2022-XX-XXXX

# Safety Verification and Navigation for Autonomous Vehicles based on Signal Temporal Logic Constraints

**Aditya Parameshwaran and Yue Wang**
Department of Mechanical Engineering, Clemson University

## Abstract

The software architecture behind modern autonomous vehicles (AV) is becoming more complex steadily. Safety verification is now an imminent task prior to the large-scale deployment of such convoluted models. For safety-critical tasks in navigation, it becomes imperative to perform a verification procedure on the trajectories proposed by the planning algorithm prior to deployment. Signal Temporal Logic (STL) constraints can dictate the safety requirements for an AV. A combination of STL constraints is called a specification. A key difference between STL and other logic constraints is that STL allows us to work on continuous signals. We verify the satisfaction of the STL specifications by calculating the robustness value for each signal within the specification. Higher robustness values indicate a safer system. Model Predictive Control (MPC) is one of the most widely used methods to control the navigation of an AV, with an underlying set of state and input constraints. Our research aims to formulate and test an MPC controller, with STL specifications as constraints, that can safely navigate an AV. The primary goal of the cost function is to minimize the control inputs. STL constraints will act as an additional layer of constraints that would change based on the scenario and task on hand. We propose using sTaliro, a MATLAB-based robustness calculator for STL specifications, formulated in a receding horizon control fashion for an AV navigation task. It inputs a simplified AV state space model and a set of STL specifications, for which it constructs a closed-loop controller. We test out our controller for different test cases/scenarios and verify the safe navigation of our AV model.

## Introduction

Safety verification techniques for autonomous vehicles (AV) have seen an unprecedented rise in demand over the past few years. AVs are also becoming widely popular for off-road applications, particularly for military ground vehicles. Defining sets of rules to follow for such complex systems is an important task in maintaining the safety of the vehicle in the case of uncertainties. Temporal logic offers a formal method to help designers transform natural language rules for safety into the mathematical formulation. Over a couple of years, temporal operators have been used to express safety requirements and reach-avoid sets with the ability to time and sequence them. Some of the operators like Linear Temporal Logic (LTL) [1], [3], Metric Temporal Logic (MTL) [4], [5], and Signal Temporal Logic (STL) [6], [7] are used to verify and synthesize controllers for motion planning of dynamic autonomous systems that satisfy the predefined specifications. Synthesis of controllers based on LTL and MTL focus on the automata-based representation of the specification and represent the workspace as a transition system [10]. However, this leads to a highly complex transition system model that is based on discrete states that are expensive to compute and can lead to state space explosion [8]. Compared to LTL and MTL, a major advantage of STL specifications is that they provide temporal logic operators that are compatible with working with real-time continuous signals over defined periods of time.

Distribution A. Approved for public release; distribution unlimited. (OPSEC 7039)

To address concerns regarding exceeding computational time and uncertainties in the modeling, T. Wongpiromsarn et al. [11]. have proposed using receding horizon control for temporal logic constraints problems. Wolff et al. [2] have converted the LTL and MTL specifications into mixed integer constraints to be applied recursively over a fixed horizon. Previous work has shown the effectiveness of real-time monitoring and control of STL specifications-based models for AV examples [15]. Similar applications have been more recently applied to STL specifications in the form of Model Predictive Control (MPC) problems with mixed integer constraints [13]. STL specifications assess the robustness of these time-varying parameters and provide a numerical value to verify and control specification satisfaction. As a result, robustness provides a validation measure for the STL specification even on complex multiple signals, and non-linear continuous-time systems such as AVs. Navigation tasks in AVs demand significant control over the model with assured constraints for maintaining safety. STL specifications can be converted to *linear inequality equations* that can be solved recursively for any AV navigation task in a known environment over a fixed horizon [9], [16].

In this paper, we discuss the possibility of developing safe trajectories for our 2D AV model pertaining to minimum robustness satisfaction of our STL specifications. The STL specifications are maintained as a string user input that is initially parsed by the solver to be converted to *Linear Inequality Equations* (LIE) that are solved by a Mixed Integer Linear Programmer (MILP) [14] and applied over a fixed horizon recursively in the MPC fashion. The MILP solver computes an optimal solution that minimizes a cost function based on the LIE. As MILP solves to search for a global solution, applying it for the entire state space can result in expensive computational time. Combining it with MPC has the advantage of predicting the states for only $H$ steps into the horizon and applying the first optimal control input to advance the system. The LIE are formed automatically for all sets of hard constraints such as STL specifications, system dynamics, and state and control limits of our simplified AV model. It is important to note all three listed sets of constraints are necessary conditions for the solution to be feasible. The cost function mentioned within the MILP solver provides the set of soft constraints that minimizes control input for optimally.

The rest of the paper is organized as follows. Section 3 sets up the preliminaries with an introduction to the simple car unicycle kinematic model, STL operators, grammar, and robustness. Section 4 defines the problem statement, where the MILP controller is formulated based on the set of constraints. Its application in a receding horizon-based MPC approach is also discussed. Section 5 demonstrates the simulations where the accuracy of the feedback linearized model is validated and then the MILP + MPC algorithm is set up in MATLAB and applied to solve different navigation tasks for known obstacles and goal regions. Section 6 concludes the paper and provides the scope of future work.

## Preliminaries

### *Simple AV Kinematic Model*

For the scope of our work, we consider a simplified 2D unicycle kinematic model for an AV where:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \tag{1}$$

The states, $[x, y, \theta]$, represent the 2D pose and orientation of the AV and $[\omega, v]$ are the vehicle's angular acceleration and linear velocity, respectively. One of the major issues most nonlinear models face is the calculation of the online solution for non-convex control problems. Suppose the state space to search an optimal solution for is relatively huge. In that case, as is possible in the case of motion planning tasks for autonomous vehicles, the solver can obtain local minimum solutions that could prove to be infeasible over the trajectory. MILP problems also have longer computational times, especially when combined with multiple LIE constraints. For a simplified model as shown in Eqn. [1], we can afford to linearise the system with minimal loss to accuracy. Also to avoid infeasible solutions and large computational times, we implement a feedback linearised discrete-time system model where:

$$\begin{aligned} \dot{x}_1 &= \dot{x} = v_x = x_3 \\ \dot{x}_2 &= \dot{y} = v_x = x_4 \\ \dot{x}_3 &= \dot{v}_x = a_x = u_1 \\ \dot{x}_4 &= \dot{v}_y = a_x = u_2 \end{aligned}$$

where the states $\mathbf{x}(t) \triangleq [x_1(t); x_2(t); x_3(t); x_4(t)] = [x(t); y(t); v_x(t); v_y(t)]$ are the global X-Y coordinates and X-Y linear velocities respectively, the control inputs are $u(t) = [u_1; u_2]$ such that $[u_1, u_2] = [acc_x, acc_y]$, i.e., the linear accelerations in X and Y coordinates, respectively. We also discretize the system with a time step $T_s = 0.1s$ as the MILP solver works only for discrete-time models. Rewriting the above equation in state space form we get:

$$\begin{aligned} \mathbf{x}(t+1) &= A\mathbf{x}(t) + Bu(t) \\ \mathbf{y}(t) &= C\mathbf{x}(t) \end{aligned} \tag{2}$$

where,

$$A = \begin{bmatrix} 0\,0\,1\,0 \\ 0\,0\,0\,1 \\ 0\,0\,0\,0 \\ 0\,0\,0\,0 \end{bmatrix};\ B = \begin{bmatrix} 0\,0 \\ 0\,0 \\ 1\,0 \\ 0\,1 \end{bmatrix};\ C = \begin{bmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \end{bmatrix};\ D = [0] \tag{3}$$

To compare the accuracy of the feedback linearised model with the nonlinear model (1), we convert our feedback linear state outputs, $[x, y, v_x, v_y]$ into linear velocity and angular velocity, calling them $[v_l, \omega_l]$. If we chose the position of the AV as the system output, we notice that:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{v_l} \\ v_l\omega_l \end{bmatrix}$$

We can then invert the matrices to calculate $[v_l, \omega_l]$ such that:

$$\begin{bmatrix} \dot{v_l} \\ v_l\omega_l \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix} \tag{4}$$

where $[u_x, u_y] = [u_1, u_2] = [acc_x, acc_y]$ are the inputs for the second order system in the above equation. These inputs are generated as the optimal solution to the MILP problem that is presented in Section 4. As a result, we now have the values for $[v_l, \omega_l]$.In Section 5, we compare the accuracy of our feedback linearised model velocities $[v_l, \omega_l]$ by applying them as inputs to a nonlinear kinematic model [1] for a simple point-to-point navigation task and compare the results with the trajectory computed by MATLAB's Nonlinear MPC block based on the kinematics in [1]. Note that within the operating domain of the states of our AV, we assume that there will always be a computed control sequence $U = [u_1, ...u_{N-1}]$ such that our system dynamics $\mathbf{x}(t+1) = A\mathbf{x}(t) + Bu(t)$ will always be satisfied. For a control sequence of $(N-1)$ states, we have $(N)$ states on the horizon. We also assume that we can observe all our system's states at all times.

### *Signal Temporal Logic (STL)*

STL has a library of operators representing the formal language's grammar and alphabet. The designers think and formulate rules in natural language that is then defined in a mathematical context using the STL operators. This allows for a more formal method to set constraints for signal-based systems.

**Definition 3.1** (STL Operators)**.** Writing in the Backus-Naur form, an STL formula can be expressed with the following syntax:

$$\phi ::= \top \mid \mu \mid \neg\phi \mid \phi \wedge \psi \mid \Box_{[a,b]}\phi \mid \Diamond_{[a,b]}\phi \mid \phi\,\mathcal{U}_{[a,b]}\,\psi$$

where $\top$ is the Boolean operator true, $\mu$ is a predicate of the form $f(s_t) > c$. $\phi$ and $\psi$ are STL specifications and $s_t$ is a signal with respect to which we check the specifications. The other temporal operators such as $\mathcal{U}$, $\Box$, and $\Diamond$ are the 'Until', 'Always', and 'Eventually' operators. For a given signal $s_t$ the list of temporal operators can be defined as follows:

$$\begin{array}{lcl} s_t \models \mu & \Leftrightarrow & f(s_t) > 0 \\ s_t \models \neg\phi & \Leftrightarrow & \neg(s_t \models \phi) \\ s_t \models \phi \wedge \psi & \Leftrightarrow & (s_t \models \phi) \wedge (s_t \models \psi) \\ s_t \models \phi \vee \psi & \Leftrightarrow & \neg(\neg(s_t \models \phi) \wedge \neg(s_t \models \psi)) \\ s_t \models \phi \Rightarrow \psi & \Leftrightarrow & \neg(s_t \models \phi) \vee (s_t \models \psi) \\ s_t \models \Diamond_{[a,b]}\phi & \Leftrightarrow & \exists t' \in [t+a, t+b] \text{ s.t. } s_{t'} \models \phi \\ s_t \models \Box_{[a,b]}\phi & \Leftrightarrow & \forall t' \in [t+a, t+b] \text{ s.t. } s_{t'} \models \phi \\ s_t \models \phi\mathcal{U}_{[a,b]}\psi & \Leftrightarrow & \exists t' \in [t+a, t+b] \text{ s.t. } (s_{t'} \models \psi) \\ & & \wedge \left(s_t \models \Box_{[0,t']}\phi\right) \end{array} \tag{5}$$

The term $\phi\mathcal{U}_{[a,b]}\psi$ means that 'eventually' in time $t' = [a, b]$ $\psi$ will be true only if from time $(0, t')$ specification $\phi$ is 'always' true. The 'eventually' and 'always' operators mean that the specifications must be true at least once and always in the specified time range. The operators can always be defined using literals in their *Negation Normal Form*. They are not restrictive as they can be applied to any of the above STL operators as shown by S. M. LaValle et al. [18]. The disjunction operator $\vee$ is an example of *Negation Normal Form* using a combination of other literals. Specifications can be combined to develop constraints for autonomous vehicle models that require multiple parameters to be controlled. For every specification $\phi_i$ built on a combination of multiple state and control parameters, a single predicate value $\mu_i$ is allocated to be computed. The signal $s_t$ can be any real-time state value $\mathbf{x}_t$ such as linear velocity $(v)$, position coordinates $[x, y]$, etc.

**Example 1.** Consider an AV where its 2 output states are the *position coordinates* $[x, y] = [x_1; x_2]$. Now imagine an STL specification that calls for $(x_1 \geq 1)$ to "always" be true for all given times. We can identify the output $y = x_1 - 1$ as the predicate $\mu_1$, such that for $(\mu_1 \geq 0) \; \forall \; t \; \epsilon \; [0, ...\infty]$. The STL specification would be:

$$\phi = \Box_{[0,\infty]}(y(t) \geq 0) \tag{6}$$

To justify the complete satisfaction of the above specification for all given times, we imply the method of quantifying the specification value with respect to the predicate being measured as a form of the robustness of the signal.

**Definition 3.2** (Robustness Semantics for STL Specifications)**.** To quantitatively measure the satisfaction of an STL specification, robust semantics are used to define real-valued numbered functions $\rho^{\phi}$. The robustness measure for any specification $\phi$ within a system corresponding to a signal $s$ over time $t$ can be defined as:

$$s_t \models \phi \leftrightarrow \rho^{\phi}(s_t) = \top \tag{7}$$

where the $\rho^{\phi}$ is defined as the robustness measure for that specification at time instant $t$, the value of the robustness measure can vary over time at every instant during the interval $t \in [a, b]$ where the specification $\phi_{[a,b]}$ holds true. The idea of the robustness measure is to provide an optimization and control parameter that can quantify the satisfaction of the specification. Positive robustness value $(\rho^{\phi} > 0)$ guarantees the satisfaction of the specification over the time period. The robust satisfaction of $\rho(\phi \wedge \psi)$ is the $min(\rho^{\phi}, \rho^{\psi})$. The robustness for specifications such as the "always" operator i.e. $\phi = \Box \; s_{[a,b]}$ is $\rho^{\phi} = min_{t \in [a,b]} \; (\rho^{\mu}(s_t))$. Variation in the predicate value $\mu$ or in the STL specification itself would change the robustness value but vice versa is not true. Shown below is a list of the entire domain of robust semantics:

$$\begin{aligned}
\rho^{\mu}(s_t) \quad &= f(s_t) \\
\rho \neg \mu(s_t) \quad &= -\mu(s_t) \\
\rho^{\phi \wedge \psi}(s_t) \quad &= \min\left(\rho^{\phi}(s_t), \rho^{\psi}(s_t)\right) \\
\rho^{\phi \vee \psi}(s_t) \quad &= \max\left(\rho^{\varphi}(s_t), \rho^{\psi}(s_t)\right) \\
\rho^{\Box_{[a,b]}\phi}(s_t) \quad &= \min_{t' \in [t+a, t+b]} \rho^{\phi}\left(s_{t'}\right) \\
\rho^{\phi}\mathcal{U}_{[a,b]\psi}(s_t) \quad &= \max_{t' \in [t+a, t+b]} \left(\min\left(\rho^{\psi}\left(s_{t'}\right), \min_{t'' \in [t, t']} \rho^{\phi}\left(s_{t''}\right)\right)\right)
\end{aligned}$$

**Example 2.** (Specification Example for Navigation Task):
An example of STL specification for an AV to avoid an obstacle with known bounds $(P_{obstacle})$ and end at a goal location with known bounds $(P_{goal})$ within the time frame $[a, b]$ would look as follows:

$$\phi = \Box \neg P_{obstacle} \wedge \diamond_{[a,b]} P_{goal} \tag{8}$$

Here, $P_{obstacle}$ and $P_{goal}$ are convex polytopes developed based on the X-Y coordinates provided by the user as inputs in the STL specification. For the navigation task, multiple STL formulae would be combined to solve for an optimal solution such that $\phi' = (\phi_1 \; \wedge \phi_2 \; \wedge ..., \phi_n)$. We use an STL parser to decode the string-based specification into LIE that are then encoded as the bounds for the obstacle regions and goal regions into the MILP problem. The MILP problem is then solved recursively for a fixed prediction horizon $(H)$ using a receding horizon MPC.

## Problem Statement

The problem at hand is to parse the STL specifications set for the system and bound them as hard constraints to an MPC-based controller that minimizes a cost function. In general, the signals $(s_t)$ for STL specifications can be any of the states $(\mathbf{x})$, control inputs $(u)$ or the state outputs $(y)$. However, it is to be noted that for the problem at hand, we work with specifications that demand control only of the system states $(\mathbf{x})$, such that $s_t \in \mathbf{x}(t)$. Moving on, we shall replace the signals $s_t$ with the state values $\mathbf{x}(t)$ for easier understanding of the problem formulation. The MPC controller solves for an optimal control sequence $u_i = [u_1, ...u_H]$ within the receding horizon of $H$ steps while satisfying the hard constraints set by STL, system dynamics and state constraints.

**Problem 1.** (Cost Function Synthesis from STL):
For an initial value of $x_0$, prediction horizon of $H$ steps going from time $t_0, t_1, ...t_H$, cost function $J$ and STL formula $\phi$, we calculate for the optimal control at each time step as:

$$\begin{aligned}
& min_u^{H,t} J(\mathbf{x}(t), u^{H,t}) \\
& \text{ s.t. } \; \rho^{\phi}(\mathbf{x}(t)) \geq \rho^{\phi}_{min} \\
& \mathbf{x}(t+1) = A\mathbf{x}(t) + Bu(t) \\
& u_t \in \mathcal{U}(t) \; , \; \mathbf{x} \in \mathcal{X}(t)
\end{aligned} \tag{9}$$

where $u^{H,t}$ is the control input to be calculated for each time step $t$ as defined by V. Raman et al. [15]. The robustness measure $\rho^{\phi}(\mathbf{x}(t))$ for the specification $\phi$ based on state values $\mathbf{x}$ at time $t$ should be greater than a minimum robustness value of $\rho_{min}$
**Problem 2.** (Encoding STL specifications into MPC):
The task of the controller is to compute for an optimal solution such that the STL specif,icatconstraints $\phi$ maintains a positive robustness value $\rho^{\phi}(\mathbf{x}(t)) > 0$ for the state values $\mathbf{x}$ within the horizon length $H$. The specification for the horizon can be computed repeatedly as shown by A. Dokhanchi et al. [17]:

$$\begin{aligned}
& h^{\mu} = 0, \\
& h^{\neg\varphi} = h^{\varphi}, \\
& h^{\vartheta_{[a,b]}\varphi} = h^{\Box_{[a,b]}\varphi} = b + h^{\varphi}, \\
& h^{\varphi \wedge \psi} = h^{\varphi \vee \psi} = \max\left(h^{\varphi}, h^{\psi}\right), \\
& h^{\varphi}\mathcal{U}_{[a,b]}\psi = b + \max\left(h^{\varphi}, h^{\psi}\right),
\end{aligned} \tag{10}$$

where $\mu$ is the numerical predicate with $\phi, \psi$ being the STL specifications. For the predicate horizon length of $h \leq H$, we can calculate the robustness value $\rho^{\phi}(t) \forall \; t \; \in \; (0, ...h)$ for the signal $\mathbf{x} \in (\mathbf{x}_t, \mathbf{x}_{t+1}, \cdots, \mathbf{x}_{t+h})$. This is possible because we predict state up to $H$ steps but only calculate the robustness values of the states up to $h$ steps. We know the current robustness value at time step $t$ at our current state $\mathbf{x}(t)$ and also know the history of all the values leading to the current state. Based on the evolution of our system, we try to maintain a positive robustness for all future predicted states from $\mathbf{x}_t, \cdots, \mathbf{x}_{t+h}$. As we are working with an MPC controller, the most distant robustness value we can calculate would be based on the prediction horizon $H$ for the controller, i.e., $\rho^{\phi}(t + H)$. Hence the set of constraints that we set range from the previous known horizon length $(t - h)$ till the future predicted state at prediction horizon $t + H$. This means that we try to find an optimal solution that would satisfy the positive robustness measure for all the steps within the prediction horizon as described by Sadraddini et. al [19]:

$$\begin{cases}
\rho(s_t)^{\phi}\,[t-h] & \geq 0 \\
\rho(s_t)^{\phi}\,[t-h+1] & \geq 0 \\
\quad \vdots & \\
\rho(s_t)^{\phi}\,[t+H] & \geq 0
\end{cases} \tag{11}$$

The calculated control sequence would be $u^{H,t}(t) = [u(t), u(t+1), \cdots, u(t+h+H)]$. Only the first control input $u_t$, i.e., at the first step, is applied to control the system. The prediction process starts again, along with calculating new robustness values based on the new current state for $t = [t+1, ...t+H+1]$.
The main aim of our controller is to compute correct and optimal solutions. Even though optimal solutions are ideally sought after,

suboptimal solutions that satisfy the constraints are also accepted. The set of hard constraints we define for our problem includes the state and control $[\mathcal{X}, \mathcal{U}]$ limits, the system dynamics for our AV model ($\mathbf{x}(t+1) = A\mathbf{x} + Bu$) and the set of STL specifications $\phi'$.

## *Mixed Integer Linear Programming*

In this section, we formulate the MILP problem based on the state, control and STL constraints for our system. A general MILP based for our system would be in the form of:

$$\begin{aligned} &\min_{u(t)} \sum_{k=0}^{H-1} \|u(t+k)\| \\ &\mathbf{Gx}(t) \geq \mathbf{l} \\ &\text{s.t. } \rho^{\phi}(\mathbf{x}(t)) \geq \rho^{\phi}_{min}, \\ &\mathbf{x}(t+1) = A\mathbf{x}(t) + Bu(t), \\ &u(t) \in \mathcal{U}(t) \ , \ \mathbf{x}(t) \in \mathcal{X}(t) \end{aligned} \tag{12}$$

where $\min_{u(t)} \sum_{k=0}^{H-1} ||u(t+k)||$ denotes the cost function to be minimised subject to *Linear Inequality Equations* (LIE) $(\mathbf{Gx}(t) \geq \mathbf{l})$ that are formulated using system dynamics $\mathbf{x}(t+1) = A\mathbf{x}(t) + Bu(t)$, control and state constraints $u(t) \in \mathcal{U}(t)$ , $\mathbf{x}(t) \in \mathcal{X}(t)$ and the robustness value for the trajectory $\rho^{\phi}(\mathbf{x}(t)) \geq \rho^{\phi}_{min}$. The optimal solution for the problem would be the one that satisfies all these constraints and minimizes the cost function. As safety is a key feature for our model, we ensure that these constraints act as a sufficient and necessary condition to compute the optimal solution. The aim of the solver is to calculate a feasible trajectory $(p)$ for every prediction horizon $H$ as discussed by R. Koymans et al. [4]. The robustness measure $\rho^{\phi}(p)$ of the calculated trajectory determines whether the trajectory sustains the constraints or voids it. The set of STL constraints are converted in LIE as discussed later in this section. Although satisfaction of these constraints depends upon the robustness of the specification $\phi'(t)$ at any time $t$, the safety of the vehicle depends only upon the specifications related to the unsafe regions i.e. $\rho^{\phi_{unsafe}}(t) \geq 0$ at all times.

Using the toolbox provided by [7], which is based on `sTaLiRo`, we calculate the critical robustness, time and predicate for the infeasible trajectory. As we already know the bounds for the obstacle region $P_{obstacle}$ and the state space bounds, we can represent the points in the safe region as a polytope $P_{safe} = (P_{state} - P_{obstacle})$. Any such set of polyhedrons, with $f$ faces can be represented by the below shown **LIE**:

$$P = \{\mathbf{x} \,|G\mathbf{x} \leq l, \ G \in \mathbb{R}^{fxn}, \ l \in \mathbb{R}^f\} \tag{13}$$

If the STL specification $\phi$ needs to be true at any time $k$, i.e. the trajectory $p^{\phi}$ needs to be in the set $P(k)$, then we represent the LIE as:

$$G\text{x(k)} \leq l$$

To counter this, if we require the STL specification to not be true at the time $k$, then we make use of a binary encoding method, where for any specification $\phi$, a binary variable $z^{\phi}(k) \in \{0,1\}$ indicates whether $\phi$ stands true or false at the time $k$. For $z^{\phi}(k) = 1$, the specification stands true and for $z^{\phi}(k) = 0$, the specification stands false. We also use a very large number $M$ such that the overall LIE look like:

$$G\mathbf{x}(k) + Mz^{\phi}(k) \geq l, \ z \ \in \ \{0,1\}^f \tag{14}$$

For instance,if the calculated trajectory is as required (i.e. $\rho(p,k) \ \geq \ \rho_{min}$) and the predicate $\mu(k)$ is within the $P_{safe}$ region, then $z = 1$ and the LIE constraints are simply to take the AV to the goal region as $M$ is sufficiently large to always the LIE holds true for any $\mathbf{x}(k)$. But if $\rho(p,k) \ \geq \ \rho_{min}$ at time $k$, then $z = 0$ and the obstacle region LIE are also enabled, thus forcing the MILP solver to calculate a new safe trajectory $p$ from $t = [k, ....k+H]$. This method helps reduce computational time as we only enable the unsafe region constraints when our predicted states have a robustness value lesser than the minimum required. This is still a safe controller as any positive robustness value ensures that the AV does not collide with the obstacle. Generally as shown in example $\{4\}$, we typically have multiple sets of STL specifications combined to form $\phi'$ usually separated by the $\wedge$ operator indicating that all of them have to be satisfied at all given times (or the time range $[t_a, t_b]$ in which they are specified). The STL parser developed separates the specifications and parses them individually to compute the polytope (G,b). For instance, for the AV navigation task, the typical STL list would look like $\phi' = (\phi_{goal} \ \wedge \ \neg\phi_{obstacle})$ which would be separated into sets of LIE.

**Definition 4.1.** (Combining Multiple Linear Inequality Equations) Let us call an LIE $(G, l)$, defining a single polytopic region $(P)$ in space, as $L$ where $L = l - G\mathbf{x}$. To maintain a minimum robustness of $\rho^{\phi}_{min}$, we modify the LIE such that

$$\begin{aligned} L^{\phi} &= b \pm \rho^{\phi}_{min} * 1_f - G\mathbf{x}, \\ L^{\phi_{obstacle}} &= -b \pm \rho^{\phi_{obstacle}}_{min} * 1_f + G\mathbf{x} + Mz^{\phi} \end{aligned} \tag{15}$$

where $1_f$ is a column vector of ones of the size of $f$. We can stack multiple such modified LIE's $L$ for each obstacle or goal region within the workspace such that $L_{goals} = [L_{goal^1}, ...L_{goal^g}]$ for $(g)$ goal regions and $L_{obstacles} = [L_{obstacle^1}, ...L_{obstacle^o}]$ for $(o)$ obstacle regions. For any trajectory $p$ within the horizon $H$, the combined LIE set $\hat{L}$ would be:

$$\begin{aligned} \hat{L} = [&\alpha_{(g,k)} * L_{goal_i} \ \wedge \ L_{obstacle_j} \ \wedge \\ &L_{state} \ \wedge \ L_{control} \ \wedge \ L_{dynamics}] \end{aligned} \tag{16}$$

where $\alpha(g,k)$ is a column vector with binary parameters of $g$ rows that dictate which one of the goal LIE's would be active for all time $k = [0, 1, ...N]$. For instance, when the AV needs to go to $goal_1$ at time $k$, then $\alpha(1,k) = 1$, and $\alpha(2:g,k) = 0$ would be inactive. This creates a hierarchy within the waypoints defining an order in which the waypoints would be visited. As a result, this reduces the size of the overall set of constraints, reducing the MILP solver computational time.

## *STL-based Receding Model Predictive Control*

In this section, we discuss the MPC solution to the MILP problem discussed previously based on the list of constraints. As mentioned earlier, a trajectory $p(t_0) = p_0$ for the first time step $t_0$ is calculated by ignoring any $\phi_{obstacle}$ and only applying constraints based on the state, control, system dynamics and remaining STL for a prediction horizon $H$. If the initial state of the system $x_0$ is within the bounds, and the trajectory calculated is a feasible one, then the control input $u(t_1)$ is applied for the first time step, and the prediction process is started again for time $t = (1, ..H+1)$.

For any iteration in the future, if the predicate value $\mu(p_k) \leq \rho^{\phi}_{min}$ for the trajectory $(p_k)$, then the calculated trajectory is infeasible at time step $(k)$. This is identified by calculating the critical robustness value for the trajectory using `sTaLiRo`. If the trajectory is infeasible and within an obstacle region, then the binary variable is set as $z = 0$, and the STL constraints regarding the obstacle region are set as an additional set of constraints for the MILP problem to solve again for the time range $t = t+k, ...t+k+H$. This occurs recursively till the trajectory is a feasible one and only then do we apply the control input $u(k)$ to advance the system by one step. The cost function that we are minimising for our problem is:

$$\begin{aligned} &\min_{\vec{u}_t} \sum_{k=0}^{H-1} \|u_{t+k}\| \\ &\text{s.t. } \mathbf{x}(t+k+1) = A\mathbf{x}(t+k) + Bu(t+k) \\ &\mathbf{x}(t) \models \phi \text{ and } u(t+k) \in \mathcal{U}(t+k), \quad k = (0, \dots, H-1) \end{aligned} \tag{17}$$

where we are minimizing the norm of the control input $u_{k,..k+H}$ subject to a set of mixed-integer constraints, system dynamics with state, and control bounds. The control input finally applied to propagate the system is $u_k$. We first apply the state, control, and system dynamics constraints to calculate a feasible trajectory from the start point $P_{start}$ to the goal location $P_{goal}$. This means that even though the set of STL constraints $\phi'$ are not first applied during the MILP synthesis, we check the robustness value of the trajectory $\rho^{\phi'}$. If the initial trajectory has $\rho^{\phi'} \geq \rho_{min}$ for time range $(t = 0, ..., H)$, then the computed control values are applied in a receding horizon fashion. As discussed in equation (11), we calculate the robustness values for time period $(t-h, ..t+H)$, and if any $\rho^{\phi'}(t) < 0$ then we term it as an "infeasible trajectory".

## *Specification Parser for Navigation Tasks*

The STL specifications mentioned for each navigation task are in the form of string-based inputs that need to be decoded to extract useful information for our MILP formulation.

**Definition 4.2.** (STL Parser)
A custom string parser is developed specifically to understand the STL-specification inputs. As defined in the MILP section, the idea of the parser is to identify the 2D coordinates for the obstacle, safe, and goal regions from the STL specification. Those coordinates are then converted into convex polytopes that can be defined in 2D space with LIE. The LIE's then act as a set of hard constraints for the MILP to solve for an optimal solution. The constraints based on the specification followed by the "$alw_{[a,b]}$" operator are recursively applied at every time step throughout the time period $[a, b]$. For the "$ev_{[a,b]}$", the constraints are applied throughout the period $[a, b]$, but once they are satisfied they are discarded from the specification list. An example STL specification input looks as shown:

**Example 3.** (*Go-To-Goal*):

$$\text{“}alw(not(X(1:2,t) \in [x_{obstacle}, y_{obstacle}])\ and$$
$$ev_{[a,b]}(X(1:2,t) \in [x_{goal}, y_{goal}])\text{”}$$

where $x_{obstacle} \in [\mathbf{x}_1, \mathbf{x}_2, ..\mathbf{x}_r]$ and $y_{obstacle} \in [\mathbf{y}_1, \mathbf{y}_2, ..\mathbf{y}_r]$ are list of X-Y coordinates for the obstacle region, "$X(1:2,t)$" are the AV's X-Y position values at time $t$ and $[t+a, t+b]$ is the range of time, starting from $t = 0$, within which we need our AV to be at the goal location. The above example only deals with a single obstacle region. For multiple obstacle regions, where the STL specification lists looks $[P_{obstacle_1}, ...P_{obstacle_n}] \Rightarrow [\phi_1, ...\phi_n]$, the combined specification list for the avoiding the obstacle regions would be $\Box(\neg\phi_1\ \wedge\ \neg\phi_2 ...\wedge\ \neg\phi_n)$

**Example 4.** (*Multi-Waypoint Navigation*):

$$\text{“}alw(not(X(1:2,t) \in [x_{obstacle}, y_{obstacle}])\ and$$
$$ev_{[a_1,b_1]}(X(1:2,t) \in [x_{goal_1}, y_{goal_1}])\ and\ ...$$
$$ev_{[a_n,b_n]}(X(1:2,t) \in [x_{goal_n}, y_{goal_n}])\text{”}$$

where we consider the set of goal coordinates to be $[(x_{goal_1}, y_{goal_1}), ...(x_{goal_n}, y_{goal_n})]$. The time stamps related to the entry and exit times for each of the way-points are also set as $[a_i, b_i] \Rightarrow \phi_{goal_i} \forall i = 1, ..n$. It is important to note that the time stamps are required to be generally in the range in which we expect the AV to reach the waypoint. Mentioning coinciding time ranges would lead to infeasible solutions. The solver will return an infeasible solution if the AV cannot reach the waypoint within the time range provided. Hence it is best to identify large ranges are inputs that can guarantee convergence.

## Simulations

To test the validity of our model, we carry out simulations on MATLAB 2022B using `Gurobi` solver through `YALMIP` for MILP formulation on a *i9 8th gen* processor with *2.9GHz* clock speed and *16GB* of RAM memory. The model is initialised for a sampling period $T_s = 0.1s$, with initial states $\mathbf{x}_0 = [-0.6, -0.4, 0, 0]$ for *Go-To-Goal* and $\mathbf{x}_0 = [0, -0.4, 0, 0]$ for *Multi-Waypoint* with a maximum run-time simulation period of 100 seconds. If the AV reaches its goal target within the simulation period, then the simulation ends.

## *Linear vs Nonlinear Model Comparison*

As we are using a feedback linearised model with the state and input matrices as shown in [3], instead of the non-linear AV system described in [1], we will first compare the results between the two systems to test the accuracy of the linear discrete-time model. For the non-linear system, we use the "*Nonlinear MPC*" (NMPC) block provided in MATLAB with the same simulation time-step $T_s = 0.1s$, the same as used to discretize the linear system as well. The cost function to minimize for the NMPC model is:

$$\begin{aligned} &\min_{\vec{u}} \sum_{0}^{H-1} \left\| \mathbf{x}^{ref} - \mathbf{x} \right\| + \|u\| \\ &\text{s.t.}\ \ \dot{\mathbf{x}} = f(\mathbf{x}(t), u(t)) \\ &u \in \mathcal{U}, \qquad k = 0, \ldots, H-1 \end{aligned} \tag{18}$$

where, $\dot{\mathbf{x}} = f(\mathbf{x}(t), u(t))$ is the system dynamics provided in [1] for NMPC and in [2] for Linear MPC. A reference trajectory ($\mathrm{x}^{\mathrm{ref}}$) is provided to both models, that is, a set of points between the start and goal locations. The cost function formulation differs from the one we used for the MILP + MPC model, as the main task of that system was to compute and traverse a safe trajectory. It is important to note that we are comparing the two systems to verify the accuracy of our feedback linearized state space model. Once we simulate the system, the states of our non-linear model are $[\mathbf{x}, \mathbf{y}, \theta]$ that can be easily plotted. The simplified AV kinematic model requires $[v, \omega]$ i.e., linear velocity and angular velocity as inputs. Using the linear and angular velocities developed from the linear feedback model [4], we simulate the non-linear simplified kinematics AV model described in (1) and verify the accuracy of our linearisation. By observing (1), we note that the trajectories for discrete-time feedback linearized and the continuous time non-linear models have a small deviation through the 6 waypoints which shows a good approximation. The green regions indicate acceptable waypoints for the AV to reach.

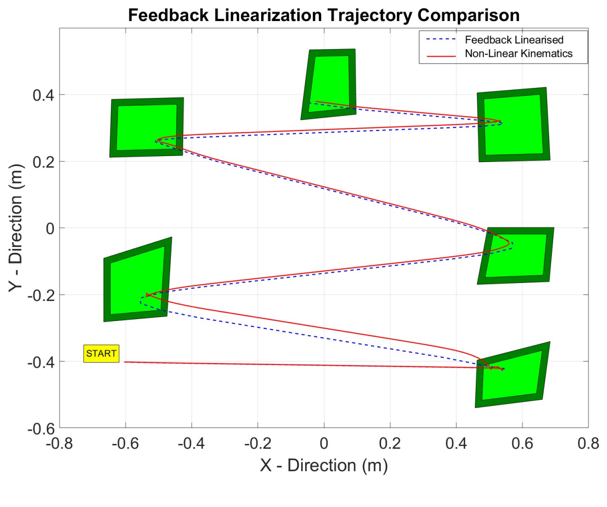

Figure 1: Comparison between Linear and Non-Linear system models

## *STL Constraint-based Simulations*

Now we test our linear discrete-time model for the two examples, {3} and {4} mentioned in the previous section. We chose a prediction horizon $H = h = 20$ steps where $H$ is the prediction horizon, and $h$

is the predicate horizon. The predicate horizon captures the trajectory's future robustness values for $h$ steps. The state bounds are $|\mathbf{x(1:2)}| \leq 1m$ for the positions, and $|\mathbf{x(3:4)}| \leq 1m/s$ for the velocities. The control input bounds are $|\mathbf{u}| \leq 0.25m/s^2$. The MILP solver attempts to minimize the cost function mentioned in 17. The very large number *M* is initialised as $10^3$ and the minimum desired robustness as $\rho^{\phi}_{min} = 0.2$. The time range in seconds $[a, b]$ is given as [15,20] for the *Go-To-Goal* example and ([15,20], [25,30]) for the two waypoints in the *Multi-Waypoint* example. The simulation results are as shown in fig: (2) and (3)

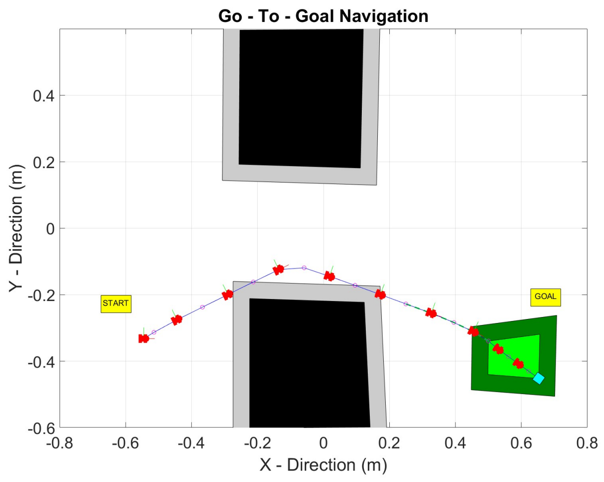

Figure 2: *An example of the Go-To-Goal STL specification*

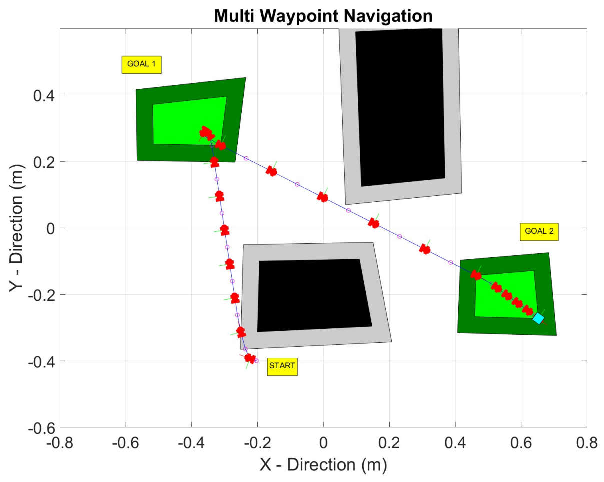

Figure 3: *An example of the Multi-Waypoint STL specification*

The lighter boxed areas around the goal and **obstacle** regions denote the boundary for minimum robustness($\rho^{\phi}_{min}$) desired around the obstacle. The AV is shown over the trajectory as it progresses toward the goal region. We observed the following results for the examples:

- For the *Go-To-Goal* example, the system was completely safe with positive robustness throughout the simulation period. The robustness boundary areas for the lower obstacle's two corners are violated but still fall within $0 < \rho^{\phi}(t) < 0.2$.The solver takes **3.5** seconds to calculate a feasible trajectory after which the STL specification asks for the AV to reach the goal within *20* secs.
- For the *Multi-Waypoint* example, the system is again completely safe with minor discrepancies. The target time range to reach each waypoint was also satisfied, as the velocities were observable and controllable states that could vary within the provided range. The solver takes *4.7* seconds to calculate a feasible trajectory. This is a bit longer than the first example, given the complexity of the problem at hand.
- There were a few infeasible solutions that we saw initially during the simulations. They were primarily due to providing the incorrect time ranges for $\{4\}$. This was because it was difficult to gauge the exact time range in which the AV would reach the waypoints. A workaround is it includes larger time frames such that "$ev_{[a,b]}$" operator has a larger frame to compute within the control input bounds.

## Conclusion

In this paper, we focused on developing a signal temporal logic based constraints approach to find safe and optimal solutions for autonomous vehicle navigation tasks. We use a model predictive control approach to apply the optimal control inputs calculated using the mixed integer linear solver. The STL constraints are user inputs provided to the model, first parsed to identify important information such as goal coordinates, obstacle regions, and time ranges. Once we acquire this information, along with the state and control bounds, we formulate an MILP problem that is solved recursively subject to satisfaction and maintenance of a minimum positive robustness value throughout the simulation. The model, constraints, and cost function are all linear in length, reducing calculation time as we are required to calculate smaller MILPs for most of the problem. Considering that we apply the safety constraints only at critical locations along the trajectory, we are not always guaranteed an optimal solution. We apply the approach to an AV model particularly focusing on safety for an autonomous navigation task. Two navigation task-focused numerical examples are solved, where feasible trajectories are generated. The computed optimal solutions are applied to navigate the AV safely to the waypoints. Future work for this approach demands expanding the application to non-linear systems to conduct real-time experiments on smaller vehicles in a controlled environment. We are also planning on making the model robust to disturbances as our application space is eventually for off-road 3D environments with higher fidelity models and environment. The STL parser would also be upgraded to work with multiple constraints without causing LIE explosion for the MILP solver. We aim to create efficient models for real-time applications on off-road vehicles.

## References

1. M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," Automatic Control, IEEE Transactions on, vol. 53, no. 1, pp. 287–297, 2008.
2. Wolff, Eric M., Ufuk Topcu, and Richard M. Murray. "Optimization-based trajectory generation with linear temporal logic specifications." 2014 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2014.
3. R. Alur, T. A. Hazinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. Proc. IEEE, 88(7):971–984, 2000.
4. R. Koymans, "Specifying real-time properties with metric temporal logic," Real-time systems, vol. 2, no. 4, pp. 255–299, 1990.
5. G. E. Fainekos and G. J. Pappas, Robustness of temporal logic specifications. Springer, 2006.
6. Lindemann, Lars, and Dimos V. Dimarogonas. "Robust motion planning employing signal temporal logic." 2017 American Control Con,ference (ACC). IEEE, 2017.

7. Saha Sayan, and A. Agung Julius. ”An MILP approach for real-time optimal controller synthesis with metric temporal logic specifications.” 2016 American Control Conference (ACC). IEEE, 2016.

8. C. Baier, J.-P. Katoen et al., Principles of model checking. MIT press Cambridge, 2008, vol. 26202649.

9. M. V. Kothare, V. Balakrishnan, and M. Morari, “Robust constrained model predictive control using linear matrix inequalities,” Automatica, vol. 32, no. 10, pp. 1361–1379, Oct. 1996.

10. H. Kress-Gazit, G. E. Fainekos, and G. J. Pappasenvironments Logic based Reactive Mission and Motion Planning,” IEEE Transactions on Robotics, vol. 25, no. 6, pp. 1370–1381, 2009.

11. T. Wongpiromsarn, U. Topcu, and R. M. Murray, “Receding horizon control for temporal logic specifications.” in Hscc. ACM, 2010, pp. 101–110.

12. E. M. Wolff, U. Topcu, and R. M. Murray, “Optimization-based Control of Nonlinear Systems with Linear Temporal Logic Specifications,” in Proc. of the International Symposium on Robotics Research (ISRR), 2014.

13. V. Raman, A. Donz´e, M. Maasoumy, R. M. Murray, A. Sangiovanni- Vincentelli, and S. A. Seshia, “Model Predictive Control with Signal Temporal Logic Specifications,” in CDC, 2014.

14. V. Raman, A. Donz´e, D. Sadigh, R. M. Murray, and S. A. Seshia, “Reactive synthesis from signal temporal logic specifications,” in Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control. ACM, 2015, pp. 239–248.

15. N. Aréchiga, ”Specifying Safety of Autonomous Vehicles in Signal Temporal Logic,” 2019 IEEE Intelligent Vehicles Symposium (IV), 2019, pp. 58-63, doi: 10.1109/IVS.2019.8813875.

16. Y. E. Sahin, R. Quirynen and S. D. Cairano, ”Autonomous Vehicle Decision-Making and Monitoring based on Signal Temporal Logic and Mixed-Integer Programming,” 2020 American Control Conference (ACC), 2020, pp. 454-459, doi: 10.23919/ACC45564.2020.9147917.

17. A. Dokhanchi, B. Hoxha, and G. Fainekos, “On-Line Monitoring for Temporal Logic Robustness,” in Runtime Verification. Springer, 2014, pp. 1–20.

18. S. M. LaValle, Planning Algorithms. Cambridge, U.K.: Cambridge Univ. Press, 2006.

19. Sadraddini, Sadra, and Calin Belta. ”Robust temporal logic model predictive control.” 2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton). IEEE, 2015.

## Contact Information

Aditya Parameshwaran
Graduate Research Assistant, Department of Mechanical Engineering, Clemson University
e-mail: aparame@clemson.edu

Yue Wang
Professor, Department of Mechanical Engineering, Clemson University
e-mail: yue6@g.clemson.edu

## Acknowledgments

This work was supported by the Simulation Based Reliability and Safety Program for modeling and simulation of military ground vehicle systems, under the technical services contract No. W56HZV-17-C-0095 with the U.S. Army DEVCOM Ground Vehicle Systems Center (GVSC).
Distribution A. Approved for public release; distribution unlimited. (OPSEC 7039)