Successive Convexification for Optimal Control with Signal Temporal Logic Specifications

Yuanqi Mao, Behcet Acikmese, Pierre-Loïc Garoche, Alexandre Chapoutot

Successive Convexification for Optimal Control with Signal Temporal Logic Specifications

Yuanqi Mao
University of Washington
Seattle, Washington, USA
yqmao@uw.edu

Behçet Açıkmeşe
University of Washington
Seattle, Washington, USA
behcet@uw.edu

Pierre-Loic Garoche
Ecole Nationale de l'Aviation Civile
Toulouse, France
pierre-loic.garoche@enac.fr

Alexandre Chapoutot
ENSTA Paris, Institut Polytechnique de Paris
Palaiseau, France
alexandre.chapoutot@ensta-paris.fr

ABSTRACT

As the scope and complexity of modern cyber-physical systems increase, newer and more challenging mission requirements will be imposed on the optimal control of the underlying unmanned systems. This paper proposes a solution to handle complex temporal requirements formalized in Signal Temporal Logic (STL) specifications within the Successive Convexification (SCvx) algorithmic framework. This SCvx-STL solution method consists of four steps: 1) Express the STL specifications using their robust semantics as state constraints. 2) Introduce new auxiliary state variables to transform these state constraints as system dynamics, by exploiting the recursively defined structure of robust STL semantics. 3) Smooth the resulting system dynamics with polynomial smooth min- and max-functions. 4) Convexify and solve the resulting optimal control problem with the SCvx algorithm, which enjoys guaranteed convergence and polynomial time subproblem solving capability. Our approach retains the expressiveness of encoding mission requirements with STL semantics, while avoiding the usage of combinatorial optimization techniques such as Mixed-integer programming. Numerical results are shown to demonstrate its effectiveness.

CCS CONCEPTS

• **Mathematics of computing** → **Nonconvex optimization**; • **Theory of computation** → **Modal and temporal logics**; • **Computer systems organization** → *Robotic autonomy*.

KEYWORDS

optimal control, successive convexification, signal temporal logic, robust semantics

ACM Reference Format:
Yuanqi Mao, Behçet Açıkmeşe, Pierre-Loic Garoche, and Alexandre Chapoutot. 2022. Successive Convexification for Optimal Control with Signal Temporal Logic Specifications. In *25th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '22), May 4–6, 2022, Milan, Italy.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3501710.3519518

HSCC '22, May 4–6, 2022, Milan, Italy

ACM ISBN 978-1-4503-9196-2/22/05…$15.00
https://doi.org/10.1145/3501710.3519518

1 INTRODUCTION

In the envisioned near future of cyber-physical systems, unmanned systems such as Unmanned Aerial Vehicles (UAVs) and intelligent ground robots are playing an increasingly important role, as they are assigned newer and more challenging tasks. From package delivery to traffic monitoring, from aerial photograph to fieldwork on modern farms, careful and autonomous trajectory planning is of utmost importance to ensure safety and mission requirements satisfaction. An onboard optimal-control-based guidance system is proved to be highly effective in this regard [12]. A notable line of work on this subject is the Successive Convexification(SCvx) algorithmic framework [14], which provides both theoretical convergence guarantee and practical real-time solving capability. However, traditional optimal control solvers, including SCvx, are designed to handle spatial (state) and physical (control) constraints, not temporal ones. Meanwhile, we have seen a growing trend of stating complex mission requirements as temporal specifications [1], thanks to the expressiveness of temporal logic encoding. Examples of these requirements include deadlines, periodic occurrences, and sequentially triggered events, etc.. The downside is that its semantics are combinatorial in nature and thus difficult to incorporate in a smooth optimization solver, which is usually required for onboard computation. This dilemma between constraint expressiveness and real-time solvability begs the question: is it possible to have a smooth optimal control solver that also incorporates temporal logic specifications as constraints?

In this work, we aim to provide a positive answer to this question. For the smooth optimization solver part, we are going to adopt the SCvx framework. Hence, the problem becomes how to express the temporal logic encoding as constraints recognizable by SCvx. Since SCvx deals with continuous-time systems we are going to use Signal Temporal Logic (STL) as the primary form of the temporal encoding, since it also operates on continuous-time trajectories ("signals"). A more detailed survey on the state of the art of both SCvx and STL is given in section 1.1. In this paper, we will establish a four-steps approach, called SCvx-STL, to tackle the problem. Novel contributions of this work are described in section 1.2.

1.1 State of the art

Optimal control (trajectory optimization) of unmanned vehicles has been extensively studied. The reader is referred to [12] for an overview of the field. In particular, convex optimization and convexification based technologies have been developed for more than a decade now [2, 3], while newer 6-DoF (Degrees of Freedom) formulations with attitude control have been studied in recent years [10, 20, 21]. Lately we have also seen commercial usages of these technologies on, for instance, SpaceX's Falcon series reusable rockets [5]. On a much smaller scale, consider quadrotors, systems that are becoming increasingly ubiquitous in the world of cyber-physical systems. Convexification-based real-time guidance are also used in these agile systems [13, 22], which includes non-convex obstacle avoidance constraints. A commonly used algorithmic paradigm in this line of work is the Successive Convexification (SCvx) framework [14, 15]. It is an iterative procedure that solves optimal control problems with nonlinear dynamics and non-convex state and control constraints, and provides proofs of global convergence and superlinear convergence rate. We will use SCvx as our smooth optimization solver thanks to its good synergy with STL specifications. More on this later.

The use of Signal Temporal Logic (STL) is often associated with formal specification and monitoring of cyber-physical systems. STL was originally developed in order to specify and monitor the expected behavior of physical systems [11], including temporal constraints between events. STL allows the specification of properties of finite-horizon, dense-time, real-valued signals, and the automatic generation of monitors for testing these properties on individual simulation traces. It has since been applied to the analysis of several types of continuous and hybrid systems, including dynamical systems where the continuous variables represent quantities like position and velocity of a vehicle [18]. STL has the advantage of naturally admitting quantitative semantics which, in addition to the yes/no answer to the satisfaction question, provide a real number that grades the quality of the satisfaction or violation. Such semantics have been defined to assess the robustness of systems to parameter or timing variations [6]. It has also been used recently [4] in the context of controller synthesis with reinforcement learning to provide a reward associated to the formalized STL properties. STL as part of the Model Predictive Control (MPC) design has also been studied in [18], in which the combinatorial nature of such specifications is preserved, and thus Mixed-Integer Linear Programming (MILP) has to be employed. More recently, for a specific multi-quadrotor setting, [16] relaxes the robustness measures using smooth approximations (log-sum-exponential functions) so that the resulting problem can be solved by a smooth optimization solver. It, however, has to employ a two-tier optimization framework because the amount of computation is not real-time implementable.

1.2 Contributions

This paper presents a four-step solution to optimal control problems with STL specifications within the SCvx framework. The novel contributions of this work are threefold:

- **The SCvx-STL solution method**: To the best of our knowledge, this is the first work to incorporate STL specified mission requirements in the SCvx framework to solve optimal control problems. By combining the two, we ensure continuous-time satisfaction of temporal logic specifications, and also enjoy the benefit of guaranteed convergence and real-time computation provided by SCvx.
- **The "STL subdynamics"**: By introducing auxiliary state variables, we take advantage of the special recursive structure of robust STL semantics and transform the STL specified state constraints into system dynamics. They can then be convexified in the same way we treat system dynamics in SCvx. This approach effectively creates STL subdynamics, and dramatically reduces the complexity of gradient computation for STL constraints.
- **The polynomial smooth min and max**: We borrow the concept of polynomial smooth min-/max- functions from computer graphics to smooth our robust STL semantics. Practically, it gives better approximation than the log-sum-exponential smooth min-/max- functions used in [16], especially in the case of two variables, thanks to the aforementioned "STL subdynamics" step.

The paper is organized as follows. Section 2 presents the background and mathematical preliminaries of both the SCvx algorithmic framework and the STL semantics. Section 3 details the proposed SCvx-STL solution method and the mathematical reasoning behind it. Section 4 gives numerical results from a quadrotor trajectory planning problem with STL specifications.

2 PRELIMINARIES

2.1 Successive Convexification

Successive Convexification (SCvx) [14, 15] is a family of iterative algorithms designed to solve non-convex constrained optimal control problems with nonlinear dynamics, usually in continuous-time. In practice, a continuous-time optimal control problem has to be discretized, see Problem 1, before it can be solved using optimization methods on a digital computer [9].

Problem 1. **Discretized Optimal Control Problem**

$$\min_{u} \quad C(x,u) \coloneqq \sum_{i=1}^{N} \phi(x_k, u_k), \tag{1a}$$

subject to:

$$x_{k+1} = f(x_k, u_k), \qquad k = 1, 2, \ldots, N-1, \tag{1b}$$
$$s(x_k, u_k) \leq 0, \qquad k = 1, 2, \ldots, N, \tag{1c}$$
$$x_k \in X_k \subseteq \mathbb{R}^{n_x}, \qquad k = 1, 2, \ldots, N, \tag{1d}$$
$$u_k \in U_k \subseteq \mathbb{R}^{n_u}, \qquad k = 1, 2, \ldots, N-1. \tag{1e}$$

Here without loss of generality we can assume $C(x,u)$ is a convex function, because any non-convex part of the cost function can be rewritten as constraints, i.e. absorbed into $s(x,u)$. x_k and u_k represent the state and control vectors at time instance k. X_k and U_k are assumed to be convex and compact sets which include the boundary conditions. Equation (1a) and (1b) represent the discrete-time objective function and system dynamics respectively. Equation (1c) includes the non-convex part of the state and control (input) constraints, while (1d) and (1e) are the convex ones. State and input

constraints can model keep-out zones, physical limitations or any other properties that can be expressed as suitable constraints.

SCvx solves the original non-convex problem, Problem 1, by successively convexifying non-convex system dynamics and constraints with respect to the solution of a previous iteration, starting from a possibly infeasible reference trajectory $(x^{(0)}, u^{(0)})$. The resulting convex subproblems as in Problem 2 are numerically tractable, and can be computed quickly and reliably using convex optimization solvers, making the SCvx algorithm well suited for real-time applications.

Problem 2. **Convex Optimal Control Subproblem**

$$\min_{d,w} \quad C(d,w) + \lambda\, P^{(i)}(d,w),$$

$$\text{s.t.} \quad u^{(i)} + w \in U, \quad x^{(i)} + d \in X, \quad \|w\| \le r^{(i)},$$

where $d \coloneqq x - x^{(i)}$, $d_k \coloneqq x_k - x_k^{(i)}$, $w \coloneqq u - u^{(i)}$, and $w_k \coloneqq u_k - u_k^{(i)}$ in terms of the solution to the current iteration, (x, u). At the k^{th} time step, the first-order approximation of (1b) and (1c) about $(x_k^{(i)}, u_k^{(i)})$ is given by

$$x_{k+1}^{(i)} + d_{k+1} = f(x_k^{(i)}, u_k^{(i)}) + A_k^{(i)} d_k + B_k^{(i)} w_k, \tag{2a}$$

$$s(x_k^{(i)}, u_k^{(i)}) + S_k^{(i)} d_k + Q_k^{(i)} w_k \le 0, \tag{2b}$$

where $A_k^{(i)}, B_k^{(i)}, S_k^{(i)}$ and $Q_k^{(i)}$ are the respective partial derivatives. $P^{(i)}(\cdot,\cdot)$ is the penalty function associated with (2a) and (2b), while $\lambda > 0$ is the penalty weight. $r^{(i)}$ is the trust region radius. For more information on the significance of these functions to the convergence and how the trust region updating mechanism works, the reader is referred to [14], in which the advantages of SCvx over other generic nonlinear programming methods are also discussed. To summarize, we have the SCvx algorithm shown as Algorithm 1.

Algorithm 1 The SCvx Algorithm

1: **procedure** SCvx($(x^{(1)}, u^{(1)}), \lambda, \epsilon_{tol}$)
2: **input** Randomly generate an initial trajectory $(x^{(1)}, u^{(1)})$. Initialize trust region radius $r^{(1)} > 0$ and penalty weight $\lambda > 0$.
3: **while** not converged, i.e. cost reduction $> \epsilon_{tol}$ **do**
4: **step 1** At $(i+1)^{th}$ succession, solve the convex subproblem, Problem 2, at $(x^{(i)}, u^{(i)}, r^{(i)})$ to get an optimal solution (d, w).
5: **step 2** Update the trust region radius, see [14]. Based on the trust region rules, either recompute with the updated trust region, or accept the solution (d, w) and assign $x^{(i+1)} = x^{(i)} + d, u^{(i+1)} = u^{(i)} + w$ and loop back to **step 1**.
6: **end while**
7: **return** $(x^{(i+1)}, u^{(i+1)})$.
8: **end procedure**

2.2 Signal Temporal Logics

Formal characterization of mission requirements can be expressed as logical statements over states. While first order logic can be typically used to describe state constraints, properties over trajectories can be expressed using temporal logics such as LTL (Linear temporal logics). These notions have been extended to bounded-horizon continuous-time signals in the form of *Signal Temporal Logics* (STL) and are convenient to express expected behavior of dynamic systems. STL formulas are described using the grammar:

$$\varphi ::= \pi^{\mu} \mid \neg\psi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond_{[a,b]}\, \varphi \mid \varphi_1\, \mathcal{U}_{[a,b]}\, \varphi_2,$$

where π^{μ} is an Boolean predicate whose truth value is determined by the sign of a real-valued function μ, and $\varphi, \varphi_1, \varphi_2, \psi$ are STL formulas. As in LTL, $\Diamond_{[a,b]}$ and $\mathcal{U}_{[a,b]}$ denote the eventually and until modalities, respectively, with a restriction to a given time frame $[a, b]$ with $a < b$. A simulation run $\xi(x_0, u)$ satisfies an STL formula φ is denoted by $\xi \models \varphi$.

Informally, we have *Eventually* denoted as $\xi \models \Diamond_{[a,b]}\, \varphi$, meaning φ holds at some time between a and b. Also, we have *Until* denoted as $\xi \models \varphi\, \mathcal{U}_{[a,b]}\, \psi$, meaning φ holds at every time before ψ holds, and ψ holds at some time between a and b. Additionally, we define *Always* as $\Box_{[a,b]}\, \varphi = \neg \Diamond_{[a,b]}\, (\neg\varphi)$, so that $\xi \models \Box_{[a,b]}\, \varphi$ means φ holds at all times between a and b.

The formal semantics of a formula φ with respect to a run ξ is defined inductively as in Definition 2.1.

*Definition 2.1 (**STL semantics**).*

$$\begin{aligned}
\xi \models \varphi \quad &\Leftrightarrow \quad (\xi, t_0) \models \varphi \\
(\xi, t_k) \models \pi^{\mu} \quad &\Leftrightarrow \quad \mu(x_k, u_k) > 0 \\
(\xi, t_k) \models \neg\psi \quad &\Leftrightarrow \quad \neg(\xi, t_k) \models \psi \\
(\xi, t_k) \models \varphi \wedge \psi \quad &\Leftrightarrow \quad (\xi, t_k) \models \varphi \ \wedge\ (\xi, t_k) \models \psi \\
(\xi, t_k) \models \Diamond_{[a,b]}\, \varphi \quad &\Leftrightarrow \quad \exists\, t_{k'} \in [t_k + a, t_k + b], (\xi, t_{k'}) \models \varphi \\
(\xi, t_k) \models \varphi\, \mathcal{U}_{[a,b]}\, \psi \quad &\Leftrightarrow \quad \exists\, t_{k'} \in [t_k + a, t_k + b], (\xi, t_{k'}) \models \psi \\
&\qquad \wedge\ \forall\, t_{k''} \in [t_k, t_{k'}], (\xi, t_{k''}) \models \varphi.
\end{aligned}$$

A key challenge here is to efficiently relate logical specifications with optimal control. Robust semantics of STL [6] amounts to characterize a real-valued function ρ^{φ} of signal ξ and time t such that $\rho^{\varphi}(\xi, t) > 0 \Rightarrow (\xi, t) \models \varphi$. That is, ρ^{φ} serves as a robustness measure and its absolute value can be viewed as the signed distance of ξ from the set of trajectories in the space of projections with respect to the function μ that define the predicates of φ. The robustness function can also be defined recursively as in Definition 2.2.

*Definition 2.2 (**Robust STL semantics**).*

$$\begin{aligned}
\rho^{\pi^{\mu}}(\xi, t_k) &= \mu(x_k, u_k) \\
\rho^{\neg\psi}(\xi, t_k) &= -\rho^{\psi}(\xi, t_k) \\
\rho^{\varphi_1 \wedge \varphi_2}(\xi, t_k) &= \min\left(\rho^{\varphi_1}(\xi, t_k), \rho^{\varphi_2}(\xi, t_k)\right) \\
\rho^{\varphi_1 \vee \varphi_2}(\xi, t_k) &= \max\left(\rho^{\varphi_1}(\xi, t_k), \rho^{\varphi_2}(\xi, t_k)\right) \\
\rho^{\Diamond_{[a,b]}\, \psi}(\xi, t_k) &= \max_{t_{k'} \in [t_k + a, t_k + b]} \rho^{\psi}(\xi, t_{k'}) \\
\rho^{\varphi_1\, \mathcal{U}_{[a,b]}\, \varphi_2}(\xi, t_k) &= \max_{t_{k'} \in [t_k + a, t_k + b]} \min\Big(\rho^{\varphi_2}(\xi, t_{k'}), \\
&\qquad \min_{t_{k''} \in [t_k, t_{k'}]} \rho^{\varphi_1}(\xi, t_{k''})\Big).
\end{aligned}$$

Temporal operators (e.g. $\max_{t_{k'} \in [t_k + a, t_k + b]}$) are treated as conjunctions or disjunctions along the time axis once discretized for computation on digital computers.

An example optimal control problem with robust STL specifications can be written as follows, in Problem 3.

Problem 3. **Optimal Control with Robust STL Specs**

$$\begin{aligned} \min_{u} \quad & C(x,u) \\ \text{s.t.} \quad & x_{k+1} = f(x_k,u_k),\ k=1,2,\dots,N-1, \\ & x_k \in X_k, u_k \in U_k,\ k=1,2,\dots,N, \\ & \rho^{\varphi}(x) \geq 0. \end{aligned} \tag{3}$$

Unfortunately as shown in Definition 2.2, ρ^{φ} is mainly built using min- and max- operators leading to difficult combinatorial problems, which necessitates the use of Mixed-Integer Programming solvers [18]. A smooth (infinitely differentiable) approximation of these function has been proposed [16] and the solution is to straightforwardly substitute min- and max- functions in ρ^{φ} with log-sum-exponential functions:

$$\begin{aligned} \tilde{\max}_k(a_1,\dots,a_n) &:= \frac{1}{k}\ln(e^{ka_1}+\dots+e^{ka_m}), \\ \tilde{\min}_k(a_1,\dots,a_n) &:= -\tilde{\max}_k(-a_1,\dots,-a_n). \end{aligned} \tag{4}$$

Despite the improved run-time from the smooth operators, experiments reported [16] were not successful to solve (3) with this smooth robust STL semantics, efficiently, in real-time, for a multi-drone fleet trajectory planning problem using the full quadrotor dynamics. It uses a two-tier sampling method instead.

One obvious drawback of this straightforward smoothing method is that the gradient calculation will quickly getting out of hand when the elemental STL specification, such as $\rho^{\varphi_1}(\xi, t_{k''})$ in the formula for the *Until* operator $\rho^{\varphi_1\,\mathcal{U}_{[a,b]}\,\varphi_2}(\xi,t_k)$, is moderately complex. Therefore, it is of great interest to explore other ways to systematically encode and smooth the robust STL semantics.

3 THE SCVX-STL METHOD

In this section, we will present the SCvx-STL method that systematically encodes and smooths the robust STL specifications $\rho^{\varphi}(x)$ in Problem 3. The first step is to follow the discretization procedure described in section 2.1 and the robustification procedure described in section 2.2 and obtain Problem 3.

3.1 STL subdynamics

Once we have established Problem 3 as our main problem to solve, the next step is to take a closer look at the temporal specifications we are trying to encode, namely the *Eventually* $\rho^{\Diamond_{[a,b]}\,\psi}(\xi,t_k)$ and the *Until* $\rho^{\varphi_1\,\mathcal{U}_{[a,b]}\,\varphi_2}(\xi,t_k)$.

3.1.1 The Eventually *operator* $\Diamond_{[a,b]}$. Notice that with the discretized state and control variables, we can denote the first and last time instance for $t_{k'} \in [t_k+a, t_k+b]$ as $k^{(a)}$ and $k^{(b)}$ respectively. Additionally, we define the set of instances in between $k^{(a)}$ and $k^{(b)}$ as $\mathcal{K} := \{k^{(a)}, k^{(a)}+1, \dots, k^{(b)}\}$. Then, we have

$$\rho^{\Diamond_{[a,b]}\,\psi}(x,k) \;=\; \max_{k'\in\mathcal{K}} \rho^{\psi}(x,k')$$

We can then introduce an auxiliary variable $y = [y_1, y_2, \dots, y_k, \dots, y_N]$ as follows.

$$\begin{cases} y_k = \rho^{\psi}(x,k^{(a)}), & \forall\, 1 \leq k \leq k^{(a)}, \\ y_{k+1} = \max\big(y_k,\ \rho^{\psi}(x,k+1)\big), & \forall\, k^{(a)} < k < k^{(b)}, \\ y_k = y_{k^{(b)}}, & \forall\, k^{(b)} < k \leq N. \end{cases}$$

Given $x_{k+1} = f(x_k,u_k)$ from the system dynamics, we have

$$\rho^{\psi}(x,k+1) = \rho^{\psi}(f(x_k,u_k)) := \tilde{\rho}^{\psi}(x_k,u_k).$$

Hence, for $k^{(a)} < k < k^{(b)}$, we have

$$y_{k+1} = \max\big(y_k,\ \tilde{\rho}^{\psi}(x_k,u_k)\big) := f_y(y_k,x_k,u_k), \tag{5}$$

which is recursively defined, and thus y can be treated as a state variable, whose corresponding dynamical equation is exactly (5) for $k^{(a)} < k < k^{(b)}$.

3.1.2 The Until *operator* $\mathcal{U}_{[a,b]}$. Similar considerations can be taken for the *Until* operator, albeit more complex. In additon to the same notation for $k^{(a)}, k^{(b)}$ and $\mathcal{K}$, we also define of all time instances in between t_k and $t_{k'}$ as $\mathcal{K}'$, and all time instances in between t_k and $t_{k^{(a)}}$ as $\mathcal{K}^{(a)}$. Then we can rewrite the formula for the *Until* operator as

$$\rho^{\varphi_1\,\mathcal{U}_{[a,b]}\,\varphi_2}(x,k) \;=\; \max_{k'\in\mathcal{K}} \min\big(\rho^{\varphi_2}(x,k'),\ \min_{k''\in\mathcal{K}'} \rho^{\varphi_1}(x,k'')\big).$$

First, we introduce an auxiliary variable $\zeta = [\zeta_1, \zeta_2, \dots, \zeta_k, \dots, \zeta_N]$ as follows.

$$\begin{cases} \zeta_k = \min_{k''\in\mathcal{K}^{(a)}} \rho^{\varphi_1}(x,k''), & \forall\, 1 \leq k \leq k^{(a)}, \\ \zeta_{k+1} = \min\big(\zeta_k,\ \rho^{\varphi_1}(x,k+1)\big), & \forall\, k^{(a)} < k < k^{(b)}, \\ \zeta_k = \zeta_{k^{(b)}}, & \forall\, k^{(b)} < k \leq N. \end{cases}$$

Similarly, given $x_{k+1} = f(x_k,u_k)$ from the system dynamics, we have

$$\rho^{\varphi_1}(x,k+1) = \rho^{\varphi_1}(f(x_k,u_k)) := \tilde{\rho}^{\varphi_1}(x_k,u_k).$$

Hence, for $k^{(a)} < k < k^{(b)}$, we have

$$\zeta_{k+1} = \min\big(\zeta_k,\ \tilde{\rho}^{\varphi_1}(x_k,u_k)\big) := f_{\zeta}(\zeta_k,x_k,u_k), \tag{6}$$

which is recursively defined, and thus ζ can be treated as a state variable, whose corresponding dynamical equation is exactly (6) for $k^{(a)} < k < k^{(b)}$.

Next, for the outer max function, we introduce another auxiliary variable $\eta = [\eta_1, \eta_2, \dots, \eta_k, \dots, \eta_N]$ as follows.

$$\begin{cases} \eta_k = \min\big(\rho^{\varphi_2}(x,k^{(a)}),\ \zeta_k\big), & \forall\, 1 \leq k \leq k^{(a)}, \\ \eta_{k+1} = \max\Big(\min\big(\rho^{\varphi_2}(x,k+1),\ \zeta_{k+1}\big),\ \eta_k\Big), & \forall\, k^{(a)} < k < k^{(b)}, \\ \eta_k = \eta_{k^{(b)}}, & \forall\, k^{(b)} < k \leq N. \end{cases}$$

Given $x_{k+1} = f(x_k,u_k)$ from the system dynamics, we have

$$\rho^{\varphi_2}(x,k+1) = \rho^{\varphi_2}(f(x_k,u_k)) := \tilde{\rho}^{\varphi_2}(x_k,u_k).$$

Combined with the dynamics for ζ (6), we have for $k^{(a)} < k < k^{(b)}$,

$$\begin{aligned} \eta_{k+1} &= \max\Big(\min\big(\tilde{\rho}^{\varphi_2}(x_k,u_k),\ f_{\zeta}(\zeta_k,x_k,u_k)\big),\ \eta_k\Big) \\ &:= f_{\eta}(\eta_k,\zeta_k,x_k,u_k), \end{aligned} \tag{7}$$

Note that the dynamical time window from $k^{(a)}$ to $k^{(b)}$ for each STL specification can be different, and they are all relative to their respective starting time t_k. Therefore, the above formulation also applies to nested STL specifications. For example, $\rho^{\psi}(\xi,t_{k'})$ in the *Eventually* formula can be an *Until* specification $\rho^{\varphi_1\,\mathcal{U}_{[a',b']}\,\varphi_2}(\xi,t_{k'})$ starting from $t_{k'}$. One just need to adjust the dynamical time window for the associated auxiliary state variable(s) accordingly.

Assuming both the *Eventually* and the *Until* operators are present, we may append the auxiliary states y, ζ and η to the original state variable x and obtain a new state variable

$$z = \begin{bmatrix} x \\ y \\ \zeta \\ \eta \end{bmatrix} \text{ and } z_{k+1} = f_z(z_k, u_k) = \begin{bmatrix} f(x_k, u_k) \\ f_y(y_k, x_k, u_k) \\ f_\zeta(\zeta_k, x_k, u_k) \\ f_\eta(\eta_k, \zeta_k, x_k, u_k) \end{bmatrix}. \tag{8}$$

We call the dynamics for the auxiliary variables y, ζ and η (5), (6), and (7) "STL subdynamics", since they correspond to the subvector of the last three lines in (8).

To replace the STL specified state constraints with these newly defined dynamical equations, one just need to add a simple convex state constraint with respect to the associated auxiliary variable, to ensure the robust satisfaction of the original constraints. Use the *Until* operator as an example, we have

Problem 4. **Optimal Control with the *Until* Specifications**

$$\begin{aligned} \min_u \quad & C(x, u) \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k), \\ & \rho^{\varphi_1 \mathcal{U}_{[a,b]} \varphi_2}(x, k) \geq 0. \end{aligned} \tag{9}$$

Problem 5. **Optimal Control with the *Until* Subdynamics**

$$\begin{aligned} \min_u \quad & C(x, u) \\ \text{s.t.} \quad & z_{k+1} = f_z(z_k, u_k), \\ & \eta_{k^{(b)}} \geq 0. \end{aligned} \tag{10}$$

and the following theorem:

Theorem 3.1. *Problem 4 and Problem 5 are equivalent.*

Proof. Directly from the reasoning in Section 3.1.2. □

One clear advantage of using this transformation is the dramatic simplification of gradient computation when doing convexification in (2a) and (2b). Previously, the temporal specification is over a time window, which could include a large number of sampling instances. Since one needs to apply the chain rule to calculate the gradient, too many terms will make the gradient formula very difficult to obtain. In contrast, the "STL subdynamics" approach exploits the recursive structure of the STL semantics, and temporally group all the terms together in the same form (the auxiliary subdynamics). Additionally, by appending the original state vector, we are essentially solving the same problem as vanilla SCvx does, just with an expanded state space, which also makes implementation much easier.

3.2 Polynomial smooth min smin()

Now that we have transformed the STL specified constraints into subdynamics, the next step is to make the corresponding dynamical equations smooth so that we can apply SCvx to solve. Note that in (5), (6), and (7), we still have non-differentiable min- or max-functions. One can certainly use the log-sum-exponential function in (4) to approximate these min-/max- functions, but this approach is usually not accurate around non-differentiable points, especially with only two variables, which is exactly the case we have on hand. To solve this problem, we use the polynomial smooth min function $\texttt{smin}(a, b, k)$ defined as

$$\texttt{smin}(a, b, k) = \begin{cases} a, & \text{if } a - b \geq k, \\ b, & \text{if } a - b \leq -k, \\ g(a, b, k), & \text{if } a - b \in (-k, k), \end{cases} \tag{11}$$

where $g(a, b, k)$ is a smooth interpolator, and $k > 0$ controls the interpolation range. The idea of this smin() function is simple: we want to smoothly interpolate the values a and b if they are close to each other (i.e. $a - b \in (-k, k)$), otherwise we return the true minimum. A polynomial smooth interpolator function is given in [19] with a a rather simple derivation:

$$g(a, b, k) = a(1 - h) + hb - kh(1 - h), \tag{12}$$

where $h = \frac{1}{2} + \frac{a-b}{2k}$. One can easily verify that with this $g(a, b, k)$ in (12), the smin() function in (11) is continuously differentiable.

To the authors knowledge, this concept of polynomial smooth min function goes back to a blog post decade ago [17], though both [17] and [19] use it on smoothing the edges in computer graphics applications. The authors have not seen this technique been used in optimization settings, probably due to the limitation of only accepting two variables. However, it lends itself perfectly for smoothing the STL subdynamics, because we only need to calculate the min or max of two values at a time thanks to the recursive nature of dynamical equations.

3.3 Solving with SCvx

In (5), (6), and (7), replace the min operator with smin() and the max operator with −smin(). Denoting the resulting smooth dynamics as $\tilde{f}_y(), \tilde{f}_\zeta(), \tilde{f}_\eta()$ and $\tilde{f}_z()$, we arrive at the optimal control problem with smooth STL subdynamics, Problem 6 (Using *Until* as an example again).

Problem 6. **Optimal Control with Smooth Subdynamics**

$$\begin{aligned} \min_u \quad & C(x, u) \\ \text{s.t.} \quad & z_{k+1} = \tilde{f}_z(z_k, u_k), \\ & \eta_{k^{(b)}} \geq 0. \end{aligned} \tag{13}$$

This problem is now finally ready to be solved by the SCvx algorithm. Therefore, we summarize the entire SCvx-STL solution method as a high-level Algorithm 2.

Algorithm 2 The SCvx-STL Algorithm

1: **procedure** SCvx-STL(Problem 1 with an *Until* specification)
2: **step 1** Use robust *Until* semantics in Definition 2.2 to obtain a real-valued problem, Problem 4.
3: **step 2** Follow the steps in section 3.1 to transform STL specified constraints into STL subdynamics (8), and thus obtain Problem 5.
4: **step 3** Smooth the STL subdynamics with polynomial smooth min function smin() in (11) and obtain the smooth optimal control problem, Problem 6.
5: **step 4** Solve Problem 6 with the SCvx algorithm, Algorithm 1, and obtain the final result.
6: **end procedure**

4 NUMERICAL RESULTS

In this section, we use a quadrotor obstacle avoidance problem to preliminarily demonstrate the capability of the proposed SCvx-STL algorithm. The non-convexity of this problem stems from cylindrical obstacle keep-out zones (see Figure 1), nonlinear aerodynamic drag, and the *Until* specified STL constraints.

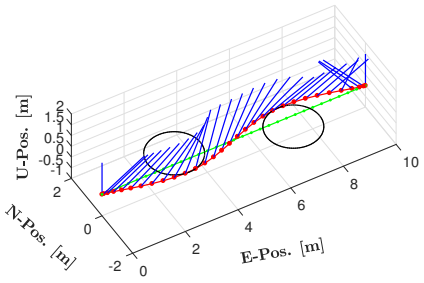

Figure 1: A perspective view of the problem setup. The obstacles are represented by the black circles. The red dots and blue lines represent the time discretized positions and thrust vectors, respectively. The green dots represent the initial trajectory (i.e., a straight line). The motion of the vehicle is from left to right.

The physical parameters for this numerical experiment closely follows [14], with the exception of STL related parameters and constraints. For a detailed description of the physical model and the problem/parameters setup, the reader is referred to [14].

The STL specified requirement we choose to simulate in this experiment is an *Until* statement: "Do not get in certain range of the destination *Until* a thrust reducing maneuver has been performed within a time window." This is a typical requirement for close proximity operations. This translate to $\rho^{\varphi_1} \mathcal{U}_{[a,b]} \varphi_2(x, t_0) \geq 0$. Here $\rho^{\varphi_1}(x, u) = \|x - x_f\| - R$ and $\rho^{\varphi_2}(x, u) = T_r - \|u\|$, and R is the keep-out range, while T_r is the reduced thrust limit. The value of these parameters selected for this experiment is as follows. $a = 0(s), b = 2.4(s), R = 3(m)$ and $T_r = T_{\max}/2$.

Algorithm 2 was implemented in MATLAB using CVX [8] and the SDPT3 [23] solver. For onboard real-time computation, one might want to use customized convex optimization solvers that make use of the problem structures. For example, the solver in [7] generates aggressive quadrotor guidance trajectories onboard at rates exceeding 8 Hz. It takes around 22 successful outer iterations to converge for this experiment, which just adds a few more iterations to the one in [14]. Not a bad trade-off for including an STL requirement.

Figure 2 shows the top view of the converged trajectory, as well as thrust profiles (both tilt and magnitude). The converged trajectory is feasible in the physical space as shown in the figure. Note that a clear thrust reducing maneuver is performed around time 2.2s, which is within the preset time window [0, 2.4s]. One can clearly observe the differences in the thrust magnitude between here in Figure 2 and the one in [14], which is exactly the effect of the additional STL requirement.

Figure 3 shows the robust measures of ρ^{φ_1} and ρ^{φ_2}. One can easily see that ρ^{φ_1} holds *Until* ρ^{φ_2} holds, which verifies the temporal requirement $\rho^{\varphi_1} \mathcal{U}_{[a,b]} \varphi_2(x, t_0) \geq 0$ for this experiment.

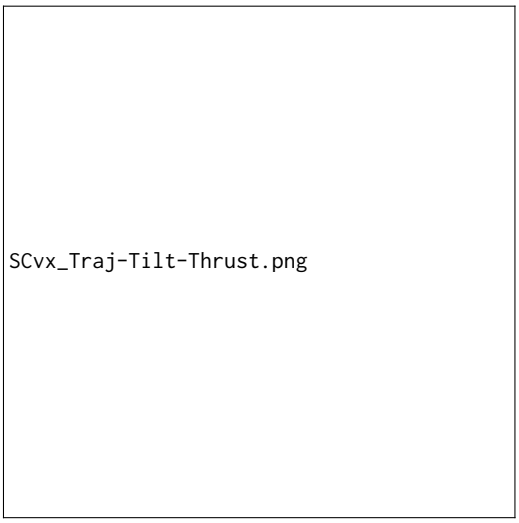

Figure 2: Converged trajectory and thrust profile. Feasibility in physical space and in terms of temporal requirement is validated.

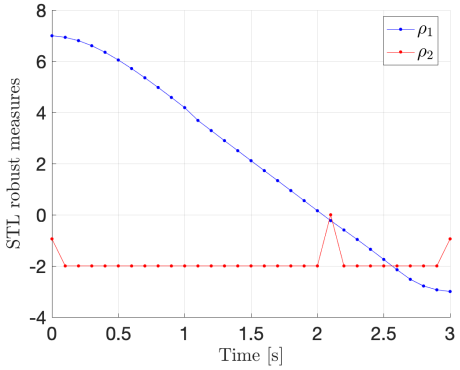

Figure 3: STL robust measures of ρ^{φ_1} and ρ^{φ_2}. The pattern verifies $\rho^{\varphi_1} \mathcal{U}_{[a,b]} \varphi_2(x, t_0) \geq 0$.

ACKNOWLEDGMENTS

The authors would like to thank ANR project FEANICSES (ANR-17-CE25-0018), Office of Naval Research (N00014-16-1-3144), National Science Foundation (NSF-CMMI-1613235), and Air Force Research Laboratory (FA9453-15-1-0303) for the support.

REFERENCES

[1] [n. d.]. International Rendezvous System Interoperability Standards (IRSIS). https://www.internationaldeepspacestandards.com

[2] B. Açıkmeşe, J.M. Carson, and L. Blackmore. 2013. Lossless Convexification of Non-convex Control Bound and Pointing Constraints of the Soft Landing Optimal Control Problem. *IEEE Transactions on Control Systems Technology* 21, 6 (2013), 2104–2113.

[3] B. Açıkmeşe and S. R. Ploen. 2007. Convex Programming Approach to Powered Descent Guidance for Mars Landing. *AIAA Journal of Guidance, Control and Dynamics* 30, 5 (2007), 1353–1366.

[4] Nicola Bernini, Mikhail Bessa, Rémi Delmas, Arthur Gold, Eric Goubault, Romain Pennec, Sylvie Putot, and François X. Sillion. 2021. A few lessons learned in reinforcement learning for quadcopter attitude control. In *HSCC '21: 24th ACM International Conference on Hybrid Systems: Computation and Control, Nashville, Tennessee, May 19-21, 2021*, Sergiy Bogomolov and Raphaël M. Jungers (Eds.). ACM, 27:1–27:11. https://doi.org/10.1145/3447928.3456707

[5] Lars Blackmore. 2016. Autonomous Precision Landing of Space Rockets. *The Bridge on Frontiers of Engineering* 4, 46 (2016), 15–20.

[6] Alexandre Donzé and Oded Maler. 2010. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 92–106.

[7] Daniel Dueri, Behçet Açıkmeşe, Daniel P Scharf, and Matthew W Harris. 2016. Customized Real-Time Interior-Point Methods for Onboard Powered-Descent Guidance. *Journal of Guidance, Control, and Dynamics* (2016), 1–16.

[8] Michael Grant and Stephen Boyd. 2014. CVX: Matlab Software for Disciplined Convex Programming, version 2.1. http://cvxr.com/cvx.

[9] D.G. Hull. 1997. Conversion of Optimal Control Problems into Parameter Optimization Problems. *Journal of Guidance, Control, and Dynamics* 20, 1 (1997), 57–60.

[10] Unsik Lee and Mehran Mesbahi. 2017. Constrained Autonomous Precision Landing via Dual Quaternions and Model Predictive Control. *Journal of Guidance, Control, and Dynamics* 40 (2017), 292–308.

[11] Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 152–166.

[12] Danylo Malyuta, Yue Yu, Purnanand Elango, and Behcet Acikmese. 2021. Advances in Trajectory Optimization for Space Vehicle Control. arXiv:2108.02335 [math.OC]

[13] Yuanqi Mao, Daniel Dueri, Michael Szmuk, and Behçet Açıkmeşe. 2017. Successive Convexification of Non-Convex Optimal Control Problems with State Constraints. *IFAC-PapersOnLine* 50, 1 (2017), 4063 – 4069. https://doi.org/10.1016/j.ifacol.2017.08.789

[14] Yuanqi Mao, Michael Szmuk, and Behçet Açıkmeşe. 2019. Successive Convexification: A Superlinearly Convergent Algorithm for Non-convex Optimal Control Problems. *arXiv e-prints*, Article arXiv:1804.06539 (Feb. 2019), arXiv:1804.06539 pages. arXiv:1804.06539 [math.OC]

[15] Yuanqi Mao, Michael Szmuk Szmuk, and Behçet Açıkmeşe. 2016. Successive convexification of non-convex optimal control problems and its convergence properties. In *IEEE 55th Conference on Decision and Control (CDC)*. 3636–3641.

[16] Yash Vardhan Pant, Houssam Abbas, Rhudii A Quaye, and Rahul Mangharam. 2018. Fly-by-logic: control of multi-drone fleets with temporal logic objectives. In *ACM/IEEE ICCPS*. IEEE, 186–197.

[17] Inigo Quilez. 2013. *smooth minimum - 2013.* Retrieved Oct 29, 2021 from https://iquilezles.org/www/articles/smin/smin.htm

[18] Vasumathi Raman, Alexandre Donzé, Mehdi Maasoumy, Richard M Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A Seshia. 2014. Model predictive control with signal temporal logic specifications. In *CDC'14*. IEEE, 81–87.

[19] Vinicius Santos. 2016. *Smooth Min Explained.* Retrieved Oct 29, 2021 from http://www.viniciusgraciano.com/blog/smin

[20] Michael Szmuk and Behçet Açıkmeşe. 2018. Successive Convexification for 6-DoF Mars Rocket Powered Landing with Free-Final-Time. *ArXiv e-prints* (Feb. 2018). arXiv:1802.03827.

[21] Michael Szmuk, Utku Eren, and Behçet Açıkmeşe. 2017. Successive Convexification for Mars 6-DoF Powered Descent Landing Guidance. In *AIAA Guidance, Navigation, and Control Conference*. 1500.

[22] M. Szmuk, C. A. Pascucci, D. Dueri, and B. Açıkmeşe. 2017. Convexification and real-time on-board optimization for agile quad-rotor maneuvering and obstacle avoidance. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 4862–4868.

[23] K.C. Toh, M.J. Todd, and R.H. Tutuncu. 1999. SDPT3 — a Matlab Software Package for Semidefinite Programming. *Optimization Methods and Software* 11, 1 (1999), 545–581.