Drive-by-Logic: Trajectory Generation for Nonholonomic Ground Robots with Signal Temporal Logic Objectives

Praneeth Kolapalli, Stephen L. Smith, Yash Vardhan Pant

***Abstract*— Autonomous mobile robots are actively applied to execute complex tasks, such as package delivery, autonomous taxiing, and search-and-rescue. Signal Temporal Logic (STL) offers a powerful formalism for such complex tasks. However, designing plans (trajectories) that satisfy tasks formalized by STL grammar, particularly for nonholonomic systems such as a car-like robot or a fixed-wing aircraft, is a challenging problem. This paper proposes a method to generate trajectories for a multi-robot system with car-like robots to perform complex tasks specified with STL grammar. The proposed method solves a nonlinear program (NLP) to construct trajectories with several constant curvature curves that satisfy the specification. In doing so, it also guarantees the kinematic feasibility of the solution trajectories. Extensive simulation studies show that the proposed method finds satisfying solutions $4\times$ faster than a model predictive control baseline. Additionally, it is able to construct trajectories for complex STL specifications that the baseline fails to satisfy.**

I. Introduction

Autonomous mobile ground robots/vehicles are carrying out increasingly complex tasks, such as autonomous driving [1], last-mile package delivery [2], and infrastructure monitoring and surveillance [3]. Signal Temporal Logic (STL) [4] offers a way to encode complex operating requirements in a mathematically unambiguous manner. The problem of designing controllers or generating motion plans for dynamical systems to satisfy STL specifications has been extensively studied [5]–[8] (see Section II for an overview of relevant literature).

In this paper, we consider the problem of generating trajectories for nonholonomic ground robots with curvature and kinematic constraints, subject to complex tasks encoded via STL. The nonlinear and nonholonomic dynamics of car-like ground robots make this a particularly challenging problem. Similar to [6] and [8], we propose a trajectory generation method for one or more robots subject to an STL specification. However, unlike those methods, we aim to develop a computationally efficient method to generate trajectories satisfying the STL specification, and are a state solution to the nonholonomic robot model

Contributions: This paper proposes a method for generating trajectories for one or more nonholonomic ground robots subject to an STL specification. The generated trajectories

1) are provably a state solution to the nonholonomic kinematic bicycle model for car-like robots;
2) respect bounds on velocities, accelerations, and maximum curvature that a robot can realize, i.e., the physical constraints; and
3) satisfy STL specifications generated using the entire grammar of STL with bounded temporal operators and possibly non-convex predicates, i.e., leverages STL's full expressivity.

Through extensive simulation studies, we demonstrate the applicability of the proposed method and highlight its performance and scalability in multi-robot motion planning compared to a nonlinear Model Predictive Control(MPC) baseline [9]. We also implement the method on a $1/10^{\text{th}}$ scale ground robot and demonstrate the ease of tracking the generated trajectories via an off-the-shelf controller.

Outline of the paper: Section II presents an overview of controller design and motion planning approaches for dynamical systems to realize STL specifications. Section III presents the required background, introduces mathematical notation, and formally states the problem we aim to solve. Section IV introduces a class of feasible trajectories for the kinematic bicycle model and introduces the optimization formulation for multi-nonholonomic robot trajectory planning to satisfy STL specifications. Section V presents simulation case studies and an experimental implementation to validate the performance of our proposed method. Finally, we discuss our approach's limitations and future work in Section VI.

II. Related Work

Control [5, 9], and trajectory planning [6, 8, 10] for dynamical systems satisfying an STL specification is now a well-studied problem. A mixed integer linear programming based encoding of STL specifications was introduced in [5], based on the work in [11], and further developed in works such as [7, 12]. These works are primarily limited to linear time-invariant systems and do not scale well in multi-agent settings [9]. Recent works [13]–[15] have developed time-varying control barrier functions (CBF) to control systems with STL specifications but are limited only to a fragment of STL or to convex predicates. Another line of work [9, 16] develops smooth (continuously differentiable) approximations of the robust semantics of STL and uses gradient-based control optimization for systems to satisfy STL specifications. [17, 18] present sampling-based methods to satisfy STL specifications with RRT* and biased sampling respectively. [19]–[21] develop learning-based method to solve for control actions that satisfy STL specifications.

This work is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and Canada Foundation for Innovation - John R. Evans Leaders Fund (CFI JELF).

The authors are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Canada. vkpkolapalli@uwaterloo.ca, stephen.smith@uwaterloo.ca, yash.pant@uwaterloo.ca

To overcome the limitations of scaling such methods to complex STL specifications, particularly involving multi-robot systems, [6, 8] instead frame the problem as trajectory generation via selecting timed waypoints, rather than low-level control of a dynamical system. In this work, we take a similar approach. [6] uses quintic splines for trajectories of multi-rotor aerial robots, and [8] uses piece-wise linear trajectory representations for general dynamical systems. While a car-like robot can track these trajectories, the tracking error would be non-negligible and require specially designed tracking controllers [22]. [10, 23] use Bezier curves to represent trajectories. The method in [10] generates kinematically feasible trajectories, which can also be tracked by a car-like robot well with a low-level controller, but is limited to STL specifications with affine predicates and does not explicitly generate trajectories that naturally arise from car-like dynamics. Here, we develop a trajectory generation method that can handle the entire grammar of STL, with non-linear predicates. The trajectories are state-solutions to kinematic bicycle model and can be guaranteed to respect the kinematic constraints of a robot. Through an experiment on a real robot, we also show that an off-the-shelf low-level controller can track these trajectories well.

III. Preliminaries

We consider nonholonomic ground robots that are modeled via the kinematic bicycle model[1] [25]:

$$\begin{aligned} &\dot{x}(t) = v(t)\cos\big(\theta(t)\big), \quad \dot{y}(t) = v(t)\sin\big(\theta(t)\big), \\ &\dot{v}(t) = a(t), \quad \dot{\theta}(t) = v(t)\kappa(t). \end{aligned} \tag{1}$$

Here, $x(t)$ and $y(t)$ denote the position of the robot in a planar coordinate frame, $\theta(t)$ the orientation of the robot (with respect to the positive x-axis), $v(t)$ is the speed of the robot along the direction of motion, $a(t)$ the acceleration and $\kappa(t)$ the curvature at time t seconds. The wheelbase of the robot is L meters. The steering angle $\delta(t)$ of the robot is determined from the input $\kappa(t)$ by the relation $\delta(t) = \tan^{-1}(L\kappa(t))$. Additionally, We use $z(t) = [x(t),\, y(t),\, v(t),\, \theta(t)]' \in \mathcal{Z} \subseteq \mathbb{R}^3 \times \mathbb{S}^1$ to denote the state and $u(t) = [a(t),\, \kappa(t)]' \in \mathbb{R}^2$ as the input to the system at time t.

Furthermore, we define bounds on the inputs and states for (1). Bounds on position $x(t), y(t)$ are workspace bounds, orientation $\theta(t)$ is unbounded in $\mathbb{S}^1$, speed $v(t)$ is bounded within $[\underline{v}, \bar{v}]$, acceleration $a(t)$ is bounded in $[\underline{a}, \bar{a}]$, and finally, curvature κ is bounded in $[-\kappa_{\max}, \kappa_{\max}]$, where $\kappa_{\max}$ is computed as the inverse of the minimum turning radius of the model or from the maximum steering angle. We refer to these bounds as *kinematic constraints*.

Let $\mathcal{R} = \{1, \ldots, R\}$ denote the set of all robots of interest. We use $\mathbf{z}_r : [0, T] \to \mathcal{Z}$ to represent a trajectory (in 2D position space, speed, along with orientation) over a time interval $[0, T]$ for a robot $r \in \mathcal{R}$. For ease of notation, we drop the subscript r when it is unnecessary. We use Z to denote the set of all trajectories, or signals, $Z = \{\mathbf{z} \mid \mathbf{z} : [0, T] \to \mathcal{Z}\}$.

[1]This is a generalized version of a Dubin's car [24], controlled by curvature (or steering angle) and acceleration.

A. A Brief Overview of Signal Temporal Logic

We represent complex tasks using STL [26], a behavioral specification language that formally represents the desired behavior of the system over space, time, and multiple agents/robots. We present a brief introduction to STL and point the interested reader to [4] for more details. An STL specification, φ, builds on a set of *predicates*, where a predicate is defined as $q := \mu(\mathbf{z}) \geq 0$. Here, $\mu : Z \to \mathbb{R}$ is a real-valued function. In addition, let $I \subset \mathbb{R}$ be a non-singleton time interval, and let $\top$, $\neg$, and $\wedge$ represent the boolean true, negation, and AND operators, respectively. STL also uses the temporal Until operator, $\mathcal{U}_I$, defined over a time interval I. An STL specification, φ, is then recursively built from these elements using the grammar

$$\varphi := \top | q | \neg\varphi | \varphi_1 \wedge \varphi_2 | \varphi_1 \mathcal{U}_I \varphi_2.$$

Here, $\varphi_1 \mathcal{U}_I \varphi_2$ implies that φ_2 must hold at some point in time in the interval I, and *until* then, φ_1 should hold throughout. Other operators, such as disjunction ($\vee$), implication ($\Rightarrow$), always ($\Box_I$), and eventually ($\Diamond_I$) can be derived from the operators defined above. An STL specification then can be interpreted as a function that goes from the space of trajectories to a boolean true or false, i.e., $\varphi : Z \to \{\top, \bot\}$.

Example 1 (Autonomous Overtaking)**.** *Consider a case where a connected autonomous vehicle (CAV), car A, needs to overtake another CAV (See Figure 1), car B, before an upcoming exit in a multi-lane roadway shown in Figure 1. The position of a car j is defined by $p_j(t) = [p_{j,x},\, p_{j,y}]' \in \mathbb{R}^2$, where, $j \in \{A, B\}$ Car A must safely overtake car B (which transmits its planned path to car A) by reaching the green region (G) within the next 40s, at which point it should be at least two car lengths $2L$, where L is the wheelbase, ahead of car B. The overtake requirement can be encoded in STL as*

$$\varphi_{\textit{overtake}} = \Diamond_{[0,40]}(p_A \in G \Rightarrow p_{A,x} - p_{B,x} \geq 2L). \tag{2}$$

Safety is defined by the two vehicles maintaining a distance of at least $0.5d$ in the y-direction and L in the x-direction; this can be encoded as

$$\begin{aligned} \varphi_{\textit{safety}} = \Box_{[0,40]}((|p_{A,x} - p_{B,x}| \geq L) \\ \wedge (|p_{A,y} - p_{B,y}| \geq 0.5d)). \end{aligned} \tag{3}$$

Additionally, the robot's workspace is limited to the lane boundaries, but it is not encoded as an STL specification. Finally, safe overtaking is represented via the specification

$$\varphi_{\textit{safe overtake}} = \varphi_{\textit{overtake}} \wedge \varphi_{\textit{safety}}. \tag{4}$$

We consider a similar setup as a case study in Section V. Note that STL was defined initially over continuous time signals but also applies to discrete-time signals [5].

Remark III.1 (Bounded STL)**.** *We consider only* bounded *STL specifications in this work, i.e., those requiring only a finite duration trajectory for evaluation. For example, the*

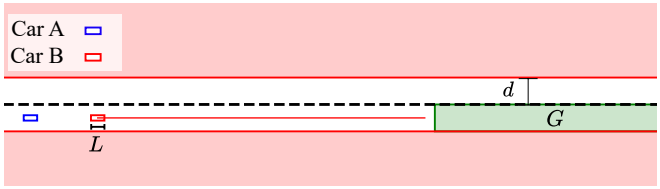


Fig. 1: Workspace of Example 1, here Car A is expected to safely overtake Car B before reaching G (4).

specification $\varphi_{\text{safe overtake}}$ requires a trajectory of a duration of 40s (at the least) for evaluation. We refer to this duration as the formula horizon *and denote it as H_φ for a specification φ.*

1) Robustness of STL formula: Associated with an STL specification φ, there exists a *robustness function* [4] which can be constructed following the grammar of STL and returns a robustness value with respect to a signal $\mathbf{z}$, $\rho_\varphi : Z \to \mathbb{R} \cup \{-\infty, \infty\}$. It has the property that for any STL formula φ, if $\rho_\varphi(\mathbf{z}) > 0$, then $\mathbf{z}$ satisfies φ (denoted as $\mathbf{z} \models z$). If $\rho_\varphi(\mathbf{z}) < 0$, then $\mathbf{z}$ violates φ. The case $\rho_\varphi(\mathbf{z}) = 0$ is inconclusive. There also exists a smooth, or continuously differentiable approximations of the robustness function [9, 16]. From the *smooth robustness* function, $\tilde{\rho}_\varphi$ defined in [9], $\tilde{\rho}_\varphi(\mathbf{z}) > \gamma_\varphi$ implies $\mathbf{z} \models \varphi$, where γ_φ is a tune-able parameter (see [9] for details). Such a smooth robustness function allows for gradient-based optimization to maximize it, generating signals that satisfy a STL specification [6].

B. Problem Statement

In this paper, we develop a trajectory generation method for solving the following problem:

Problem 1 (Multi nonholonomic robot trajectory planning for complex tasks)**.** *Given a multi-robot task encoded via the STL specification φ defined over trajectories of R robots modeled via* (1)*, generate trajectories $\mathbf{z}_r \, \forall r \in \mathcal{R}$ such that:*

1) Task satisfaction: $(\mathbf{z}_1, \ldots, \mathbf{z}_R) \models \varphi$*, i.e., all robots satisfy the STL specification φ, which is defined over trajectories of all robots in $\mathcal{R}$.*
2) *All trajectories $\mathbf{z}_r$ are valid state solutions to the kinematic bicycle model* (1)*.*
3) Kinematic feasibility:[2] *The trajectories respect predefined bounds on speed $v_r(t) \in [\underline{v}, \bar{v}]$, acceleration $a_r(t) \in [\underline{a}, \bar{a}]$, and curvature $\kappa_r(t) \in [-\kappa_{max}, \kappa_{max}]$ for all robots $r \in \mathcal{R}$ and for all time $t \in [0, H_\varphi]$.*

IV. Trajectory generation for satisfying STL specifications with Car-Like robots.

To solve Problem 1, we aim to generate trajectories of (1) that solve a given STL specification φ without having to optimize over the low-level control signal $u(t)$, $\forall t \in [0, H_\varphi]$ (or its discrete-time counterpart) directly. To do so, we create trajectories that consist of segments connecting successive *waypoints*, with the waypoints being uniformly spaced in time with a period Δt. We define a waypoint w_i as:

$$w_i := [x_i,\, y_i,\, v_i,\, a_i,\, \kappa_i,\, \theta_i]' \in \mathcal{W} \subseteq \mathbb{R}^5 \times \mathbb{S}^1.$$

[2]For simplicity, and without loss of generality, we assume all robots have homogeneous kinematic constraints and model parameters in (1).

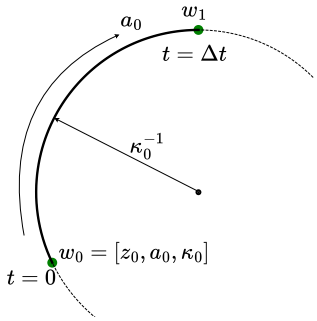


Fig. 2: Two waypoints w_0, w_1 that are Δt apart in time connected by (5).

Using a slight modification of the parametrization of a Dubin's curve defined in [24], the trajectory between two waypoints w_0 and w_1 in terms of the states of (1) is

$$\begin{aligned}
&\forall t \in [0, \Delta t]: \\
&v(t) = v_0 + t a_0 \\
&\theta(t) = \theta_0 + t\kappa_0 v(t) \\
&x(t) = x_0 + t v(t)\,\mathrm{sinc}\left(\frac{\theta(t)-\theta_0}{2}\right)\cos\left(\frac{\theta(t)+\theta_0}{2}\right) \\
&y(t) = y_0 + t v(t)\,\mathrm{sinc}\left(\frac{\theta(t)-\theta_0}{2}\right)\sin\left(\frac{\theta(t)+\theta_0}{2}\right)
\end{aligned} \tag{5}$$

This corresponds to the robot (1) traveling along a curve of constant curvature κ_0 with constant acceleration a_0, traveling from w_0 to w_1 in time Δt seconds, as shown in Figure 2. The boundary constraints of the parametric curve (5) are thus defined by the initial state of the robot, $z(0)$, and the state of the robot at time Δt, $z(\Delta t)$:

$$\begin{aligned}
z(0) &= [x(0), y(0), v(0), \theta(0)] \\
z(\Delta t) &= [x(\Delta t), y(\Delta t), v(\Delta t), \theta(\Delta t)].
\end{aligned} \tag{6}$$

Lemma IV.1 ([24])**.** *The parametric equation* (5) *is a state solution of the non-holonomic kinematic bicycle model* (1)*.*

The interested reader can refer to [24] and [27] for details.

A. Selecting Kinematically Feasible Waypoints

We aim to generate trajectories that respect given kinematic bounds for a robot (Problem 1), i.e., $\forall t$, $a(t) \in [\underline{a}, \bar{a}]$, $v(t) \in [\underline{v}, \bar{v}]$, and $\kappa(t) \in [-\kappa_{\max}, \kappa_{\max}]$. In this section, we derive constraints on a pair of successive waypoints w_i and w_{i+1} such that the trajectory segment (5) connecting them is *kinematically feasible* for all time $t \in [i\Delta t, (i+1)\Delta t]$. For ease of notation, let $i = 0$ in the following discussion.

First, note from (5) that the speed, $v(t)$, monotonically increases or decreases between two waypoints. Therefore, it is sufficient to constrain the speed components of the

waypoints to be within the given speed bounds, i.e., $v_0, v_1 \in [\underline{v}, \bar{v}]$. From the speed component in (5), we also see that $v_1 = v(\Delta t) = v_0 + \Delta t a_0$. Given the acceleration bounds $a(t) \in [\underline{a}, \bar{a}]$, we then also need to ensure that the commanded change in velocities does not require acceleration outside of this bound, i.e.,

$$a_0 = \frac{v_1 - v_0}{\Delta t} \in [\underline{a}, \bar{a}] \Rightarrow v_1 - v_0 \in [\Delta t \underline{a}, \Delta t \bar{a}]. \tag{7}$$

Next, we can also impose a bound on the acceleration along the curve, a_0, based on the maximum and minimum change in speed between waypoints w_0 and w_1, i.e.,

$$a_0 \in \left[\frac{v_0 - \underline{v}}{\Delta t}, \frac{\bar{v} - v_0}{\Delta t}\right] \cap [\underline{a}, \bar{a}]. \tag{8}$$

Finally, we can pick waypoint curvatures κ_0, κ_1 such that they respect the curvature constraint of the robot, i.e, $\kappa_0, \kappa_1 \in [-\kappa_{\max}, \kappa_{\max}]$.

Definition IV.1 (Pairwise waypoint constraint set $\Delta\mathcal{W}_{\text{feas}}$)**.** *Let w_i and w_{i+1} be two waypoints defining the boundary constraints* (6) *of the motion primitive* (5)*, then, we say that they respect a pairwise waypoint constraint denoted by $w_{i+1} - w_i \in \Delta\mathcal{W}_{feas}$, if the waypoint pair respects the constraints in* (7)*,* (8)*, $a_i, a_{i+1} \in [\underline{a}, \bar{a}]$, $v_i, v_{i+1} \in [\underline{v}, \bar{v}]$, and $\kappa_i, \kappa_{i+1} \in [-\kappa_{max}, \kappa_{max}]$.*

As discussed above, this ensures that velocities, accelerations and curvature is within prescribed bounds for the trajectory segment (5) connecting the two waypoints w_i and w_{i+1}. This is formalized below:

Lemma IV.2 (Kinematic feasibility of waypoint pairs)**.** *If two successive waypoints w_i and w_{i+1} respect the pairwise constraint in Definition IV.1, i.e., $w_{i+1} - w_i \in \Delta\mathcal{W}_{feas}$, then the resulting trajectory* (5) *connecting them is kinematically feasible, i.e., $\forall t \in [i\Delta t, (i+1)\Delta t]$, we have $a(t) \in [\underline{a}, \bar{a}]$, $v(t) \in [\underline{v}, \bar{v}]$, and $\kappa(t) \in [-\kappa_{max}, \kappa_{max}]$.*

Proof sketch: The proof follows directly from the construction earlier in this section, leveraging the parametric curve (5) that defines the trajectory segment between two waypoints.

Now, given a sequence of $N+1$ waypoints, $\mathbf{w} = [w_0, w_1, \ldots, w_N]$, we can then obtain a trajectory of duration $N\Delta t$ seconds where each successive pair of waypoints is connected by the parametric curve in (5). If we constrain the waypoints according to Definition IV.1, then the entire trajectory is kinematically feasible.

Remark IV.1. *We use the notation $f_{kin}(\mathbf{w})$ to define the trajectory generated by applying* (5) *to each pair of successive waypoints,*

$$f_{kin} : \mathcal{W}^{N+1} \to \mathcal{Z} \times [0,\ N\Delta t].$$

We also define a discrete-time version of this trajectory, sampled at a rate of $1/h$ Hz as

$$f^h_{kin} : \mathcal{W}^{N+1} \to \mathcal{Z}^{\frac{N\Delta t}{h}}.$$

Here, we use shorthand $\mathcal{W}^{N+1}$ to denote the Cartesian product of $N+1$ spaces defined by $\mathcal{W}$, i.e., $\mathcal{W}^{N+1} = \mathcal{W} \times \mathcal{W} \times \ldots \times \mathcal{W}$.

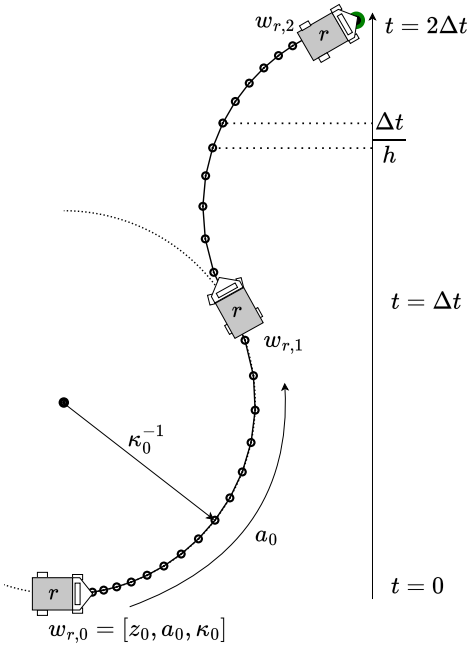


Fig. 3: The figure shows the discretization of the parametric curve given by (5). Each waypoint is Δt apart in time, and all samples are $\Delta t/h$ apart.

Figure 3 shows a continuous-time trajectory formed by three waypoints and the corresponding discrete-time trajectory. Note, $\Delta t \gg h$, i.e., the time between waypoints is greater than the discretization period of the low-level dynamics (1).

B. Trajectory Generation for STL Specifications

We now have a framework to represent (discrete-time) trajectories for nonholonomic robots (1) via a sequence of waypoints. We consider $N+1$ waypoints, per robot, $\mathbf{w}_r$, that are Δt apart in time and recursively joined by (5).

The resulting (discrete-time) trajectories for all $r \in \mathcal{R}$ robots are defined by the function $F(\mathbf{w}_1, \ldots, \mathbf{w}_R) = [f^h_{\text{kin}}(\mathbf{w}_1), \ldots, f^h_{\text{kin}}(\mathbf{w}_R)]$, i.e., a mapping from waypoints to trajectories for each robot following the parametric curve (5).

We can now formulate an optimization over waypoints to generate discrete-time trajectories for each robot r such that they satisfy a given task represented by an STL specification φ, while ensuring that the trajectories are kinematically feasible (Lemma IV.2).

Given R robots, and their initial states $[z_{1,0}, \ldots, z_{R,0}]$ (positions, speeds, and orientations), we formulate an optimization to maximize the *smooth robustness* (see Section III) of the resulting trajectories:

$$\mathcal{P}_{\text{DBL}} : \underset{\mathbf{w}_1, \ldots, \mathbf{w}_R}{\text{maximize}}\ \tilde{\rho}_\varphi(F(\mathbf{w}_1, \ldots, \mathbf{w}_R)) \tag{9a}$$

$$\forall i \in \{0, \ldots, N-1\},\ \forall r \in \mathcal{R},$$

$$w_{r,i+1} - w_{r,i} \in \Delta\mathcal{W}_{\text{feas}} \quad \text{(Definition IV.1)} \tag{9b}$$

$$\tilde{\rho}_\varphi(F(\mathbf{w}_1, \ldots, \mathbf{w}_R)) \geq \gamma_\varphi \tag{9c}$$

Also included are constraints to account for the initial states of the robots and on the last waypoint for each robot such that the terminal speed of each trajectory is zero. We omit writing these explicitly (above) to avoid clutter. Here, the objective, $\tilde{\rho}_\varphi$ is the smooth robustness over trajectories as defined in [9], but can be replaced by any *smooth robustness* variant such as in [16, 28]. The final constraint requires that the resulting trajectories have a robustness of at least γ_φ. A feasible solution to the optimization (9) is a solution to Problem 1.

Theorem IV.1. *Consider an STL specification φ defined over R nonholonomic robots* (1) *with pre-defined kinematic bounds. Then, given the initial states for R robots a feasible solution to the optimization* (9) *generates $\{\mathbf{w}_1, \ldots, \mathbf{w}_R\}$ (a sequence of waypoints for each robot) such that:*

1) *The trajectories across all robots, $F(\mathbf{w}_1, \ldots, \mathbf{w}_R)$ satisfy the given specification φ in continuous time.*
2) *The waypoints $\mathbf{w}_r$ define a trajectory $f_{kin}(\mathbf{w}_r)$ that is a state solution to* (1) *for each robot.*
3) *The trajectories are kinematically feasible, i.e., satisfy predefined bounds on speed, acceleration, and curvature.*

Proof. We provide a short proof sketch below:

1) Theorem 5.1 in [6] states that for a given specification φ and a trajectory sampling period of h seconds, we can find a lower bound on smooth robustness $\gamma_\varphi > 0$ such that a discrete-time trajectory satisfying constraint (9c) implies that the continuous time trajectory satisfies the specification φ.
2) Lemma IV.1 states that each trajectory segment between two waypoints is a valid solution to (1). By induction on the number of segments, the trajectory resulting from connecting N such segments is also a solution to (1).
3) Lemma IV.2 states that if waypoints respect Definition IV.1, then the resulting trajectory segment is kinematically feasible. A feasible solution to the optimization (9) also satisfies Definition IV.1 (via constraint (9b)); hence the resulting trajectory is kinematically feasible.

The statements above then prove Theorem IV.1. □

For a given specification φ and its associated specification horizon H_φ, the duration of the generated trajectory should be such that $N\Delta t \geq H_\varphi$. The number of waypoints, $N+1$, and the time between waypoints ΔT are design parameters in (9); selecting them requires domain expertise.

The trajectories generated by the waypoints selected by (9) are $\mathcal{C}^1$ - continuous. Since the arc segment between waypoints has constant acceleration and curvature, the commanded jerk and steering angles are discontinuous when the robot transitions from one segment to another. Consequently, the trajectories are not *perfectly* trackable and are only kinematically feasible. However, the tracking error is not noticeable enough to be a practical concern, as is demonstrated on a real robot in Section V.

Finally, note that the trajectory generation optimization (9) is a non-convex program; there is no guarantee that it will result in a trajectory that satisfies the STL specification, sometimes even in cases where a satisfying solution does exist. Multi-starting the optimization [6, 9] can overcome this in practice. However, as we will see in Section V, the proposed method does generate trajectories that satisfy the given specifications across all case studies.

V. Simulation Studies

We demonstrate the performance of our proposed method through multiple simulation studies. We compare our method to a baseline: an MPC-based approach [9] to maximize robustness over a sequence of vehicle controls (acceleration and steering angle (1)) discretized at a period of h seconds, spanning over a total of H_φ seconds. The collocation of these discretized controls are mapped into states for evaluating the robustness function.

A. Simulation Setup

We consider two modes of operation for both the proposed method (9) and the baseline:

Boolean mode, where the optimizations terminate after finding a feasible trajectory that satisfies the specification with a robustness value of at least γ_φ.

Robust mode, where the optimization is set to find trajectories that maximize the robustness function to a local optimum.

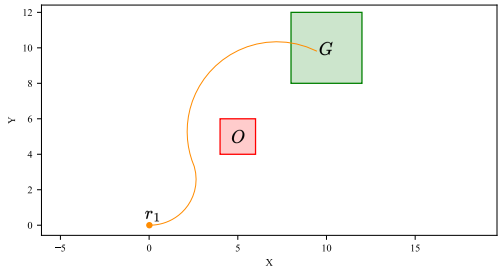
The two algorithms' modes call for different metrics for comparing the baseline and proposed approach. For the Robust mode, we compare the robustness achieved on termination. We compare the runtime and success rate for Boolean mode over a stipulated number of runs.

For all evaluations and metrics, we report the mean and standard deviation over 100 simulations with randomly generated initial configurations for the robots.

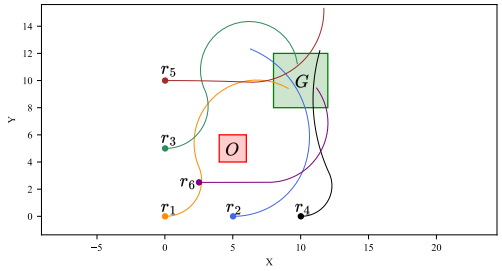
The simulations are of AgileX Limo [29] Robot's kinematics, and with the optimization formulated in Casadi [30] with Ipopt [31] as the NLP solver for both the baseline method and our proposed method (9). We set the maximum solver iteration to 50000, i.e., the algorithm is considered to time out if it does not find a satisfying solution in 50000 iterations. The reported times and simulations are from Intel i5-9300H processor (4.1 GHz) and 8Gb RAM running Ubuntu 22.04. The code is publicly hosted here: `https://github.com/CL2-UWaterloo/drive-by-logic.git`.

B. Case Study 1: Reach and Avoid

In an effort to benchmark the performance of our algorithm, we start with a reach and avoid specification. In this configuration, the robots in the environment are tasked to reach a goal set $G = [8, 12] \times [8, 12]$, while avoiding each other's paths by at least $d = 1.414$ and a set $O = [4, 6] \times [4, 6]$, within $T = 30$ seconds. Let $p_r = [x_r, y_r]$ be the position of the robot $r \in \mathcal{R}$, Then,



(a) Workspace and generated trajectory visualized for the reach avoid problem with $R = 1$ robots and $N = 3$ waypoints in robust mode.



(b) Workspace and generated trajectories visualized for the reach avoid problem with $R = 6$ robots and $N = 3$ waypoints in robust mode. For an animation of the results check: `https://youtu.be/ZgjFBtyJZeY`

Fig. 4: Trajectories generated for the reach and avoid case study Section V-B in robust mode. Here G is the set to visit and O to is the region avoid.

$$\varphi_{\text{reach avoid}} = \bigwedge_{\forall r \in \mathcal{R}} \Diamond_{[0,T]}(p_r \in G) \wedge \bigwedge_{\forall r \in \mathcal{R}} \Box_{[0,T]}(p_r \notin O) \wedge \bigwedge_{\substack{\forall r_1, r_2 \in \mathcal{R}, \\ r_1 \neq r_2}} \Box_{[0,T]}(||p_{r_1} - p_{r_2}|| \geq d) \tag{10}$$

Simulation Results. For each value of $\mathcal{R} \in \{1, 2, 4, 6\}$, we generate 100 feasible initial positions, and generate trajectories by solving (9) to find a feasible solution (Table I) and robust solution (Table II) for $\varphi_{\text{reach avoid}}$.

Discussion. We observe that the proposed method finds a satisfying solution faster than the baseline approach in boolean mode. And, also generates trajectories with higher robustness value on an average in the robustness mode. Example plots for the robust mode solutions have been presented in Figure 4.

C. Case Study 2: Narrow Corridors

Building on the reach and avoid example, we now provide a complex environment with narrow corridors for the algorithms to solve. The map has multiple obstacles enclosing the workspace; we denote the set of obstacles as O. The goal region is $G = [12, 14] \times [6, 8]$. The specification remains the same as that of reach and avoid (10), except now there are five obstacles to avoid (see Figure 5).

R	Runtime(s)	
	Baseline	Proposed
1	0.21 ± 0.001	$\mathbf{0.03 \pm 0.003}$
2	0.44 ± 0.13	$\mathbf{0.14 \pm 0.09}$
4	2.0 ± 1.13	$\mathbf{0.34 \pm 0.19}$
6	3.3 ± 1.72	$\mathbf{1.06 \pm 0.32}$

TABLE I: Reach and avoid case study in boolean mode with $N = 3$ waypoints. Both methods terminate successfully in all 100 simulations, and the proposed method computation time is *significantly faster* than the baseline.

R	Robustness	
	Baseline	Proposed
1	1.99 ± 0.00	1.99 ± 0.00
2	1.66 ± 0.38	1.85 ± 0.20
4	1.21 ± 0.23	1.27 ± 0.16
6	0.99 ± 0.15	1.04 ± 0.25

TABLE II: Reach and avoid case study in robust mode with $N = 3$ waypoints. Proposed method generates trajectories with higher robustness on average, than those generated by the baseline.

Simulation Results. Similar to the reach and avoid experiment, We generate generate trajectories by solving (9). The boolean mode results have been presented in Table IIIa and robust mode in Table IIIb.

Discussion. The baseline algorithm times out across all 100 simulations, i.e, does not converge within the maximum iterations allowed. This is most likely due to the number of obstacles increasing computational complexity. The proposed approach satisfies the specification in all 100 simulations. See Figure 5 for the results.

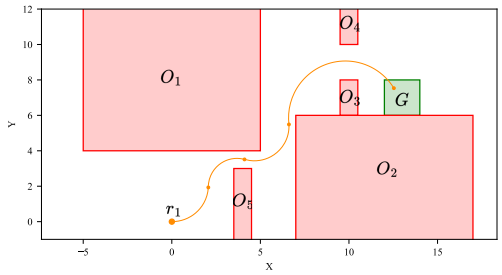
D. Case Study 3: Open Says Me

Building on the narrow corridors environment, we provide another map with collaboration required between two robots to solve it (See Figure 6). Robot r_1 is tasked to visit G_1, but r_1 is not allowed to enter G_2 until r_2 visits G_3. The robots are tasked to carry this out in $T = 40$ seconds, and r_2 is to reach G_3 by $T_1 = 0.75T$, to allow r_1 enough time to reach G_1. Additionally, the robots have to avoid each other by at least d distance and all other obstacles in the obstacle set $O = \{O_1, O_2, O_3, O_4\}$. The specification is provided as

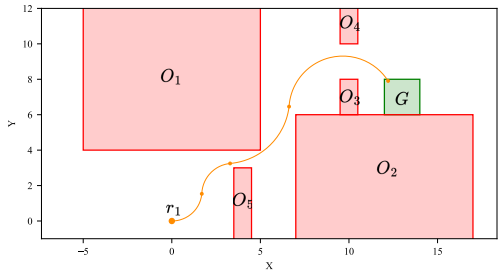
$$\varphi_{puzzle} = \Diamond_{[0,T]}(p_{r_1} \in G_1) \wedge (p_{r_2} \in G_3)\mathcal{U}_{[0,T_1]}(p_{r_1} \notin G_2) \wedge \bigwedge_{\substack{r \in \mathcal{R}, \\ o \in O}} \Box_{[0,T]}(p_r \notin o) \wedge \Box_{[0,T]}(||p_{r_1} - p_{r_2}|| > d) \tag{11}$$

Simulation Results. Similar to previous experiment, We generate 100 feasible initial positions for both the robots and generate trajectories by solving (9). The boolean mode results have been presented in Table IIIa and robust mode in Table IIIb.

Discussion. The baseline times out and does not generates satisfying trajectories. The proposed approach successfully



(a) Robust mode solution



(b) Boolean mode solution

Fig. 5: These figures present the trajectories generated for the narrow corridors case study (Section V-C) in both boolean and robust modes with $N = 5$ waypoints. The specification is reach avoid given by (10), the robot is supposed to take multiple tight corners to reach the final goal G. The baseline times out and does not generate satisfying trajectories.

Experiment	Runtime(s)
Narrow Corridors	1.39 ± 0.32
Open says me	0.83 ± 0.66

(a) Boolean mode results.

Experiment	Robustness
Narrow Corridors	0.34 ± 0.12
Open says me	0.49 ± 0.00

(b) Robust mode results.

TABLE III: These tables present the metrics for narrow corridors (Section V-C) and open says me (Section V-D) case studies. The baseline algorithm times out in both studies.

generates satisfying trajectories in all 100 simulations. See Figure 6 for the results.

E. Case Study 4: Need For Speed

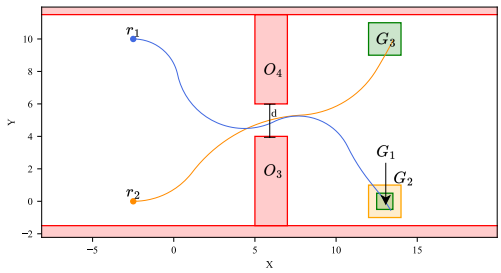
As presented in Example 1, we benchmark an overtaking scenario.

Simulation Results. The results have been taken with 100 different distances between car A and B. The boolean mode results have been presented in Table IVa and robust mode in Table IVb.

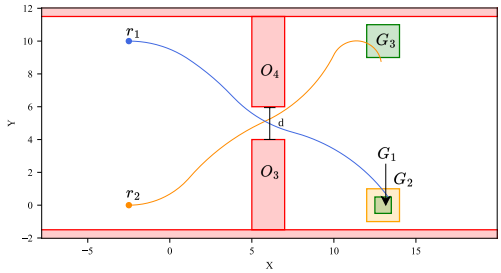
Discussion. In robust mode, the baseline and proposed method achieve similar robustness on an average of 100 simulations. In boolean mode, the proposed method is *significantly faster* than the baseline in generating valid trajectories.

F. Case Study 5: demonstration on a robot

We demonstrate our method on a real AgileX Limo robot 8, tasked with satisfying a reach-avoid specification



(a) Robust mode solution. To view an animation, visit here: `https://youtu.be/1TnH-z0Vojw`



(b) Boolean mode solution. To view an animation, visit here: `https://youtu.be/4GwpWS63VIw`

Fig. 6: These figures present the trajectories generated for the open says case study (Section V-D) in both boolean and robust modes with $N = 6$ waypoints. The specification is given by (11). r_1 is unable to visit G_1, unless G_2 receives an "open says me" from r_2 visiting G_3. The baseline times out during this experiment.

Algorithm	Runtime(s)
Baseline	0.48 ± 0.03
Proposed	$\mathbf{0.16 \pm 0.03}$

(a) Boolean mode results.

Algorithm	Robustness
Baseline	0.48 ± 0.003
Proposed	0.49 ± 0.002

(b) Robust mode results.

TABLE IV: These tables present a comparison between baseline and proposed method in boolean and robust modes for the need for speed case study Section V-E

Section V-B; We show that the robot, equipped with an off-the-shelf tracking controller, is able to follow trajectories generated by our method, satisfying the STL specification.

VI. Conclusion

Summary. We developed a non-convex optimization-based method, *drive-by-logic*, for generating kinematically feasible trajectories for car-like robots to satisfy Signal Temporal Logic specifications. To the best of our knowledge, it is the first method that generates trajectories, which are state solutions to the kinematic bicycle model and can handle specifications involving the full grammar of STL. Extensive simulations demonstrate that our approach is computationally faster than a Model Predictive Control baseline and can satisfy specifications that the baseline cannot.

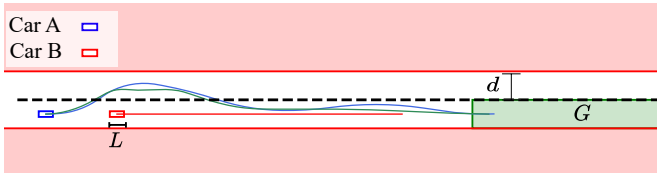


Fig. 7: This figure shows the trajectories generated in robust mode with $N = 10$ waypoints, comparing the baseline (green) and proposed (blue) trajectories. Here, the robustness achieved for both the trajectories is the same. An animation viewed here: To view an animation, visit here: `https://youtu.be/1953Gh-BBSI`

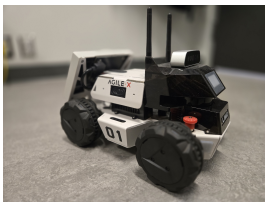


Fig. 8: An image of the AgileX Limo robot used in our studies. An experimental run is shown here: `https://ece.uwaterloo.ca/~sl2smith/CCTA2025-Robot-Tracking.mp4`.

Limitations. As discussed in section IV-B, the proposed method is not a *complete* algorithm in that it is not guaranteed to find a feasible solution to (9), even when one exists. The generated trajectory is also only C^1 - continuous.

Future Work. To address the latter limitation, we will extend the method to account for angular acceleration [32] and the sharpness of the curves that make up the entire trajectory of the robot. We will also extend the methodology to motion planning for other systems, such as an unmanned fixed-wing aircraft [33].

REFERENCES

[1] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE access*, vol. 8, pp. 58 443–58 469, 2020.

[2] Y. Wang, D. Zhang, Q. Liu, F. Shen, and L. H. Lee, "Towards enhancing the last-mile delivery: An effective crowd-tasking model with scalable solutions," *Transportation Research Part E: Logistics and Transportation Review*, vol. 93, pp. 279–293, 2016.

[3] J. López, D. Pérez, E. Paz, and A. Santana, "Watchbot: A building maintenance and surveillance system based on autonomous robots," *Robotics and Autonomous Systems*, 2013.

[4] A. Donzé and O. Maler, "Robust Satisfaction of Temporal Logic over Real-Valued Signals," in *Formal Modeling and Analysis of Timed Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[5] V. Raman, M. Maasoumy, and A. Donzé, "Model predictive control from signal temporal logic specifications: a case study," in *Proceedings of the ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems*. Association for Computing Machinery, 2014.

[6] Y. V. Pant, H. Abbas, R. A. Quaye, and R. Mangharam, "Fly-by-Logic: Control of Multi-Drone Fleets with Temporal Logic Objectives," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018.

[7] S. Sadraddini and C. Belta, "Robust temporal logic model predictive control," in *Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE Press, 2015.

[8] D. Sun, J. Chen, S. Mitra, and C. Fan, "Multi-agent Motion Planning from Signal Temporal Logic Specifications," 2022. [Online]. Available: http://arxiv.org/abs/2201.05247

[9] Y. V. Pant, H. Abbas, and R. Mangharam, "Smooth operator: Control using the smooth robustness of temporal logic," in *IEEE Conference on Control Technology and Applications (CCTA)*, 2017.

[10] Y. Yuan, T. Quartz, and J. Liu, "Signal temporal logic planning with time-varying robustness," *IEEE Control Systems Letters*, 2024.

[11] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, 1999.

[12] S. Saha and A. A. Julius, "An MILP approach for real-time optimal controller synthesis with metric temporal logic specifications," in *American Control Conference (ACC)*. IEEE, 2016.

[13] L. Lindemann and D. V. Dimarogonas, "Control Barrier Functions for Signal Temporal Logic Tasks," *IEEE Control Systems Letters*, 2019.

[14] A. Wiltz and D. V. Dimarogonas, "Handling disjunctions in signal temporal logic based control through nonsmooth barrier functions," in *Conference on Decision and Control (CDC)*, 2022.

[15] G. Yang, C. Belta, and R. Tron, "Continuous-time signal temporal logic planning with control barrier functions," in *2020 American Control Conference (ACC)*. IEEE, 2020, pp. 4612–4618.

[16] Y. Gilpin, V. Kurtz, and H. Lin, "A smooth robustness measure of signal temporal logic for symbolic control," *IEEE Control Systems Letters*, vol. 5, no. 1, pp. 241–246, 2021.

[17] J. Karlsson, F. S. Barbosa, and J. Tumova, "Sampling-based motion planning with temporal logic missions and spatial preferences," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 15 537–15 543, 2020.

[18] Y. Kantaros and M. M. Zavlanos, "Stylus*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems," *The International Journal of Robotics Research*, 2020.

[19] Y. Meng and C. Fan, "Signal temporal logic neural predictive control," 2023. [Online]. Available: https://arxiv.org/abs/2309.05131

[20] X. Li, C.-I. Vasile, and C. Belta, "Reinforcement learning with temporal logic rewards," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[21] A. Balakrishnan and J. V. Deshmukh, "Structured reward shaping using signal temporal logic specifications," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[22] Y. V. Pant, H. Yin, M. Arcak, and S. A. Seshia, "Co-design of control and planning for multi-rotor uavs with signal temporal logic specifications," in *American Control Conference (ACC)*. IEEE, 2021.

[23] J. Verhagen, L. Lindemann, and J. Tumova, "Temporally robust multi-agent stl motion planning in continuous time," in *2024 American Control Conference (ACC)*. IEEE, 2024, pp. 251–258.

[24] P. Bevilacqua, M. Frego, D. Fontanelli, and L. Palopoli, "A novel formalisation of the Markov-Dubins problem," in *European Control Conference (ECC)*, 2020.

[25] M. Althoff, M. Koschi, and S. Manzinger, "Commonroad: Composable benchmarks for motion planning on roads," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 719–726.

[26] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, Y. Lakhnech and S. Yovine, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 152–166.

[27] E. Bertolazzi and M. Frego, "A note on robust biarc computation," *arXiv preprint arXiv:1711.00935*, 2017.

[28] N. Mehdipour, C.-I. Vasile, and C. Belta, "Arithmetic-geometric mean robustness for control from signal temporal logic specifications," in *American Control Conference (ACC)*. IEEE, 2019.

[29] A. Robotics, "LIMO PRO — global.agilex.ai," https://global.agilex.ai/products/limo-pro.

[30] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[31] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, 2006.

[32] H. Sussmann, "The Markov-Dubins problem with angular acceleration control," in *Proceedings of the IEEE Conference on Decision and Control*, 1997.

[33] T. McLain, R. W. Beard, and M. Owen, "Implementing dubins airplane paths on fixed-wing uavs," *Contributed chapter to the Handbook of Unmanned Aerial Vehicles, Springer*, 2014.