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) [1] are synthesized into controllers 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].

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 computational burden 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.

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

The work in [15] uses time-varying control barrier functions (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.

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 work, 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 environment do not violate the specification, all robots in the system are holonomic, and all robots have full knowledge of the state of the other robots in multi-robot tasks.

Contributions: We propose a framework for encoding tasks that contain timing constraints and reaction to environmental events, creating a control strategy to satisfy the task using control barrier functions, 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 timed tasks where the robots must react to environment events, 2) an automata-based synthesis framework for generating decentralized controllers for multi-robot systems under an Event-based STL specification using time-varying CBFs, 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.

In this paper, following the preliminaries (Sec. II), we formally define Event-based STL (Sec. III) and provide ex-
ample tasks (Sec. [IV]). Sec. [V] describes our control synthesis approach and Sec. [VI] describes how we generate feedback for infeasible Event-based STL specifications. Finally, in Sec. [VII] and Sec. [VIII] we demonstrate the capabilities of Event-based STL through simulation and physical demonstrations.

II. PRELIMINARIES

A. Signal Temporal Logic (STL)

Consider a discrete time dynamical system representing robot motion:

$$x_{t+1} = f(x_t) + g(x_t)u_t$$  \hspace{1cm} (1)

Where $x_t \in \mathbb{R}^n$ is the state of the system at time $t$, $u_t \in U \subseteq \mathbb{R}^m$ is the bounded control input of the system at time $t$, and $f$ and $g$ are locally Lipschitz continuous functions.

Let $\mu \in \{\text{True, False}\}$ represent a predicate whose truth value is defined by the evaluation of a predicate function $h(x_t)$.

$$\mu := \begin{cases} \text{False} & \Rightarrow h(x_t) < 0 \\ \text{True} & \Rightarrow h(x_t) \geq 0 \end{cases}$$  \hspace{1cm} (2)

Syntax: An STL formula $\phi$ is defined recursively as

$$\phi := \text{True} \mid \mu \mid \neg \phi \mid \phi_1 \land \phi_2 \mid F[a,b] \phi \mid G[a,b] \phi \mid \phi_1 U[a,b] \phi_2$$  \hspace{1cm} (3)

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

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

$$x_t \models \mu \iff h(x_t) \geq 0$$

$$x_t \models \neg \phi \iff x_t \not\models \phi$$

$$x_t \models \phi_1 \land \phi_2 \iff x_t \models \phi_1 \text{ and } x_t \models \phi_2$$

$$x_t \models F[a,b] \phi \iff \exists i \in [t + a, t + b] \text{ s.t. } x_{i+1} \models \phi$$

$$x_t \models G[a,b] \phi \iff \forall i \in [t + a, t + b], x_i \models \phi$$

$$x_t \models \phi_1 U[a,b] \phi_2 \iff \exists i \in [t + a, t + b] \text{ s.t. } x_{i+1} \models \phi_2 \text{ and } \forall t_1 \in [t + a, t_2], x_{i+1} \models \phi_1$$

Intuitively, $F[a,b] \phi$ is True if there exists a time between $a$ and $b$ where $\phi$ is True, $G[a,b] \phi$ is True if $\phi$ is True for all time between $a$ and $b$, and $\phi_1 U[a,b] \phi_2$ is True if $\phi_1$ is True for all time until $\phi_2$ becomes True.

B. CBFs for STL specifications

Control barrier functions (CBFs) were proposed by [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 control barrier functions (CBFs). To ensure a task is satisfied given the timing constraints of an STL formula, [15] creates CBFs $cbf(x_t)$ that are time varying and forward invariant. These CBFs are constructed using predicate functions of the STL formula, $h(x_t)$. The function is forward invariant if eqn. [4] holds for all $x_t$.

$$\sup_{u \in U} \frac{\partial cbf(x_t)^T}{\partial x}(f(x_t) + g(x_t)u_t) + \frac{\partial cbf(x_t)}{\partial t} \geq -\nu(cbf(x_t))$$  \hspace{1cm} (4)

where $\nu : \mathbb{R}^2_+ \rightarrow \mathbb{R}^2_+$ is a locally Lipschitz continuous function. The following equations from [15] describe the conditions for the satisfaction of an STL formula given timing constraints.

$$cbf(x_0) \geq 0$$  \hspace{1cm} (5a)

$$cbf(x_t) \geq 0 \forall t$$  \hspace{1cm} (5b)

$$cbf(x_{\gamma}) \leq h(x)$$  \hspace{1cm} (5c)

where $t_f$ is the upper bound on the timing constraints of an 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, $cbf_\phi$, such that if $cbf_\phi \geq 0$, then $cbf_i \geq 0 \forall i [15]$

$$cbf_\phi = -\ln \left(\sum_{i=1}^{I} \exp(-cbf_i(x_t))\right)$$  \hspace{1cm} (6)

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

C. Linear Temporal Logic (LTL) and Büchi Automata

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

$$\gamma := \pi \mid \neg \gamma \mid \gamma_1 \lor \gamma_2 \mid X \gamma \mid \gamma_1 U \gamma_2$$  \hspace{1cm} (7)

where $\pi \in AP$, $\neg$ and $\lor$ are the Boolean operators “not” and “or”, $X$ 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 $\sigma = \sigma_1, \sigma_2, \ldots$, where $\sigma_i \subseteq AP$ represents the propositions that are True in position $i$ of the sequence. The truth value of an LTL formula is defined recursively as:

$$\sigma, i \models \pi \iff \pi \in \sigma_i$$

$$\sigma, i \models \neg \gamma \iff \sigma, i \not\models \gamma$$

$$\sigma, i \models (\gamma_1 \lor \gamma_2) \iff (\sigma, i) \models \gamma_1 \text{ or } (\sigma, i) \models \gamma_2$$

$$\sigma, i \models X \gamma \iff (\sigma, i+1) \models \gamma$$

$$\sigma, i \models (\gamma_1 U \gamma_2) \iff \exists k \geq i \text{ s.t. } (\sigma, k) \models \gamma_2 \text{ and } \forall i \leq j < k, (\sigma, j) \models \gamma_1$$

Intuitively, $X \gamma$ is True if for every execution $\gamma$ is True in the next position of the sequence, $F \gamma$ is True if for every execution $\gamma$ is True at some position in the sequence, $G \gamma$ is True if for every execution $\gamma$ is True at all positions of the sequence, and $\gamma_1 U \gamma_2$ is True if for every execution $\gamma_1$ is True until $\gamma_2$ becomes True.
A deterministic Büchi automaton is a tuple

\[ B = (S, s_0, \Sigma, \delta, F) \]  

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

Given an LTL formula \( \gamma \), we can construct a Büchi automaton \( B_\gamma \) such that \( B_\gamma \) only accepts input words that satisfy \( \gamma \) [21], [22]. In this work 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 \( \Psi \) as follows:

\[
\begin{align*}
\varphi &::= \mu | \neg \mu | \varphi_1 \land \varphi_2 \\
\alpha &::= \pi | \neg \alpha | \alpha_1 \land \alpha_2 \\
\Psi &::= G_{[a,b]} \varphi | F_{[a,b]} \varphi | \varphi_1 U_{[a,b]} \varphi_2 | G(\alpha \Rightarrow \Psi) | G(\varphi \Rightarrow \Psi) | \Psi_1 \land \Psi_2
\end{align*}
\]  

where \( \mu \) is a predicate representing \( h(x_i) \) as described in eqn. [2] \( \alpha \) 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 \([0, \infty]\).

#### C. Semantics of Event-Based STL

We define the semantics of Event-based STL over \((x_t, \sigma_t)\) where \( x_t \) is the state of the system at time \( t \) and \( \sigma_t \) is a set of environment propositions that are \( True \) at time \( t \).
The sub formulas $Ψ_{1,2,3,4}$ describe when the robots should be in a certain region. $Ψ_{collision,i}$ describes six sub formulas for collision avoidance which states that each robot must maintain a distance of at least 0.3 units from each other robot. $Ψ_{approach,i}$ 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 $n$, $σ_i$, $x_i$, and the functions $h_i(x_i)$. The outputs are the control inputs $u_i \in U_i$ for each robot, that satisfy $Ψ_{STL}$.

Algorithm 1: Control synthesis for Event-based STL

```
Input : $Ψ_{STL}$, $n$, $σ_i$, $x_i$, $h_i(x_i)$
Output: $u_i$
1 $∀i, cbf_{μ_i} = CBFTemplate(h_i(x_i))$
2 $(Ψ_{LTL}, Π_{μ_i}) = STL2LTL(Ψ_{STL})$
3 $B_{Ψ_{LTL}} = LTL2Buchi(Ψ_{LTL})$
4 $currS = s_0$
5 $σ_{-1} = σ_0$
6 $(σ_{currS, nextS}, Π_{μ_act}, currS) = findTransition(σ_0, x_0, B_{Ψ_{LTL}}, h_i(x_i), currS)$
7 while True do
    // check whether reached nextS or environment event changed
    8 if $(σ_{currS, nextS} is True) or σ_i \neq σ_{-1}$ then
        9 $(σ_{currS, nextS}, Π_{μ_act}, currS) = findTransition(σ_i, x_i, B_{Ψ_{LTL}}, h_i(x_i), currS)$
    end
    // Execute Barrier Functions
    10 for $i = 1$ to $n$ do
        11 $u_i = Barrier(Π_{μ_act}, t, x_i)$
    end
    12 if Eqn. [7] is infeasible then
        13 Stop;
    end
    14 $σ_{i-1} = σ_i$
15 end
```

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 $B_{Ψ_{LTL}}$ 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. In this work we use the methods from [15] to create CBF formula templates that use parameters from an Event-based STL formula.

Given eqn. [5] and [15], we create a control barrier function template, eqn. [12] that changes linearly with time and utilizes the entirety of the time bound that is given. For this template we use the predicate function $h_i(x_i)$, 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).

$$cbf_{μ_i}(x_i) = \frac{(t - t_{int} - a)h_i(x_i)}{b - a} - h_i(x_{i_{int}}) + h_i(x_i)$$ (12)

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, $cbf_{μ_i}(x_i) = 0$. At $t = t_{int} + b$, the final time in the interval for the Event-based STL formula, $cbf_{μ_i}(x_i) = h_i(x_i)$.

For example, the predicate from the single robot example is $μ_1 = || x - [5,5]T || < 1$. We form a predicate function $h_1(x_i) = 1 - || x - [5,5]T ||$, and construct a CBF $cbf_{μ_1}(x_i) = \frac{(t - t_{int} - a)(1||x_{i_{int}} - [5,5]T ||) + || x_{i_{int}} - [5,5]T ||}{b - a}$.

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 $π_{μ_i,[a,b]} \in Π_{μ_i}$ that we consider a controllable proposition. Each controllable proposition maintains the timing associated with its Event-based STL subformula. For example, eqn. [13] is abstracted to eqn. [14].

$$Ψ_{STL} = G(\text{alarm} \Rightarrow F_{[0,10]}(|| x - [5,5]T || < 1))$$ (13)

$$Ψ_{LTL} = G(\text{alarm} \Rightarrow F(π_{μ_i,[0,10]}))$$ (14)

where $π_{μ_i,[0,10]}$ replaces $(|| x - [5,5]T || < 1)$ and the associated timing constrains.

Generating $B_{Ψ_{LTL}}$ (Line 3 of Algo. 1): We create a Büchi automaton $B_{Ψ_{LTL}}$ from $Ψ_{LTL}$ using [23]. The transitions are labeled with Boolean formulas over the set $AP \cup Π_{μ_i}$ as seen in figure 7. We denote a Boolean formula over $AP \cup Π_{μ_i}$
representing the label of the transition between \( s_i \) and \( s_j \) as \( \sigma_{s_i,s_j} \), i.e. \( \Sigma = \{ \sigma_{s_i,s_j} | \exists s_i, s_j \in S, \delta(s_i, \sigma_{s_i,s_j}) = s_j \} \)

\[ \sigma_{s_i,s_j} \rightarrow ]

Fig. 1: Graphical representation of \( B_{\Psi,LTL} \) for \( \Psi_{LTL} \) in eqn. [4] The grey circles represent states and the double circle represents an accepting state. Transitions between states are labeled with the Boolean formulas \( \sigma_{s_i,s_j} \).

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 \( \{ \text{alarm}, \pi_{[0,10]} \} \). For example, \( \sigma_{s_0,s_1} = \neg \text{alarm} \land \neg \pi_{[0,10]} \). The task is satisfied when the system is in the accepting state, i.e. when the predicate \( \mu_1 \) 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 Alg. 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 \( \Pi_\mu \).

At each time step, given \( \sigma_t \), the set of environment propositions that are True, and the state of the system \( x_t \), we first determine the truth values of all the propositions \( AP \cup \Pi_\mu \); for \( \pi \in AP \):

\[
\pi = \begin{cases} 
\text{False} & \text{if } \pi \notin \sigma_t \\
\text{True} & \text{if } \pi \in \sigma_t 
\end{cases}
\]

(15)

and for \( \pi_{[a,b]} \in \Pi_\mu \):

\[
\pi_{[a,b]} = \begin{cases} 
\text{False} & \text{if } h_i(x_t) < 0 \\
\text{True} & \text{if } h_i(x_t) \geq 0 
\end{cases}
\]

(16)

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 \( \mu_i \) 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 \( \sigma \) to evaluate to False.

Given the set of possible transitions, we find the shortest path to an accepting state. Given this shortest path, we choose, as the active transition, the next transition in this path. We denote the set of \( \pi_{[a,b]} \) propositions that must be True to satisfy the Boolean formula \( \sigma_{currS,s_j} \) for this transition as \( \Pi_{\mu act} \); this set represents the CBFs that are activated for that transition to complete. If there is more than one transition \( \sigma_{currS,s_j} \) we can choose from on the shortest path, we choose the transition which has the lowest number of \( \pi_{[a,b]} \) that are True, thereby reducing the number of CBFs we need to consider.

Using the example, if the system is in state \( s_2 \) the shortest and only path to an accepting state is from state \( s_2 \) to state \( s_1 \). For this transition to occur, the controllable proposition \( \pi_{[0,10]} = \text{True} \) therefore \( \pi_{[0,10]} \in \Pi_{\mu act} \). The activated CBF will progress the system towards the accepting state \( s_1 \). The system will not reach \( s_1 \) until \( h_1(x_t) \), the predicate function associated with \( \pi_{[10]} \), becomes \( \geq 0 \).

### C. Control synthesis

**Finding Control input \( u \):**

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. [17] where \( cfb_{\Psi_i} \) is the combination of all activated CBFs for robot \( i \) in the system found from eqn. [6]. Eqn. [17] describes the optimization problem where a control law \( u_i \) is found that ensures that \( cfb_{\Psi_i}(x_t) \geq 0 \forall t \).

\[
\min_{u_i \in U_i} \| u_i - \hat{u}_i \| \text{ s.t. } \frac{\partial cfb_{\Psi_i}(x_t)}{\partial t} f(x,u) + \frac{\partial cfb_{\Psi_i}(x_t)}{\partial t} \geq -\nu(cfb_{\Psi_i}(x_t))
\]

where the nominal controller for each robot \( \hat{u}_i \) is the maximum control input in the direction of the safe-set. 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 \( t_0 \) where the proposition alarm becomes True, and at \( t_{final} = t_0 + 10 \).

### 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 first provide feedback regarding possibly conflicting CBFs. To do this we examine the set $\Pi_{\text{act}}$ of each transition in $B\Psi_{\text{LTL}}$. If, for a given transition, the sets $h_i(x_i)$ associated with $\Pi_{\text{act}}$ 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.

**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 $h(x_i)$. We then compare this distance to the largest distance the system can move in state space given the bounds on the control and the time remaining to satisfy the predicate

$$\| u_{\text{max}} \| (b + t_{\text{int}} - t)$$

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.

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. [17] 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.

**VII. SIMULATION RESULTS**

A. **Example Description**

We consider 4 holonomic robots that operate in a shared environment. They are performing the multi-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 $x_0 = [x_1, y_1, \theta_1, ..., x_4, y_4, \theta_4]$ of the system is $x_0 = [0, 0, 0, 0, 1, 0, 0, 2, 0, 0, 2, 75, 0]$ and the velocity bound $U = \pm [u_{\text{R1}}, u_{\text{R2}}, u_{\text{R3}}, ..., u_{\text{R4}}]$ of the system is $U = \pm [0.7, 0.7, 0.5, 0.9, 0.5, 0, 0.5, 0.8, 0, 0.5]$. 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 [2] shows the trajectory of the system at different times. All robots are able to satisfy their individual tasks while avoiding collisions as defined in $\Psi_{\text{collision},i}$. 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.

**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

- $\Psi_1 = F_{[0,15]}(|x_1 - [-2,1]^T| < 0.5)$
- $\Psi_2 = F_{[1,16]}(|x_2 - [2,1]^T| < 0.5)$
- $\Psi_3 = G(\text{alarm} \Rightarrow F_{[0,10]}(|x_1 - [0,-1]^T| < 0.5))$
- $\Psi_4 = G_{[0,25]}(|x_1 - x_2| > 0.5)$

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

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 $\Psi_1$ and $\Psi_3$ which can not be satisfied at the same time. This only occurs when the robot senses $\text{alarm}$ and $\Psi_1$ 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 $\text{alarm}$ becomes True at different times. In the first execution $\text{alarm}$ 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 $\text{alarm}$ event at $t \approx 17$. This is after robot 1 has satisfied $\Psi_1$. Figure [5] shows the position of the robots at various times.
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 $\Psi_1$, $\Psi_3$, and $\Psi_4$. Robot 3 has to change its path to avoid colliding with robot 1. In figure 3c, the robots have satisfied $\Psi_1$, $\Psi_3$, and $\Psi_4$. 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 $\Psi_{align_2}$ and $\Psi_{align_4}$. Figure 3f shows the configuration of the robots at $t = 30$.

Fig. 4: The robots do not sense *alarm* and the robots remain in an accepting state in the safe-sets defined by $\Psi_1$ and $\Psi_2$.

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 execution the collision avoidance described by $\Psi_4$ can be seen as both robots change their paths so that they do not collide with each other.

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. Kloezer 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 (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 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 (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3253, pp. 152–166, 2004.

[10] A. M. Jones, K. Leaby, C. Vasile, S. Sadraddini, Z. Sertin, R. Tron, and C. Belta, “ScRATCHS: Scalable and Robust Algorithms for Task-based Coordination from High-level Specifications,” 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*, vol. 2015-February, no. February, 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,” 2019.

[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 (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 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 (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9938 LNCS, pp. 122–129, 2016.