an LTL formula Ψ_{LTL} and create a Büchi automaton $B_{\Psi_{LTL}}$ that we use to choose the CBFs that are executed.

Creating CBF templates cbf_{μ_i} (Line 1 of Algo. 1): Given an STL formula ϕ , Lindemann et. al. [15] provide a method for constructing CBFs that satisfy time constrained STL specifications assuming unbounded control. Here, we use the methods from [15] to create CBF formula templates that use parameters from an Event-based STL formula.

We create a control barrier function template, eqn. 11, that changes linearly with time and utilizes the entirety of

Algorithm 1: Control synthesis for Event-based STL

Input : $\Psi_{STL}, n, \sigma_t, \mathbf{x}_t, h_i(\mathbf{x}_t)$,
Output: \mathbf{u}

```

1  $\forall i, cbf_{\mu_i} = CBFTemplate(h_i(\mathbf{x}_t));$ 
2  $(\Psi_{LTL}, \Pi_{\mu}) = STL2LTL(\Psi_{STL});$ 
3  $B_{\Psi_{LTL}} = LTL2Buchi(\Psi_{LTL});$ 
4  $currS = s_0;$ 
5  $\sigma_{-1} = \sigma_0;$ 
6  $(\sigma_{currS, nextS}, \Pi_{\mu_{act}}, currS) =$ 
    $findTransition(\sigma_0, \mathbf{x}_0, B_{\Psi_{LTL}}, h_i(\mathbf{x}_0), currS);$ 
7 while True do
   // check whether reached nextS or
   // environment event changed
8   if  $(\sigma_{currS, nextS}$  is True) or  $\sigma_t \neq \sigma_{t-1}$  then
9      $(\sigma_{currS, nextS}, \Pi_{\mu_{act}}, currS) =$ 
        $findTransition(\sigma_t, \mathbf{x}_t, B_{\Psi_{LTL}}, h_i(\mathbf{x}_t), currS);$ 
10  end
   // Execute Barrier Functions
11  for  $i = 1$  to  $n$  do
12     $\mathbf{u}_i = Barrier(\Pi_{\mu_{act}}, t, \mathbf{x}_t);$ 
13  end
14  if Eqn. 18 is infeasible then
15    Stop;
16  end
17   $\sigma_{t-1} = \sigma_t;$ 
18 end
```

the time bound that is given. For this template we use the predicate function $h_i(\mathbf{x}_t)$, the time that the CBF is initially activated t_{int} , and a, b as place holders for the exact timing bounds specified in the subformulas of Ψ_{STL} ; these timing bounds will be instantiated during execution (Sec. V-C).

$$cbf_{\mu_i}(\mathbf{x}_t) = \frac{(t - t_{int} - a)h_i(\mathbf{x}_{t_{int}})}{b - a} - h_i(\mathbf{x}_{t_{int}}) + h_i(\mathbf{x}_t) \quad (11)$$

We create CBFs in this way so that a robot has the greatest opportunity to satisfy its task. The CBF changing linearly and using the entire time bound represents a worst-case scenario of the safe-set at a point in time. At $t = t_{int} + a$, the initial time the CBF becomes activated, $cbf_{\mu_i}(\mathbf{x}_t) = 0$. At $t = t_{int} + b$, the final time in the interval for the Event-based STL formula, $cbf_{\mu_i}(\mathbf{x}_t) = h_i(\mathbf{x}_t)$.

For example, the predicate from the single robot example is $\mu_1 = \| \mathbf{x} - [5, 5]^T \| < 1$. We form a predicate function $h_1(\mathbf{x}) = 1 - \| \mathbf{x} - [5, 5]^T \|$ and construct a template CBF $cbf_{\mu_1}(\mathbf{x}_t) = \frac{(t - t_{int} - a)(1 - \| \mathbf{x}_{t_{int}} - [5, 5]^T \|)}{b - a} + \| \mathbf{x}_{t_{int}} - [5, 5]^T \|$.

Abstracting Ψ_{STL} (Line 2 of Algo. 1): We abstract Ψ_{STL} to Ψ_{LTL} by replacing $F_{[a,b]}$ with F , $G_{[a,b]}$ with G , and $U_{[a,b]}$ with U . Furthermore, we replace each μ_i with a proposition $\pi_{\mu_i, [a,b]} \in \Pi_{\mu}$ that we consider a controllable proposition. Each controllable proposition maintains the timing associated with its Event-based STL subformula. For example, eqn. 12

is abstracted to eqn. 13.

$$\Psi_{STL} = G(alarm \Rightarrow F_{[0,10]}(\|\mathbf{x} - [5, 5]^T\| < 1)) \quad (12)$$

$$\Psi_{LTL} = G(alarm \Rightarrow F(\pi_{\mu_1, [0,10]})) \quad (13)$$

where $\pi_{\mu_1, [0,10]}$ replaces $(\|\mathbf{x} - [5, 5]^T\| < 1)$ and the associated timing constraints. Event-based STL allows limited nesting of temporal operators through the formulas with implications, as long as the antecedent is either a Boolean combination of environment propositions, or a conjunction of predicates and their negation. For example, eqn. 14 is abstracted to eqn. 15

$$\Psi_{STL} = G(\mathbf{A} \Rightarrow G(\mathbf{B} \Rightarrow F_{[0,10]}(\|\mathbf{x} - [5, 5]^T\| < 1))) \quad (14)$$

$$\Psi_{LTL} = G(\mathbf{A} \Rightarrow G(\mathbf{B} \Rightarrow F(\pi_{\mu_1, [0,10]}))) \quad (15)$$

Where \mathbf{A} and \mathbf{B} are external environment events. We do not allow nesting of any other form.

Generating $B_{\Psi_{LTL}}$ (Line 3 of Algo. 1): We create a Büchi automaton $B_{\Psi_{LTL}}$ from Ψ_{LTL} using [23]. The transitions are labeled with Boolean formulas over the set $AP \cup \Pi_\mu$ as seen in figure 1. We denote a Boolean formula over $AP \cup \Pi_\mu$ representing the label of the transition between s_i and s_j as σ_{s_i, s_j} , i.e. $\Sigma = \{\sigma_{s_i, s_j} \mid \exists s_i, s_j \in S, (s_i, \sigma_{s_i, s_j}, s_j) \in \delta\}$

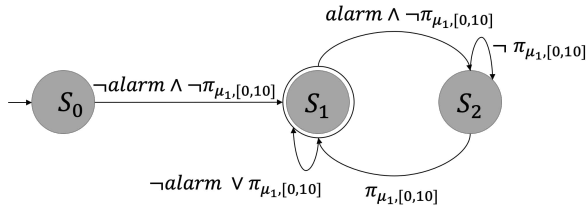


Fig. 1: Graphical representation of $B_{\Psi_{LTL}}$ for Ψ_{LTL} in eqn. 13. The grey circles represent states and the double circle represents an accepting state. Transitions between states are labeled with the Boolean formulas σ_{s_i, s_j} .

Here $S = \{s_0, s_1, s_2\}$, s_0 is the initial state, and s_1 is an accepting state. The transitions between states are labeled with Boolean formulas over $\{alarm, \pi_{\mu_1, [0,10]}\}$. For example, $\sigma_{s_0, s_1} = \neg alarm \wedge \neg \pi_{\mu_1, [0,10]}$. The task is satisfied when the system is in the accepting state and can remain in a cycle that contains an accepting state, in this case when the predicate μ_1 is *True* or the alarm is not activated. At runtime, based on the environment events and state of the system, we choose the next transition in the automaton, and then create the control to drive the robot(s).

B. Choosing transitions

Determining $\Pi_{\mu_{act}}$ (Lines 6 and 8 of Algo. 1): During execution, we create the control for the robot(s) based on the label of the active transition in $B_{\Psi_{LTL}}$. The active transition is the transition the system is currently trying to take, by activating the CBFs associated with the controllable propositions Π_μ .

At each time step, given σ_t , the set of environment propositions that are *True*, and the state of the system \mathbf{x}_t ,

we first determine the truth values of all the propositions $AP \cup \Pi_\mu$; for $\pi \in AP$:

$$\pi = \begin{cases} False & \text{if } \pi \notin \sigma_t \\ True & \text{if } \pi \in \sigma_t \end{cases} \quad (16)$$

and for $\pi_{\mu_i, [a,b]} \in \Pi_\mu$:

$$\pi_{\mu_i, [a,b]} = \begin{cases} False & \text{if } h_i(\mathbf{x}_t) < 0 \\ True & \text{if } h_i(\mathbf{x}_t) \geq 0 \end{cases} \quad (17)$$

We then evaluate whether we need to find a new active transition; this would happen under two conditions, either (1) the environment propositions changed, i.e. $\sigma_t \neq \sigma_{t-1}$ which could change the truth value of the formula labeling the transition $\sigma_{currS, nextS}$, or (2) $\sigma_{currS, nextS}$ becomes *True* indicating that all the associated predicates μ_i are *True* and the system transitioned to the next state.

If one of the above conditions holds, we choose a new active transition. To choose one, we first find the set of possible transitions. The system can choose to take transitions that are consistent with the current truth value of the (uncontrollable) environment propositions AP . Put another way, the set of possible transitions excludes transitions where the truth values of the propositions in AP would cause σ to evaluate to *False*.

Given the set of possible transitions, we search for a path that will cause the LTL specification to be satisfied. Specifically, we find a *prefix* – a path to an accepting state and a *suffix* – a cycle that will cause the system to infinitely visit accepting states; the *suffix* may be a self transition on an accepting state. If there are multiple paths, we choose the shortest prefix path; that being said, due to the structure of the formulas we allow (no disjunction over predicates or temporal formulas, no formulas of the form FG), and as is the case with all the examples in this paper, the Büchi automaton $B_{\Psi_{LTL}}$ is typically deterministic; therefore, the choice in paths is due to the possible different values of the controlled propositions. When deciding on the next state, if there are multiple paths of the same length, we choose the path where the next transition has the lowest number of $\pi_{\mu_i, [a,b]}$ that are *True*, thereby reducing the number of CBFs we need to activate.

Given the path, we choose as the active transition the next transition in this path. We denote the set of $\pi_{\mu_i, [a,b]}$ propositions that must be *True* to satisfy the Boolean formula σ_{currS, s_j} for this transition as $\Pi_{\mu_{act}}$; this set represents the CBFs that are activated for that transition to complete.

Using the example, if the system is in state s_2 , the only path to an accepting state is from state s_2 to state s_1 . For this transition to occur, the controllable proposition $\pi_{\mu_1, [0,10]} = True$ therefore $\pi_{\mu_1, [0,10]} \in \Pi_{\mu_{act}}$. The activated CBF will progress the system towards the accepting state s_1 . The system will not reach s_1 until $h_1(\mathbf{x}_t)$, the predicate function associated with $\pi_{\mu_1, [0,10]}$, becomes non-negative.

C. Control synthesis

Finding Control input \mathbf{u} (Line 12 of Algo. 1): Given the set of propositions $\Pi_{\mu_{act}}$, we activate CBFs that are associated

with those propositions for each robot i . Using the time at which a CBF is activated and the position of the robots at time t , we activate each pre-constructed barrier function (Section V-A) corresponding to $\Pi_{\mu_{act}}$ if t is in the interval $[t_{int} + a, t_{int} + b]$. The optimization problem we solve to find the control for each robot is shown in eqn. 18 where cbf_{Ψ_i} is the combination of all activated CBFs for robot i in the system found from eqn. 5. Eqn. 18 describes the optimization problem where a control law u_i is found that ensures that $cbf_{\Psi_i}(\mathbf{x}_t) \geq 0 \forall t$.

$$\min_{\mathbf{u}_i \in \mathbf{U}_i} \|\mathbf{u}_i - \hat{\mathbf{u}}_i\| \text{ s.t. } \frac{\partial cbf_{\Psi_i}(\mathbf{x}_t)^T}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}) + \frac{\partial cbf_{\Psi_i}(\mathbf{x}_t)}{\partial t} \geq -\nu(cbf_{\Psi_i}(\mathbf{x}_t)) \quad (18)$$

where the nominal controller for each robot $\hat{\mathbf{u}}_i$ is determined using the safe-set associated with the activated CBF. We find a goal point within the safe-set and choose $\hat{\mathbf{u}}_i$ in the direction of the goal point with a magnitude equal to the maximum control input for the robot. If the optimization problem is not feasible, it means we cannot find a control input that satisfies the specification and we stop the execution and provide feedback to the user.

Figure 2 shows the trajectory of an execution of the motivating example. The safe-set associated with the CBF is represented as a circle at time t_0 when the proposition *alarm* becomes *True*, and at $t_{final} = t_0 + 10$.

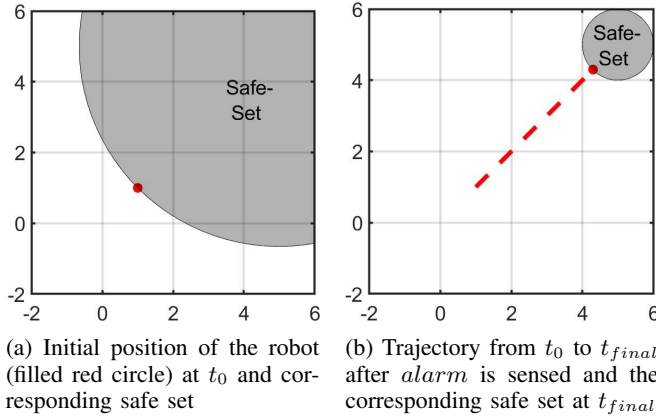


Fig. 2: Safe sets associated with the CBF and trajectory of the robot at the time when the robot senses *alarm* (t_0) and at t_{final} for the single-robot example

VI. FEEDBACK AND GUARANTEES

We give several forms of feedback regarding the feasibility of satisfying a task given the Event-based STL specification and the properties of the robots such as their dynamics and control bounds. We classify the feedback as *a priori* feedback and run-time feedback.

A priori feedback: We provide two types of feedback based solely on the specification: (i) possibly conflicting CBFs and (ii) inadmissible environment behaviors. For the possibly conflicting CBF, we examine the set $\Pi_{\mu_{act}}$ of each transition in $B_{\Psi_{LTL}}$. If, for a given transition, the

sets $h_i(\mathbf{x}_t)$ associated with $\Pi_{\mu_{act}}$ are non-intersecting, we provide feedback to the user that there might not be a control input that satisfies all the associated CBFs. This feedback is conservative as it does not take into account the timing of the STL formulas; depending on the timing, the task may or may not be feasible.

We place no assumptions on the timing of the environment events; however, depending on the specifications, there might be event timings that cause the LTL formula to be unsatisfiable. To alert the user to such situations, for each state, we examine the possible transitions out of the state and find combinations of environment events that are not allowed i.e., none of the transitions out of the state allows such a combination. We provide these combinations to the user.

Run-time feedback: Unknown disturbances such as other robots in the system, environment disturbances, or deadlocks can prevent the system from completing the task. We provide feedback on the feasibility of satisfying a task during an execution given the configuration of the system, the timing requirements, and the control bounds of the robots.

At each time-step, we calculate how far the system is from satisfying the predicate functions by evaluating $h(\mathbf{x}_t)$. We then compare this distance to the largest distance the system can move given the bounds on the control and the time remaining to satisfy the predicate $\|\mathbf{u}_{max}\| (b + t_{int} - t)$.

If the distance to the predicate function is larger than the maximum distance the system can travel, it means the system will fail the task and we provide feedback to the user. In this situation a solution to the optimization problem may still be found even though the specification will be violated at a future time. While for the CBF template we present in this paper this is not the case, other templates may define larger safe-sets that shrink faster, leading to the robot not being able to remain in the safe-set in the future.

We check the distance from each individual predicate; however, even if all predicates are within reach, when combining several CBFs in the optimization problem eqn. 18, it may become infeasible. This might happen when the system is trying to reach two predicates that require motion in opposite direction. In these cases, we stop the system and provide feedback to the user.

Completeness: Our specification formalism and control synthesis approach enable a user to capture complex tasks and automatically execute them; however, our approach is not complete. There may be a control that could satisfy the task in situations in which we have determined that we cannot find a control input. This can occur, for example, when there are conflicting predicates that may be achieved by adding a prioritization scheme. The feedback we are providing before and during execution is meant to mitigate possible execution failures.

VII. SIMULATION RESULTS

A. Simulation Example Description

We consider 4 holonomic robots that operate in a shared environment. They are performing the multi-

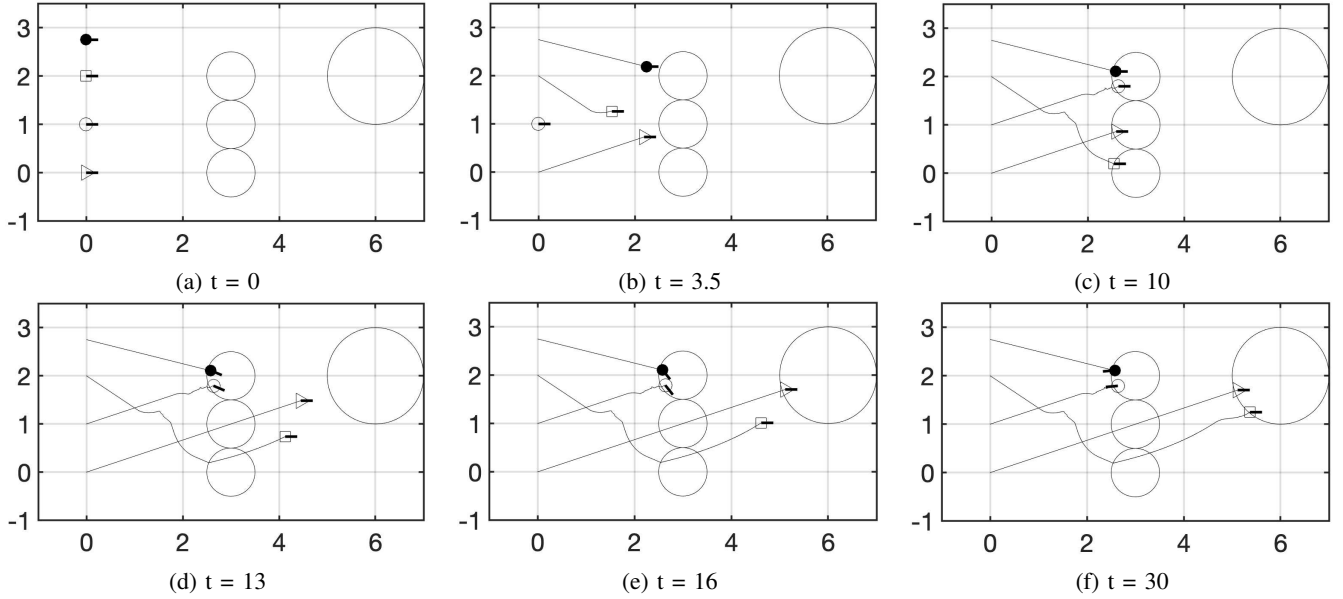


Fig. 3: Figure 3a shows the initial position of robot 1 (triangle), robot 2 (unfilled circle), robot 3 (square), and robot 4 (filled circle). Figure 3b shows robots 1, 3, and 4 progressing towards satisfying Ψ_1 , Ψ_3 , and Ψ_4 . Robot 3 has to change its path to avoid colliding with robot 1. In figure 3c the robots have satisfied Ψ_1, \dots, Ψ_4 . At $t \approx 12$ the robots sense *approach* and robots 1 and 3 begin to satisfy $\Psi_{approach_1}$ and $\Psi_{approach_3}$. At $t \approx 14$ the robots sense *align* and robots 2 and 4 begin to satisfy Ψ_{align_2} and Ψ_{align_4} (3d and 3e). Figure 3f shows the configuration of the robots at $t = 30$.

robot task described in Sec IV. The robots do not collaborate and each robot only has information about the position of the other robots. The initial state $\mathbf{x}_0 = [x_1, y_1, \theta_1, \dots, x_4, y_4, \theta_4]$ of the system is $\mathbf{x}_0 = [0, 0, 0, 0, 1, 0, 0, 2, 0, 0, 2.75, 0]$ and the velocity bound $\mathbf{U} = \pm[u_{x1}, u_{y1}, u_{\theta1}, \dots, u_{x4}, u_{y4}, u_{\theta4}]$ of the system is $\mathbf{U} = \pm[0.7, 0.7, 0.5, 0.9, 0.9, 0.5, 0.65, 0.65, 0.5, 0.8, 0.8, 0.5]$.

The Büchi automaton took 1:15 minutes to compute on a 2.3 GHz Quad-Core CPU with 8 GB of RAM and contains 281 states, 21,121 transitions, and 14 CBFs.

B. Simulation Results

Figure 3 shows the trajectory of the system at different time steps. All robots are able to satisfy their individual tasks while avoiding collisions as defined in $\Psi_{collision_{ij}}$. The robots were able to proceed to their goal regions represented by the circular regions without collision. This simulation was run at 10Hz and the controllers for all robots in the system took approximately 0.07 seconds to compute. The simulation was run on a 2.3 GHz Quad-Core CPU with 8 GB of RAM. The a priori feedback is that if $approach_1$ or $approach_3$ are sensed before the predicates in Ψ_1 or Ψ_3 are satisfied, the specification may be violated.

VIII. PHYSICAL DEMONSTRATION

A. Example Description

To further show the expressive power of Event-based STL and the feedback we can generate we conduct a physical demonstration with two iRobot Creates. We consider the following Event-based STL specification for the multi-robot system

- $\Psi_1 = F_{[0,15]}(\|\mathbf{x}_1 - [-2, 1]^T\| < 0.5)$
- $\Psi_2 = F_{[1,16]}(\|\mathbf{x}_2 - [2, 1]^T\| < 0.5)$
- $\Psi_3 = G(alarm \Rightarrow F_{[0,10]}(\|\mathbf{x}_1 - [0, -1]^T\| < 0.5))$
- $\Psi_4 = G_{[0,25]}(\|\mathbf{x}_1 - \mathbf{x}_2\| > 0.5)$

The task is defined as the conjunction of all of the Event-based STL formulas $\Psi = \Psi_1 \wedge \Psi_2 \wedge \Psi_3 \wedge \Psi_4$

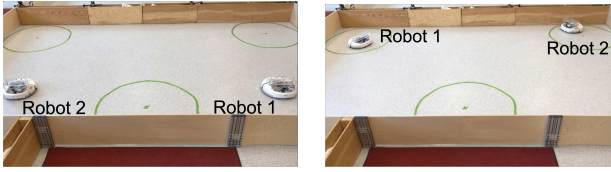
B. A priori feedback

Before executing a run, we provide feedback to the user on the feasibility of a task. To do this we check if conflicting CBFs exist that may be activated at the same time during an execution, as outlined in Sec. VI. For the physical demonstration there are several transitions in the Büchi automaton that activate conflicting CBFs. These conflicting CBFs come from the predicate functions associated with Ψ_1 and Ψ_3 which can not be satisfied at the same time. This only occurs when the robot senses *alarm* and Ψ_1 has not been satisfied. We alert the user of this potential issue so that they can change the specification accordingly.

C. Physical Demonstration Results

The following section describes the results of the physical demonstrations where *alarm* becomes *True* at different times. In the first execution *alarm* never becomes *True* and the system remains in an accepting state. Snapshots of this run are shown in figure 4.

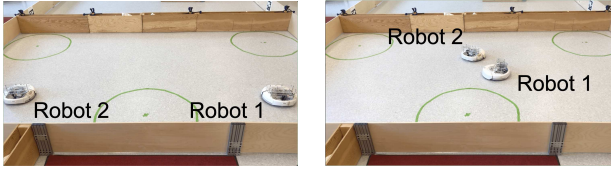
In the second run the robot senses the *alarm* event at $t \approx 17$. This is after robot 1 has satisfied Ψ_1 . Figure 5 shows the position of the robots at various timesteps. In this execution the collision avoidance described by Ψ_4 can be seen as both robots change their paths so that they do not



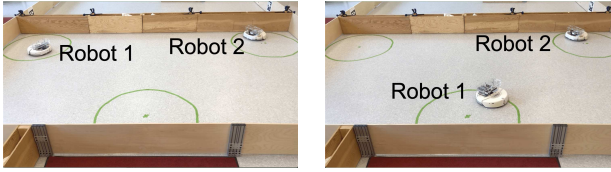
(a) Initial configuration of robot 1 and robot 2 (b) Configuration of the robots at $t = 25$.

Fig. 4: The robots do not sense *alarm* and the robots remain in an accepting states in the safe-sets defined by Ψ_1 and Ψ_2

collide with each other. During execution there is no run-time feedback given because the robots are always able to satisfy their tasks based on their bounded control inputs.



(a) Initial configuration of robot 1 and robot 2 (b) Both robots have to change directions to avoid colliding with each other



(c) The robots sense *alarm* after robot 1 satisfies Ψ_1 at ≈ 17 (d) Robots in safe-set satisfying the full specification.

Fig. 5: Snapshots when the robots sense *alarm*

IX. CONCLUSIONS

We provide a framework for expressing and synthesizing control for high-level specifications that include reactions to uncontrolled events and bounds on time and control input. To do this we create a specification formalism called Event-based STL and show its capabilities through simulation and physical demonstrations. Because there are bounded control inputs and a possibility of unknown disturbances and environment inputs, we cannot provide a-priori guarantees that a specification can be satisfied. Instead we provide feedback to the user as to why the specification can not be satisfied, when we detect a problem. In future work we will consider specifications in complex environments and work to expand the feedback given to users regarding infeasible tasks and provide suggestions of changes to make the specification satisfiable.

REFERENCES

- [1] H. Kress-Gazit, M. Lahijanian, and V. Raman, "Synthesis for Robots: Guarantees and Feedback for Robot Behavior," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 211–236, 2018.
- [2] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multi-agent motion tasks based on LTL specifications," *Proceedings of the IEEE Conference on Decision and Control*, vol. 1, pp. 153–158, 2004.
- [3] I. Filippidis, D. V. Dimarogonas, and K. J. Kyriakopoulos, "Decentralized multi-agent control from local LTL specifications," *Proceedings of the IEEE Conference on Decision and Control*, no. 0, pp. 6235–6240, 2012.
- [4] V. Raman and H. Kress-Gazit, "Synthesis for multi-robot controllers with interleaved motion," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4316–4321, 2014.
- [5] M. Kloetzer and C. Belta, "Temporal logic planning and control of robotic swarms by hierarchical abstractions," *IEEE Transactions on Robotics*, vol. 23, no. 2, pp. 320–330, 2007.
- [6] J. Chen, S. Moarref, and H. Kress-Gazit, "Verifiable control of robotic swarm from high-level specifications robotics track," *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, vol. 1, no. 4, pp. 568–576, 2018.
- [7] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Lecture Notes in Computer Science*, vol. 6246 LNCS, pp. 92–106, 2010.
- [8] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.
- [9] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," *Lecture Notes in Computer Science*, vol. 3253, pp. 152–166, 2004.
- [10] A. M. Jones, K. Leahy, C. Vasile, S. Sadraadini, Z. Serlin, R. Tron, and C. Belta, "ScRATCHS : Scalable and Robust Algorithms for Task-based Coordination from High-level Specifications," *International Symposium of Robotics Research*, pp. 1–16, 2019.
- [11] V. Raman, M. Maasoumy, A. Donze, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," *Proceedings of the IEEE Conference on Decision and Control*, pp. 81–87, 2014.
- [12] Z. Liu, J. Dai, B. Wu, and H. Lin, "Communication-aware motion planning for multi-agent systems from signal temporal logic specifications," *Proceedings of the American Control Conference*, pp. 2516–2521, 2017.
- [13] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control Barrier Function Based Quadratic Programs for Safety Critical Systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3861–3876, 2017.
- [14] L. Wang, A. D. Ames, and M. Egerstedt, "Safety barrier certificates for collisions-free multirobot systems," *IEEE Transactions on Robotics*, vol. 33, no. 3, pp. 661–674, 2017.
- [15] L. Lindemann and D. V. Dimarogonas, "Control barrier functions for signal temporal logic tasks," *IEEE Control Systems Letters*, vol. 3, no. 1, pp. 96–101, 2019.
- [16] L. Lindemann and D. V. Dimarogonas, "Control barrier functions for multi-agent systems under conflicting local signal temporal logic tasks," *IEEE Control Systems Letters*, vol. 3, no. 3, pp. 757–762, 2019.
- [17] L. Lindemann and D. V. Dimarogonas, "Decentralized control barrier functions for coupled multi-agent systems under signal temporal logic tasks," in *2019 18th European Control Conference, ECC 2019*, pp. 89–94, 2019.
- [18] L. Lindemann and D. V. Dimarogonas, "Efficient Automata-based Planning and Control under Spatio-Temporal Logic Specifications," *2020 American Control Conference (ACC)*, pp. 4707–4714, 2020.
- [19] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia, "Reactive synthesis from signal temporal logic specifications," *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC 2015*, pp. 239–248, 2015.
- [20] P. Wieland and F. Allgöwer, "Constructive safety using control barrier functions," *IFAC Proceedings Volumes (IFAC-PapersOnline)*, vol. 7, no. PART 1, pp. 462–467, 2007.
- [21] P. Gastin and D. Oddoux, "Fast LTL to büchi automata translation," in *Lecture Notes in Computer Science*, pp. 53–65, Springer, 2001.
- [22] G. J. Holzmann, "The model checker SPIN," *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, 1997.
- [23] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, É. Renault, and L. Xu, "Spot 2.0 — a framework for LTL and ω -automata manipulation," *Lecture Notes in Computer Science*, vol. 9938 LNCS, pp. 122–129, 2016.