Minimum-Violation Velocity Planning with Temporal Logic Constraints

Patrick Halder[1] and Matthias Althoff[2]

***Abstract*— Motion planning for autonomous agents requires compliance with a variety of constraints. However, in some situations, all constraints cannot be simultaneously satisfied, e.g., due to the misbehavior of other traffic participants. In such situations, one strategy for guaranteeing a feasible behavior of the system is deliberately ignoring less important constraints while maintaining the satisfaction of more important ones. We address this problem by presenting a novel velocity planner for an autonomous shuttle that violates temporal logic constraints as little as possible in the sense that important constraints are violated last. In particular, we provide an A^*-based velocity planner that minimally violates a hierarchically ordered set of constraints formalized in signal temporal logic. We apply our approach to challenging scenarios from the CommonRoad benchmark suite, showing that our proposed method yields easily interpretable and explainable decisions.**

Fig. 1. The autonomous shuttle GRT 3 of ZF (picture: ZF Group).

I. Introduction

Compliance with the intended driving behavior of an autonomous shuttle is a particularly hard task to solve, as there are various constraints to be satisfied simultaneously. Some examples of system constraints are safety and traffic regulation rules, mission specifications, and comfort requirements. During operation, the satisfaction of all constraints often cannot be guaranteed due to the potential misbehavior of other traffic participants. Nonetheless, especially in such situations, the explainability and justification of the chosen actions are significant. Minimum-violation planning tackles this challenge by finding a plan when the violation of constraints is unavoidable [1]. It tries to restore the system to a state of compliance as fast and with as little additional constraint violation as possible.

Motion planning is often separated into a lateral and longitudinal subproblem [2]. In this work, we focus on the latter and present a novel minimum-violation velocity planner for autonomous vehicles. An application of our planner is ZF's autonomous shuttle (see Fig. 1), which moves along a predefined path using a magnetic positioning system according to an online computed target velocity profile [3].

A. Related Work

Subsequently, we review related works on formalizing constraints, the qualitative preferences between them, and minimum-violation planning.

Formalization of Constraints and Quantification of Constraint Violation: Regarding motion planning, temporal logic is often used to formalize traffic rules because of their inherent temporal nature [4]. For instance, [5], [6] formalize traffic rules in metric temporal logic (MTL) based on German traffic laws and court decisions. Unfortunately, to the best of our knowledge, no set of formalized constraints solely dedicated to the operation of autonomous shuttles is available.

The predominant temporal formalization language in minimum-violation planning is linear temporal logic (LTL) (e.g., see [7]–[9]) and subsets of it, such as syntactically co-safe LTL (e.g., see [10]–[12]). However, without additional assumptions, LTL does not provide quantitative semantics. Thus, defining a cost function for a motion planning problem subject to LTL constraints can be cumbersome, as a metric quantifying the violation must be found manually [13], [14]. In contrast, signal temporal logic (STL) offers the quantitative semantics of *robustness* [15], which indicates the level of satisfaction or violation of a system's behavior with respect to a constraint.

Qualitative Preferences of Constraints: In many multi-objective motion planning approaches, preferences between constraints are quantified and expressed by scalar weights [2]. In contrast, rulebooks [16] define qualitative relations between constraints as a preorder. That is, they prioritize compliance with constraints, where several constraints can have the same priority. Based on these relations, rulebooks induce a preorder on trajectories, including equal priorities between them. This ambiguity between trajectories is seen as beneficial by the authors, as it provides additional freedom of choice. In [17], such ambiguities are resolved by using a decision tree to provide one best output trajectory.

[1]Patrick Halder is with ZF Friedrichshafen AG, 88046 Friedrichshafen, Germany, and the Department of Informatics, Technical University of Munich, 85748 Garching, Germany.

[2]Matthias Althoff is with the Department of Informatics, Technical University of Munich, 85748 Garching, Germany.

{patrick.halder, althoff}@tum.de

Minimum-Violation Planning: Most minimum-violation planning approaches operate on a sampled configuration space, use constraints formalized in temporal logic, and apply graph-search-based methods. These approaches translate the temporal logic constraints into a weighted transition system and combine this transition system with the sampled configuration space in a product automaton. Therein, the shortest accepting run correlates with the minimum-violation trajectory [7], [18], [19]. Some approaches build the product automaton explicitly [20]–[22], whereas others construct it incrementally using sampling-based algorithms [8], [23]. Furthermore, some approaches avoid converting the LTL formulas into finite automata and immediately create a combined transition system [9], [11], [14].

Only a few minimum-violation planning approaches use constraints formalized in STL. The authors of [24] develop a sampling-based minimum-violation planner using STL constraints. The costs in the planning problem are composed from a combination of space and time robustness. The authors of [25] show that the robustness property of STL is well suited for encoding interpretable driving-behavior preferences.

An exception among the presented methods of minimum-violation planning is [26]. The authors transfer the constraints of the planning problem successively from hard to soft constraints and solve an optimal control problem in each computation cycle until a feasible solution is found. The formulation of their constraints is inspired by STL.

B. Contributions

In this paper, we present a novel A^*-based minimum-violation velocity planner for longitudinal motion applications, which are subject to constraints formalized in STL. In detail, our contributions are:

- combining a lattice representation of the configuration space with an A^* search to solve a lexicographic optimization problem,
- formalizing 32 STL constraints specifying the movement of an autonomous shuttle along a predefined reference path,
- redefining the robustness of the temporal *globally* operator to quantify the level of constraint violation over the planning time horizon, and
- presenting a method to significantly reduce the overall number of constraint evaluations in the A^* search process.

The remainder of this paper is structured as follows. In Sec. II, we introduce the necessary preliminaries and state the motion planning problem. Sec. III describes the redefinition of the robustness and presents our STL constraints. Sec. IV explains the minimum-violation planning algorithm and the approach to reduce the number of rule evaluations. Finally, we present the results of numerical experiments in Sec. V and conclude in Sec. VI.

II. Preliminaries and Problem Statement

In this section, we present the semantics of STL, introduce totally ordered rulebooks, and formalize our minimum-violation planning problem.

A. Signal Temporal Logic

STL facilitates temporal reasoning over real-valued trajectories [15]. Let a discrete time step $k \in \mathbb{N}_0$ correspond to the time $t_k = k\Delta t$, where $\Delta t \in \mathbb{R}^+$ is the time increment. A discrete-time trajectory is a map $\boldsymbol{\tau} : \mathbb{K} \to \mathbb{R}^n$, where $\mathbb{K} = [k_0, k_f] \subseteq \mathbb{N}_0$ is the discrete time domain, with k_0 and k_f being the initial and final time step, respectively, and $k_0 \leq k_f$. The state of a trajectory at time step k is denoted as $\boldsymbol{\tau}_k$. We refer to $(\boldsymbol{\tau}, I)$ as the portion of $\boldsymbol{\tau}$ in the time interval $I \subseteq \mathbb{N}_0$. When $I = [k, k_f]$, we simply write $(\boldsymbol{\tau}, k)$. In order to map a trajectory $\boldsymbol{\tau}$ to the domain of Boolean values $\mathbb{B}$, we define predicates of the form $\mu := p(\boldsymbol{\tau}_k) \geq c$, with $p : \mathbb{R}^n \to \mathbb{R}$ and $c \in \mathbb{R}$. An STL formula φ is defined by the following grammar [27, Sec. 2.1]:

$$\varphi := \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathbf{U}_I \varphi_2 \mid \varphi_1 \mathbf{S}_I \varphi_2,$$

where φ_1, φ_2 are STL formulas, and $\neg, \wedge$ are negation and conjunction, respectively. $\mathbf{U}_I$ is the temporally bounded *until* operator, and $\mathbf{S}_I$ is the temporally bounded *since* operator. Note that I is omitted from the grammar when $I = [k_0, k_f]$. We denote the satisfaction of a formula φ by a trajectory $\boldsymbol{\tau}$ in the time interval $[k, k_f]$ by $(\boldsymbol{\tau}, k) \models \varphi$. For the semantics of STL, we refer to [27, Sec. 2.1]. Let us introduce $\top := \neg(\neg\varphi \wedge \varphi)$ and the shorthand notations *future* $\mathbf{F}_I\varphi := \top \mathbf{U}_I \varphi$, *once* $\mathbf{O}_I\varphi := \top \mathbf{S}_I \varphi$, and *globally* $\mathbf{G}_I\varphi := \neg\mathbf{F}_I\neg\varphi$, as described in [27, Sec. 2.1]. Additionally, the implication $\varphi_1 \Rightarrow \varphi_2$ can be written as $\neg\varphi_1 \vee \varphi_2$.

Robustness quantifies the level of compliance of an STL formula, where a positive value indicates satisfaction, and a negative value indicates violation [27]. Formally, the robustness $\rho(\varphi, \boldsymbol{\tau}, k)$ of a formula φ and a trajectory $\boldsymbol{\tau}$ at time step k is defined as follows [27, Sec. 2.2]:

$$\begin{aligned}
\rho(\top, \boldsymbol{\tau}, k) &:= \infty, \\
\rho(p(\boldsymbol{\tau}) \geq c, \boldsymbol{\tau}, k) &:= p(\boldsymbol{\tau}_k) - c, \\
\rho(\neg\varphi, \boldsymbol{\tau}, k) &:= -\rho(\varphi, \boldsymbol{\tau}, k), \\
\rho(\varphi_1 \wedge \varphi_2, \boldsymbol{\tau}, k) &:= \min\big(\rho(\varphi_1, \boldsymbol{\tau}, k), \rho(\varphi_2, \boldsymbol{\tau}, k)\big), \\
\rho(\mathbf{G}_I\varphi, \boldsymbol{\tau}, k) &:= \min_{k' \in (k+I)\cap\mathbb{K}} \big(\rho(\varphi, \boldsymbol{\tau}, k')\big), \\
\rho(\mathbf{O}_I\varphi, \boldsymbol{\tau}, k) &:= \max_{k' \in (k-I)\cap\mathbb{K}} \big(\rho(\varphi, \boldsymbol{\tau}, k')\big), \\
\rho(\varphi_1 \mathbf{S}_I \varphi_2, \boldsymbol{\tau}, k) &:= \max_{k' \in (k-I)\cap\mathbb{K}} \big(\min(\rho(\varphi_2, \boldsymbol{\tau}, k'), \\
&\qquad \min_{k'' \in (k', k]} (\rho(\varphi_1, \boldsymbol{\tau}, k''))\big)\big).
\end{aligned}$$

B. Rules and Rulebooks

In the remainder of this work, we refer to rules as a general term summarizing different types of constraints (e.g., traffic rules, mission specifications, or comfort requirements). Let a rule be an STL formula φ. The relation $\rho(\varphi, \boldsymbol{\tau}_1, k) > \rho(\varphi, \boldsymbol{\tau}_2, k)$ indicates that $\boldsymbol{\tau}_1$ violates φ to a smaller extent (or satisfies it to a greater extent) than $\boldsymbol{\tau}_2$.

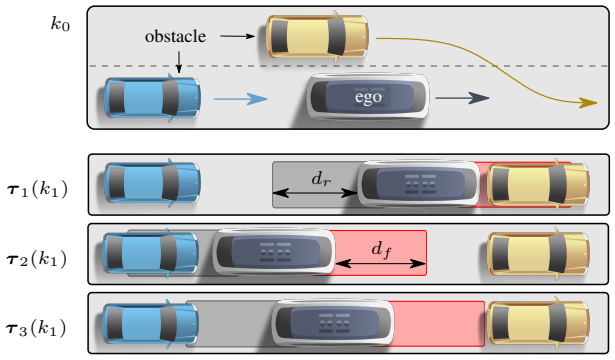

Fig. 2. An example scenario at initial time step k_0 with three possible trajectories $\boldsymbol{\tau}_1$, $\boldsymbol{\tau}_2$, and $\boldsymbol{\tau}_3$ at time step $k_1 > k_0$. The required distance to the front and the rear of the ego vehicle is represented by d_f and d_r, respectively. A rulebook $B_1 = (\textit{keep front distance} > \textit{keep rear distance})$ induces the order $\boldsymbol{\tau}_3 > \boldsymbol{\tau}_2 > \boldsymbol{\tau}_1$, and a rulebook $B_2 = (\textit{keep rear distance} > \textit{keep front distance})$ induces the order $\boldsymbol{\tau}_1 > \boldsymbol{\tau}_3 > \boldsymbol{\tau}_2$.

We sort the rules $\varphi_i \in \mathcal{R}$ in descending order according to their importance and define a totally ordered rulebook as a tuple $B := \langle \mathcal{R}, < \rangle$, where $\mathcal{R}$ is a set of rules and $<$ is a strict total order. A rule φ_i corresponds to a rank i in B, with $i \in 1, 2, \ldots, R$ and $R = |\mathcal{R}|$, where $i = 1$ is the highest rank, and $i = R$ is the lowest rank ($\varphi_1 \geq \varphi_i \geq \varphi_R$). Note that we do not consider the general case of preordered rulebooks in this work (cf. [16]).

Based on the totally ordered rules $\mathcal{R}$, a rulebook B induces the lexicographic order [28, (8.17)] on a set of trajectories $\mathcal{T}$ [16]. Given two trajectories $\boldsymbol{\tau}_1, \boldsymbol{\tau}_2 \in \mathcal{T}$, we write $\boldsymbol{\tau}_1 > \boldsymbol{\tau}_2$ if $\boldsymbol{\tau}_1$ is lexicographically better than $\boldsymbol{\tau}_2$ with respect to a rulebook B. To illustrate the induced order on a set of trajectories by a rulebook, Fig. 2 shows an example scenario with three trajectories and two different rulebooks.

C. Reference Path and Vehicle Model

Let us introduce the reference path $\Gamma : \mathbb{R} \to \mathbb{R}^2$, which maps the arc length $s \in \mathbb{R}$ to a position coordinate $z \in \mathbb{R}^2$ in the global Cartesian frame. Further, let the set of admissible states be $\mathcal{X} \subseteq \mathbb{R}^n$ and the set of admissible inputs be $\mathcal{U} \subseteq \mathbb{R}^m$, where $x \in \mathcal{X}$ represents the state and $u \in \mathcal{U}$ represents the input. We model the dynamics of the autonomous vehicle along the reference path Γ with:

$$x_{k+1} = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} x_k + \begin{pmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{pmatrix} u_k, \tag{1}$$

where $x = (s, v)^T$ corresponds to the position and the velocity of the ego's center point along Γ, and the input $u = a$ is the acceleration. In the remainder of this paper, the notations $\underline{\square}$ and $\overline{\square}$ are used to indicate a minimum and maximum value of a variable $\square$, respectively. We assume the states and the inputs of system (1) to be constrained by:

$$\begin{aligned} \underline{v} \leq v \leq \overline{v}, \\ \sqrt{a^2 + (v^2 \kappa_s)^2} \leq \overline{a}, \\ \underline{a} \leq a, \end{aligned} \tag{2}$$

where κ_s is the curvature of $\Gamma(s)$. We denote the solution of (1) for an input trajectory $u(\cdot)$ and an initial state $x(k_0)$ by $\boldsymbol{\tau}(k, x(k_0), u(\cdot))$.

D. Minimum-Violation Planning Problem

We define the minimum-violation planning problem as a lexicographic optimization problem [29, (13)]:

$$\begin{aligned} &\max_{\boldsymbol{\tau} \in \mathcal{T}} \ \rho(\varphi_i, \boldsymbol{\tau}, [k_0, k_f]) \\ &\text{s.t. } \rho(\varphi_j, \boldsymbol{\tau}, [k_0, k_f]) = \rho(\varphi_j, \boldsymbol{\tau}_j^*, [k_0, k_f]), \\ &\text{with } j = 1, 2, \ldots, i-1, \ \text{when } i > 1, \\ &\text{for } i = 1, 2, \ldots, R, \end{aligned} \tag{3}$$

where i corresponds to a rank in B. This problem describes a series of $i = 1, 2, \ldots, R$ optimization problems evaluated sequentially, each aiming to find an optimal trajectory $\boldsymbol{\tau}_i^*$ maximizing the robustness regarding a rule φ_i. The i^{th} optimization problem is constrained by the j optimal robustness values $\rho(\varphi_j, \boldsymbol{\tau}_j^*, [k_0, k_f])$ resulting from the optimal trajectories $\boldsymbol{\tau}_j^*$ of the problems $j = 1, 2, \ldots, i-1$ when $i > 1$. Informally, the optimal robustness values for higher rank rules constrain the solution space of the optimization problems of lower rank rules. Please note that we incorporate the state and input constraints (2) as rules φ_i into B.

III. Robustness and Rules

In this section, we redefine the robustness of the temporal *globally* operator and present our formalized STL rules for an autonomous shuttle application.

a) Redefinition of Robustness: To quantify the rule violation of a trajectory, we draw inspiration from [24], which defines a new temporal operator similar to the *globally* operator, whose semantics combines space and time robustness. In contrast, we merely redefine the robustness of the *globally* operator $\mathbf{G}_I$ and derive a measure quantifying the rule violation of a trajectory over the planning time horizon as:

$$\rho(\mathbf{G}_I\, \varphi, \boldsymbol{\tau}, k) := \sum_{k' \in (k+I) \cap \mathbb{K}} \min\big(0, \rho(\varphi, \boldsymbol{\tau}, k')\big) \Delta t. \tag{4}$$

Informally, this semantics sums up the extent to which a formula φ is violated by a trajectory $\boldsymbol{\tau}$ over the planning time horizon, based on a numerical integration. Note that we bound the robustness by zero and only consider negative robustness values. As presented subsequently, all our rules are of type $\mathbf{G}_I\, \varphi$; hence, according to (3) and (4), only the violation of rules by a trajectory generates costs in the optimization problem. To illustrate this, Fig. 3 presents three velocity profiles subject to a maximum velocity rule and their respective robustness values over an increasing time horizon. Herein, the motivation to redefine the robustness of the $\mathbf{G}_I$ operator becomes evident since, using its standard definition as presented in Sec. II-A, the robustness for $\boldsymbol{\tau}_1$ and $\boldsymbol{\tau}_3$ would be equal. We consider this to be counterintuitive in our application because, although the maximum violation of the rule is the same for both trajectories, $\boldsymbol{\tau}_1$ returns to a state of rule compliance, whereas $\boldsymbol{\tau}_3$ does not.

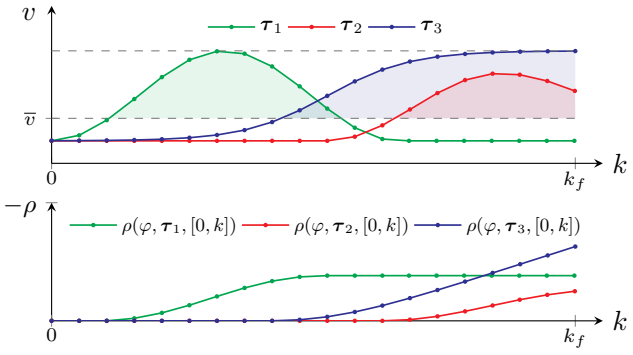

Fig. 3. An example of the robustness of three trajectories $\boldsymbol{\tau}_1$, $\boldsymbol{\tau}_2$, and $\boldsymbol{\tau}_3$ and rule $\varphi = \mathbf{G}\,\phi$, with $\phi := v < \overline{v}$ and $\rho(\phi, \boldsymbol{\tau}, k) = \overline{v} - v$, over an increasing time horizon. The induced order on the trajectories in the time horizon $[0, k_f]$ is $\boldsymbol{\tau}_2 > \boldsymbol{\tau}_1 > \boldsymbol{\tau}_3$.

b) Formalized STL Rules: Subsequently, we formalize rules inspired by the German traffic law and by [5], [6]. We want to emphasize that some rules are simplified and we do not claim any legal correctness or completeness. Nevertheless, we believe that our formalized rules capture the essence of the law text, as shown in the experiments in Sec. V.

Tab. I shows our formalized STL rules. As they are of a simple structure, we focus on explaining the used predicates and functions subsequently. For readability, we mark functions with [f], whereas the predicates remain unmarked. We denote the ego vehicle by e and obstacles by $o \in \mathcal{O}$, where $\mathcal{O}$ is the set of obstacles in a scenario. The position s_o corresponds to the center point of obstacle o projected onto Γ, δ_o is the minimum Euclidean distance of the obstacle's center point to Γ, and v_o is the obstacle's velocity. Moreover, let us introduce the abbreviation *VRU*, which stands for *vulnerable road user*. Note that rule φ_{24} considers intersections (left turns are excluded) and crosswalks [30]. Notably, some of the rules can be interpreted as cost functions (e.g., rule φ_{32}). The predicates and functions are as follows:

- *keeps_vehicle_constraints*(e): Indicates if the vehicle constraints (cf. (2)) are satisfied;
- *contact*(e, o, ε): Indicates if there is a geometrical overlap between the occupancy of ego e and obstacle o. The optional parameter ε specifies a rule-dependent expansion of the respective occupancies. For instance, in rule φ_8, ε corresponds to an enlargement of the ego's occupancy in its driving direction. For the sake of simplicity, we omit a detailed description of ε, as its meaning directly follows from the context of the rules;
- *type*(o) [f]: Returns the type of obstacle o (cf. [30]);
- *is_emergency_vehicle_on_duty*(o): Indicates if obstacle o is an emergency vehicle and currently on duty;
- *is_leading*(e, o): Indicates if obstacle o is ahead of ego e along Γ. Therefore, we evaluate $(s < s_o) \wedge (\delta_o < \overline{\delta})$, where $\overline{\delta}$ is a predefined threshold;
- *dist*(e, o) [f]: Returns the Euclidean distance between the occupancy of ego e and obstacle o;
- *safe_dist*(e, o) [f]: Returns the safe distance between ego e and obstacle o as defined in [31, (4)];
- *passing*(e, o): Indicates if ego e passes obstacle o by evaluating $(s > s_o) \wedge \mathbf{O}_{[0,1]}(s \leq s_o)$;
- *speed_limit_conditions*$(sc, \{weather, road\})$ [f]: Returns a maximum velocity limit depending on the weather or road conditions in scenario *sc* (cf. [30]);
- *in_degradation*(e): Indicates whether ego e has an internal error;
- *speed_limit_lane*(e) [f]: Returns the speed limit of the lane on which the center point of ego e is located;
- *has_right_of_way*(o, e): Indicates if obstacle o has priority over ego e, based on traffic signs, traffic lights, and the future path of the ego vehicle (cf. [6]);
- *occupies*$(\{e, o\}, \textit{object\_type})$: Indicates if ego e or obstacle o occupies a road object of the respective type (cf. [6]);
- *crosses_stop_line*(e): Indicates if ego e crosses a stop line by evaluating $(s > s_{stop}) \wedge \mathbf{O}_{[0,1]}(s \leq s_{stop})$, with $s_{stop} = s\_stop\_line(\{\textit{stop\_sign, traffic\_light}\})$;
- *s_stop_line*$(\{\textit{stop\_sign, traffic\_light}\})$ [f]: Returns the stop line position corresponding to a stop sign or traffic light;
- *is_red*(tl): Indicates if traffic light *tl* is currently *red*;
- *s_in_range*$(\underline{s}, \overline{s})$: Indicates if the current position s is in the interval $[\underline{s}, \overline{s}]$;
- $F_{res}(e)$ [f]: Returns the current driving resistance force of ego e (cf. [32, Sec. 4.1]).

Further, we introduce the following adjustable parameters: $\underline{s}_{sc}$ and $\overline{s}_{sc}$ are station limits; $\underline{v}_{cl,b}$, $\underline{v}_{cross}$, $\overline{v}_{bic}$, $\overline{v}_{bus}$, $\overline{v}_{np}$, $\overline{v}_{deg}$, $\overline{v}_{mission}$, $\overline{v}_{bump}$, and $\overline{v}_{noise}$ are velocity limits; $\overline{d}_{bic}$, $\overline{d}_{bus}$, and $\overline{d}_{np}$ are limits on the Euclidean distance; $\underline{a}_{abr}$, $\overline{a}_{mission}$, $\overline{a}_{lat,comf}$, and $\overline{a}_{lon,comf}$ are acceleration limits; $\overline{P}$ is a power limit; Δv_{flow} is a velocity range; I_{stop}, I_{abr}, and I_{sc} are time intervals; and k_{sc} is a time step.

IV. Algorithm

Our approach to solving problem (3) can be summarized as follows: We discretize the configuration space using a lattice and sample time, position, and velocity equidistantly. Within the lattice graph, a lexicographic A^* search is performed, where the traversal costs of a trace are based on its robustness regarding the rules of a rulebook. We omit a detailed description of the A^* algorithm and refer to [33] for further details. The optimal trajectory $\boldsymbol{\tau}^*$ corresponds to the lexicographically best trace in the lattice graph. We increase the performance of our algorithm by reducing the number of necessary rule evaluations and using a custom priority queue.

A. Lattice Representation

In order to explain our velocity planning algorithm, we first introduce the directed acyclic lattice graph $G = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is a set of nodes, and $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ is a set of edges. Each node is uniquely specified by a state $x = (s, v)^T \in \mathcal{X}$ and a time step $k \in \mathbb{N}_0$. Further, let State(n) and TimeStep(n) return the state x and the time step k associated with node $n \in \mathcal{N}$. Edges are added between nodes if the corresponding state transition satisfies the vehicle model (1) subject to (2) [34].

TABLE I

STL RULES FOR AN AUTONOMOUS SHUTTLE APPLICATION.

Description	Rule	Description	Rule
Vehicle constraints	$\varphi_1 = \mathbf{G}(\textit{keeps\_vehicle\_constraints}(e))$	Mission acc. limit	$\varphi_{17} = \mathbf{G}(\lvert a\rvert \leq \overline{a}_{mission})$
Stop after crash	$\varphi_2 = \mathbf{G}(\mathbf{O}(\textit{contact}(e,o)) \Rightarrow v = 0)$	Lane velocity limit	$\varphi_{18} = \mathbf{G}(v \leq \textit{speed\_limit\_lane}(e))$
Collision avoid. VRU	$\varphi_3 = \mathbf{G}(\textit{type}(o) = \textit{vru} \wedge \textit{contact}(e,o) \Rightarrow v = 0)$	Right of way	$\varphi_{19} = \mathbf{G}(\textit{has\_right\_of\_way}(o,e) \Rightarrow \neg\textit{contact}(e,o,\varepsilon))$
Collision avoid. vehicle	$\varphi_4 = \mathbf{G}(\textit{type}(o) = \textit{vehicle} \wedge \textit{contact}(e,o) \Rightarrow \lvert v - v_o\rvert = 0)$	Stop at crosswalk	$\varphi_{20} = \mathbf{G}(\textit{occupies}(o,\textit{crosswalk}) \Rightarrow \neg\textit{contact}(e,o,\varepsilon))$
Collision avoid. static	$\varphi_5 = \mathbf{G}(\textit{type}(o) = \textit{static} \wedge \textit{contact}(e,o) \Rightarrow v = 0)$	Stop at stop sign	$\varphi_{21} = \mathbf{G}(\textit{crosses\_stop\_line}(e) \Rightarrow \mathbf{O}_{I_{stop}}(v = 0))$
Emergency vehicle	$\varphi_6 = \mathbf{G}(\textit{is\_emergency\_vehicle\_on\_duty}(o) \wedge \neg\textit{contact}(e,o,\varepsilon))$	Stop at red traffic light	$\varphi_{22} = \mathbf{G}(\textit{is\_red}(tl) \wedge \textit{crosses\_stop\_line}(e) \Rightarrow s \leq s_{stop})$
Safe distance	$\varphi_7 = \mathbf{G}(\textit{is\_leading}(e,o) \Rightarrow \textit{dist}(e,o) \geq \textit{safe\_dist}(e,o))$	Abrupt braking	$\varphi_{23} = \mathbf{G}(a \geq \underline{a}_{abr})$
Clearance front	$\varphi_8 = \mathbf{G}(\textit{contact}(e,o,\varepsilon) \Rightarrow v = 0)$	Blocking of crossings	$\varphi_{24} = \mathbf{G}(\textit{occupies}(e,\textit{crossing}) \Rightarrow v \geq \underline{v}_{cross})$
Clearance back	$\varphi_9 = \mathbf{G}(v \geq \underline{v}_{cl,b} \Rightarrow \neg\textit{contact}(e,o,\varepsilon))$	Preserve traffic flow	$\varphi_{25} = \mathbf{G}(v \geq (\textit{speed\_limit\_lane}(e) - \Delta v_{flow}))$
Passing of bicycles	$\varphi_{10} = \mathbf{G}(\textit{type}(o) = \textit{bicycle} \wedge \textit{passing}(e,o) \wedge \textit{dist}(e,o) \leq \overline{d}_{bic} \Rightarrow v \leq \overline{v}_{bic})$	Lat. comf. acceleration	$\varphi_{26} = \mathbf{G}(\lvert v^2\kappa_s\rvert \leq \overline{a}_{lat,comf})$
Passing of buses	$\varphi_{11} = \mathbf{G}(\textit{type}(o) = \textit{bus} \wedge \textit{passing}(e,o) \wedge \textit{dist}(e,o) \leq \overline{d}_{bus} \Rightarrow v \leq \overline{v}_{bus})$	Lon. comf. acceleration	$\varphi_{27} = \mathbf{G}(\lvert a\rvert \leq \overline{a}_{lon,comf})$
Narrow passing	$\varphi_{12} = \mathbf{G}(\textit{passing}(e,o) \wedge \textit{dist}(e,o) \leq \overline{d}_{np} \Rightarrow v \leq \overline{v}_{np})$	Vertical comfort	$\varphi_{28} = \mathbf{G}(\textit{occupies}(e,\textit{speed\_bump}) \Rightarrow v \leq \overline{v}_{bump})$
Weather conditions	$\varphi_{13} = \mathbf{G}(v \leq \textit{speed\_limit\_conditions}(\textit{scenario},\textit{weather}))$	Schedule	$\varphi_{29} = \mathbf{G}(k = k_{sc} \Rightarrow \mathbf{O}_{I_{sc}}(\textit{s\_in\_range}(\underline{s}_{sc},\overline{s}_{sc})))$
Road conditions	$\varphi_{14} = \mathbf{G}(v \leq \textit{speed\_limit\_conditions}(\textit{scenario},\textit{road}))$	Power limitation	$\varphi_{30} = \mathbf{G}(vF_{res}(e) \leq \overline{P})$
Degradation	$\varphi_{15} = \mathbf{G}(\textit{is\_in\_degradation}(e) \Rightarrow v \leq \overline{v}_{deg})$	Noise emission	$\varphi_{31} = \mathbf{G}(v \leq \overline{v}_{noise})$
Mission velocity limit	$\varphi_{16} = \mathbf{G}(v \leq \overline{v}_{mission})$	Minimize acceleration	$\varphi_{32} = \mathbf{G}(a^2 = 0)$

B. Planning Procedure

Alg. 1 shows our minimum-violation velocity planning approach. The input is a lattice graph G, an initial state x_0, a rulebook B, and the final time step k_f. The output is the optimal final node n^*, where the trace leading to n^* encodes the minimum-violation velocity profile. In contrast to the standard A^* algorithm, we do not use scalar costs. However, we compare traces in G lexicographically according to their robustness regarding the rules in B. We present our modifications for the trace comparison (see line 16) and the priority queue representation (see lines 3, 5, 12, and 18) in the subsequent subsections.

C. Trace Comparison

The function BETTER$(\cdot)$ (see line 16 in Alg. 1) performs a lexicographic comparison between two competing traces, which end in the same node. Therefore, it evaluates and compares the robustness values of all rules $\mathcal{R}$ in the rulebook B for a trajectory $\boldsymbol{\tau}_{tr}$, which is induced by the respective traces. We refer to the robustness of a trace regarding a rule φ_i as $\rho_{tr}^{\varphi_i} = \rho(\varphi_i, \boldsymbol{\tau}_{tr}, [k_0, k_f^{tr}])$, with $k_0 \leq k_f^{tr} \leq k_f$.

Algorithm 1 MINIMUMVIOLATIONVELOCITYPLANNING

Input: Lattice G, initial state x_0, rulebook B, final time step k_f
Output: Optimal final node n^*

1: $n_0 \leftarrow$ SETINITIALNODE(x_0, B)
2: $\mathcal{C} \leftarrow \{\}, Q \leftarrow \{\}$
3: ADD(Q, n_0) ▷ Sec. IV-D.a
4: **while** true **do**
5: $n_p \leftarrow$ POP(Q) ▷ Sec. IV-D.b
6: **if** TIMESTEP(n_p) == k_f **then**
7: **return** $n^* \leftarrow n_p$
8: **end if**
9: INSERT($\mathcal{C}, n_p$)
10: **for** n_c in GETCHILDNODES(G, n_p) **do**
11: **if** $n_c \notin \mathcal{C} \wedge n_c \notin Q$ **then**
12: ADD(Q, n_c) ▷ Sec. IV-D.a
13: **end if**
14: **if** $n_c \in Q$ **then**
15: $n_{inc} \leftarrow Q(n_c)$
16: **if** BETTER(n_c, n_{inc}, B) **then** ▷ Sec. IV-C
17: REMOVE(Q, n_{inc})
18: ADD(Q, n_c) ▷ Sec. IV-D.a
19: **end if**
20: **end if**
21: **end for**
22: **end while**

An admissible heuristic never overestimates the costs of reaching the goal. In our application, this corresponds to not overestimating the negative robustness of a trace. Thus, we introduce $\boldsymbol{\tau}_h$ as the trace starting from the final state of $\boldsymbol{\tau}_{tr}$ at k_f^{tr} and ending at the time step k_f, determined based on vehicle model (1) with the constant inputs $\underline{a}$ or $\overline{a}$. For rules that are of the form $\varphi := v \leq \overline{v}$ or $\varphi := v \geq \underline{v}$ (e.g., see φ_{13} or φ_{25} in Tab. I), an admissible heuristic $\rho_h^{\varphi_i} = \rho(\varphi_i, \boldsymbol{\tau}_h, [k_f^{tr}, k_f])$ can be easily derived: The constant inputs $\underline{a}$ or $\overline{a}$ return system (1) to a rule-compliant state as fast as possible. Thus, there cannot be a trace with a smaller negative robustness value than the trace generated by the inputs $\underline{a}$ or $\overline{a}$ for rules of the mentioned form. We save the definition of $\rho_h^{\varphi_i}$ for the remaining rules for future work and set $\rho_h^{\varphi_i} = 0$.

Each trace corresponds to a tuple of robustness values $\boldsymbol{\varrho} = (\rho_{tr}^{\varphi_1} + \rho_h^{\varphi_1}, \ldots, \rho_{tr}^{\varphi_R} + \rho_h^{\varphi_R})$, where each entry is the sum of the trace and the heuristic robustness for each rule $\varphi_i \in \mathcal{R}$. To compare two traces in G, we compare their corresponding robustness tuples lexicographically. According to the lexicographic comparison rule (i.e., $(y_1, y_2) > (y_3, y_4)$ iff $y_1 > y_3 \vee (y_1 = y_3 \wedge y_2 > y_4)$, with $y \in \mathbb{R}$ [28]), only the lowest non-equal element of the tuples is decisive. To prevent unnecessary evaluations of rules, function BETTER$(\cdot)$ successively evaluates each rule φ_i of rulebook B for the competing traces and directly compares their robustness values. For instance, the robustness tuples $\boldsymbol{\varrho}_1 = (\text{-}2, \text{-}\mathbf{3}, \ldots)$ and $\boldsymbol{\varrho}_2 = (\text{-}2, \text{-}\mathbf{4}, \ldots)$ are equal for the robustness of rule φ_1 but already deviate for the robustness of rule φ_2. Thus, the result of the lexicographic comparison between $\boldsymbol{\varrho}_1$ and $\boldsymbol{\varrho}_2$ is independent of rules of rank $i > 2$, and the robustness does not need to be evaluated for these rules.

D. Priority Queue

To reduce the number of lexicographic comparisons, we present a custom priority queue. The main idea is to assign the priority queue elements based on a lexicographic comparison into unsorted buckets (likewise the *bucket sort* algorithm [35, Sec. 8.4]) and, subsequently, only search the most promising bucket based on the lexicographic sort algorithm [36, Alg. 3.1] (also known as the *most significant digit radix sort* algorithm [35, Sec. 8.3]) for the current best element. This contrasts with classic priority queue approaches, where a completely ordered list of elements is always maintained. Below, we illustrate with an example how we add and pop nodes to and from our priority queue.

a) Adding Nodes: We create a bucket $\mathcal{B}_i \in \mathcal{P}$ for each rank i in rulebook B (i.e., $\mathcal{P}$ contains $|\mathcal{P}| = R$ buckets). Function ADD$(\cdot)$ (see lines 3, 12, and 18 in Alg. 1) adds a new child node $n_c \in \mathcal{N}$ to Q by comparing its robustness tuple $\boldsymbol{\varrho}_c$ lexicographically with the robustness tuple $\boldsymbol{\varrho}^*$ of the popped (i.e., best) element from the previous iteration of the A^* algorithm. The node n_c is then assigned to bucket $\mathcal{B}_j$, where j is the rank of the lowest non-equal entry of the robustness tuples (cf. Sec. IV-C). Fig. 4 shows an example. Therein, ADD$(\cdot)$ is called on tuple $\boldsymbol{\varrho}_c = (\text{-}1, \text{-}\mathbf{2}, \text{-}3)$, which is compared with $\boldsymbol{\varrho}^* = (\text{-}1, \text{-}\mathbf{1}, \text{-}3)$. As $\boldsymbol{\varrho}_c$ deviates from $\boldsymbol{\varrho}^*$ on rank $i = 2$, it is assigned to bucket $\mathcal{B}_2$.

b) Popping Nodes: To find and pop the current lexicographically best node in Q, we only must search the bucket corresponding to the lowest-rank non-empty bucket $\mathcal{B}_i^*$. Therefore, function POP$(\cdot)$ (see line 5 in Alg. 1) sorts the best elements in $\mathcal{B}_i^*$ recursively into buckets corresponding to lower rank rules using the lexicographic sort algorithm and returns the lexicographically best element. The robustness tuple of this best element replaces the current value of $\boldsymbol{\varrho}^*$. Again, Fig. 4 shows an example. When POP$(\cdot)$ is called, the lowest-rank non-empty bucket $\mathcal{B}_2^*$ is evaluated. The best tuples regarding the robustness of rule φ_2, $(\text{-}1, \text{-}\mathbf{2}, \text{-}1)$ and $(\text{-}1, \text{-}\mathbf{2}, \text{-}3)$, are then transferred to bucket $\mathcal{B}_3^*$, where holds that $(\text{-}1, \text{-}2, \text{-}\mathbf{1}) > (\text{-}1, \text{-}2, \text{-}\mathbf{3})$. Thus, $(\text{-}1, \text{-}2, \text{-}1)$ replaces the current value of $\boldsymbol{\varrho}^*$, and the corresponding node is returned.

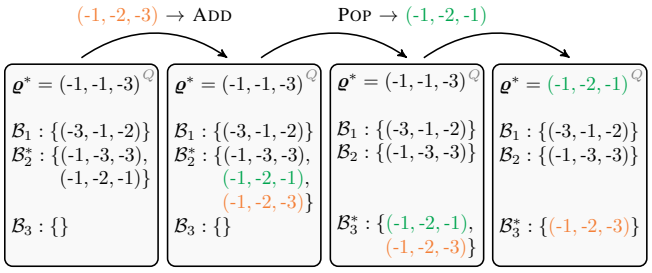

Fig. 4. An exemplary priority queue progress showing a priority Q already filled with nodes and their respective robustness tuples.

V. NUMERICAL EXPERIMENTS

To emphasize the interpretability of our velocity planner's outputs, we present three scenarios from the CommonRoad benchmark suite [30]. A reference path is given by a high-level planner, and the predicted trajectories of other traffic participants are provided in the scenarios. We use the rules from Tab. I and keep the presented order to form a rulebook B ($\varphi_1 > \varphi_2 > \cdots > \varphi_{32}$). We set $k_f = 15$ and the time increment $\Delta t = 0.4$ s. The algorithm is implemented in Python, and collision checks are performed with the CommonRoad Drivability Checker [37]. The experiments are executed on an Intel Core™ i5-10310U CPU. Animations of the scenarios are provided as supplementary material.

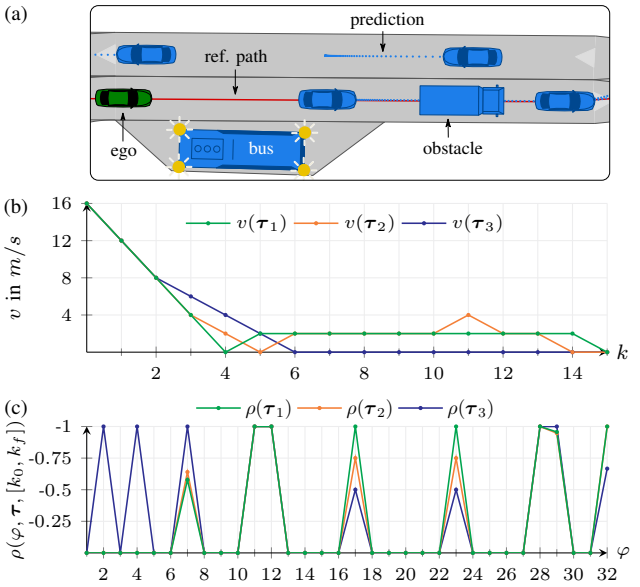

Fig. 5. Scenario I: Passing of a bus at a bus stop. (a) Start configuration of the scenario. (b) Minimum-violation velocity profile $\boldsymbol{\tau}_1$ and two lexicographically worse candidates $\boldsymbol{\tau}_2$ and $\boldsymbol{\tau}_3$. (c) Robustness for the rules according to Tab. I.

A. Scenario I

The first scenario, `DEU_Flensburg-6_2_T-1`, is an inner-city scenario, in which the ego vehicle passes a bus at a bus stop. Fig. 5 shows the start configuration of the scenario, the resulting minimum-violation velocity profile $\boldsymbol{\tau}_1$ and two lexicographically worse profiles $\boldsymbol{\tau}_2$ and $\boldsymbol{\tau}_3$ for comparison. We visualize the robustness per rule normalized to the range $[0, \text{-}1]$, where -1 corresponds to the minimum robustness value of either $\boldsymbol{\tau}_1$, $\boldsymbol{\tau}_2$, or $\boldsymbol{\tau}_3$. At the initial time step k_0, the ego vehicle already violates certain rules, e.g., the mission velocity limit φ_{16}. As a result, the safe-distance rule φ_7 and the passing-of-buses rule φ_{11} cannot be satisfied. Nevertheless, by performing a hard braking maneuver, the minimum-violation trajectory minimizes the negative robustness of these rules. The velocity plot shows that $\boldsymbol{\tau}_2$ applies less deceleration starting from $k = 3$ than $\boldsymbol{\tau}_1$. This leads to a higher violation of rule φ_7 by $\boldsymbol{\tau}_2$ and, thus, to the relation $\boldsymbol{\tau}_1 > \boldsymbol{\tau}_2$. As velocity profile $\boldsymbol{\tau}_3$ does not brake in time and causes a crash, the resulting order is $\boldsymbol{\tau}_1 > \boldsymbol{\tau}_2 > \boldsymbol{\tau}_3$. Due to the lexicographic preferences, the violation of rules on lower ranks (e.g., schedule rule φ_{29}) does not influence the resulting order of the trajectories in this scenario.

B. Scenario II

As a second example, we present the cut-in scenario `ZAM_Zip-2_1_T-1`, where an obstacle performs a cut-in maneuver in front of the ego vehicle, resulting in an unavoidable front-bumper crash. Fig. 6 presents the results. We can see that all three trajectories violate the stop-after-crash rule φ_2, as all continue driving for some time. However, velocity profile $\boldsymbol{\tau}_1$ generates the fastest stopping maneuver, resulting in the order $\boldsymbol{\tau}_1 > \boldsymbol{\tau}_2 > \boldsymbol{\tau}_3$. Furthermore, profile $\boldsymbol{\tau}_3$ is the only profile that does not come to a stop on the crosswalk and, hence, does not violate rule φ_{24}. However, as this rule is on the low rank $i = 24$, the corresponding violation is not pivotal to the decision process in this scenario.

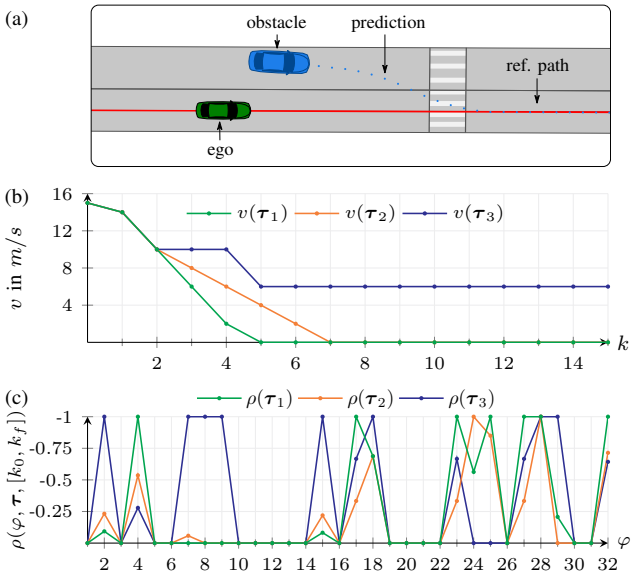

Fig. 6. Scenario II: Cut-in maneuver of an obstacle. (a) Start configuration of the scenario. (b) Minimum-violation velocity profile $\boldsymbol{\tau}_1$ and two lexicographically worse candidates $\boldsymbol{\tau}_2$ and $\boldsymbol{\tau}_3$. (c) Robustness for the rules according to Tab. I.

C. Scenario III

The last scenario we present is `USA_Peach-5_1_T-1`, where an emergency vehicle on duty is approaching an intersection; however, the ego vehicle is blocking its way. Additionally, poor visibility and weather conditions prevail. Fig. 7 shows the results. Our rulebook B implies that not blocking an emergency vehicle on duty is more important than remaining stopped at a red traffic light. Thus, the minimum-violation velocity profile violates the stop-at-red-traffic-light rule φ_{22} and slowly moves into the intersection to avoid interfering with the emergency vehicle. In contrast, velocity profile $\boldsymbol{\tau}_2$ stops and remains stopped at the traffic light and, thus, violates rule φ_6. The resulting order is $\boldsymbol{\tau}_1 > \boldsymbol{\tau}_3 > \boldsymbol{\tau}_2$.

D. Performance

Our algorithm is complete and optimal regarding the set of traces induced by the lattice graph G. It has the same time complexity as the standard A^* algorithm, but additionally, it also depends on the number of rules that must be evaluated for each transition. Tab. II presents the number of evaluated rules N_r and the average computation times T_c for the three scenarios discussed previously. We compare a pure A^* implementation with the adapted approach using a minimum number of rule evaluations A^*_{mre} and find that the number of evaluated rules can be reduced significantly. However, the differences in the calculation times appear to be minor. This is because the number of evaluations for rules on lower ranks (e.g., φ_{31}) can mostly be reduced, which are less costly to evaluate in our implementation than rules on higher ranks (e.g., φ_4) where collision checking is involved. Overall, we believe that the calculation times are reasonable for a prototypical implementation.

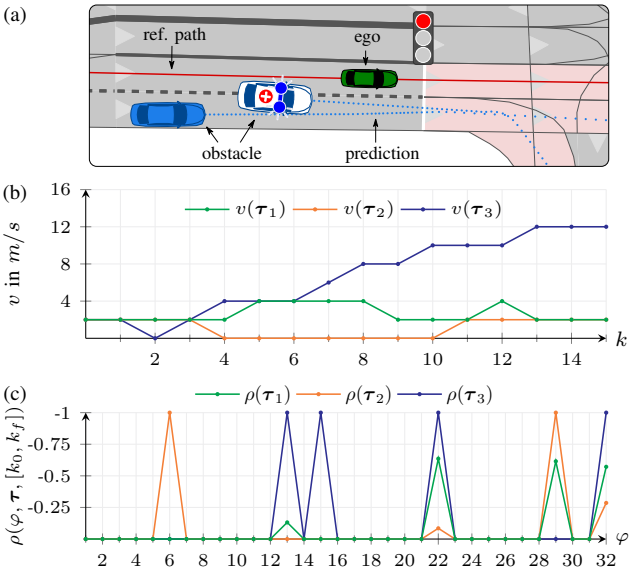

Fig. 7. Scenario III: Blocking of an emergency vehicle on duty at a red traffic light. (a) Start configuration of the scenario. (b) Minimum-violation velocity profile $\boldsymbol{\tau}_1$ and two lexicographically worse candidates $\boldsymbol{\tau}_2$ and $\boldsymbol{\tau}_3$. (c) Robustness for the rules according to Tab. I.

TABLE II
NUMBER OF RULE EVALUATIONS AND AVERAGE COMPUTATION TIMES.

Algo.	Scenario I		Scenario II		Scenario III	
	N_r	T_c (ms)	N_r	T_c (ms)	N_r	T_c (ms)
A^*	13152	106	5376	35	18912	90
A^*_{mre}	9332	94	1978	30	11965	78

VI. CONCLUSIONS

We propose a novel method for minimum-violation velocity planning. Contrary to existing works, our A^*-based velocity planner utilizes rules formalized in STL and uses its quantitative semantics to reason about the level of rule violation of a trajectory over time. We facilitate this by redefining the robustness of the temporal *globally* operator. Furthermore, we show that the overall number of rule evaluations can be reduced. Our experiments demonstrate

that even for scenarios where multiple rules must be violated simultaneously, the resulting velocity profiles remain interpretable and reasonable. Our approach can be beneficial for the certification of autonomous shuttles. In the future, we intend to develop more complex and realistic rules (e.g., based on [5], [6]) and deduce an automatism to derive heuristics, independent of the rule type. As we solely use totally ordered rulebooks in this work, we want to extend our approach to the more general case of preordered rulebooks.

REFERENCES

[1] T. Wongpiromsarn, K. Slutsky, E. Frazzoli, and U. Topcu, "Minimum-violation planning for autonomous systems: Theoretical and practical considerations," in *Proc. of the American Control Conf.*, 2021, pp. 4866–4872.

[2] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[3] R. Boersma, D. Mica, B. Arem, and F. Rieck, "Driverless electric vehicles at businesspark Rivium near Rotterdam (the Netherlands): From operation on dedicated track since 2005 to public roads in 2020," in *Proc. of the Int. Electric Vehicle Symposium & Exhibition, Int. Electric Vehicle Technology Conf.*, 2018.

[4] A. Rizaldi, F. Immler, B. Schürmann, and M. Althoff, "A formally verified motion planner for autonomous vehicles," in *Proc. of the Int. Symposium on Automated Technology for Verification and Analysis*, 2018, pp. 75–90.

[5] S. Maierhofer, A.-K. Rettinger, E. C. Mayer, and M. Althoff, "Formalization of interstate traffic rules in temporal logic," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2020, pp. 752–759.

[6] S. Maierhofer, P. Moosbrugger, and M. Althoff, "Formalization of intersection traffic rules in temporal logic," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2022, pp. 1135–1144.

[7] J. Tůmová, G. C. Hall, S. Karaman, E. Frazzoli, and D. Rus, "Least-violating control strategy synthesis with safety rules," in *Proc. of the Int. Conf. on Hybrid Systems: Computation and Control*, 2013, pp. 1–10.

[8] L. I. Reyes Castro, P. Chaudhari, J. Tůmová, S. Karaman, E. Frazzoli, and D. Rus, "Incremental sampling-based algorithm for minimum-violation motion planning," in *Proc. of the IEEE Conf. on Decision and Control*, 2013, pp. 3217–3224.

[9] J. Rong and N. Luan, "Safe reinforcement learning with policy-guided planning for autonomous driving," in *Proc. of the IEEE Int. Conf. on Mechatronics and Automation*, 2020, pp. 320–326.

[10] J. Tůmová, S. Karaman, C. Belta, and D. Rus, "Least-violating planning in road networks from temporal logic specifications," in *Proc. of the ACM/IEEE Int. Conf. on Cyber-Physical Systems*, 2016, pp. 1–9.

[11] J. Karlsson, C.-I. Vasile, J. Tůmová, S. Karaman, and D. Rus, "Multi-vehicle motion planning for social optimal mobility-on-demand," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2018, pp. 7298–7305.

[12] J. Karlsson and J. Tůmová, "Intention-aware motion planning with road rules," in *Proc. of the IEEE Int. Conf. on Automation Science and Engineering*, 2020, pp. 526–532.

[13] H. Schlüter, P. Schillinger, and M. Bürger, "On the design of penalty structures for minimum-violation LTL motion planning," in *Proc. of the IEEE Conf. on Decision and Control*, 2018, pp. 4153–4158.

[14] C.-I. Vasile, J. Tůmová, S. Karaman, C. Belta, and D. Rus, "Minimum-violation scLTL motion planning for mobility-on-demand," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2017, pp. 1481–1488.

[15] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 2004, pp. 152–166.

[16] A. Censi, K. Slutsky, T. Wongpiromsarn, D. Yershov, S. Pendleton, J. Fu, and E. Frazzoli, "Liability, ethics, and culture-aware behavior specification using rulebooks," in *Proc. of the Int. Conf. on Robotics and Automation*, 2019, pp. 8536–8542.

[17] B. Helou, A. Dusi, A. Collin, N. Mehdipour, Z. Chen, C. Lizarazo, C. Belta, T. Wongpiromsarn, R. D. Tebbens, and O. Beijbom, "The reasonable crowd: Towards evidence-based and interpretable models of driving behavior," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2021, pp. 6708–6715.

[18] L. Niu, J. Fu, and A. Clark, "Minimum violation control synthesis on cyber-physical systems under attacks," in *Proc. of the IEEE Conf. on Decision and Control*, 2018, pp. 262–269.

[19] S. Andersson and D. V. Dimarogonas, "Human in the loop least violating robot control synthesis under metric interval temporal logic specifications," in *Proc. of the European Control Conf.*, 2018, pp. 453–458.

[20] J. Tůmová, L. I. Reyes Castro, S. Karaman, E. Frazzoli, and D. Rus, "Minimum-violation LTL planning with conflicting specifications," in *Proc. of the American Control Conf.*, 2013, pp. 200–205.

[21] M. Lahijanian, S. Almagor, D. Fried, L. E. Kavraki, and M. Y. Vardi, "This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction," in *Proc. of the AAAI Conf. on Artificial Intelligence*, 2015, pp. 3664–3671.

[22] S. Bharadwaj, T. Wongpiromsarn, N. Neogi, J. Muffoletto, and U. Topcu, "Minimum-violation traffic management for urban air mobility," in *Proc. of the NASA Formal Methods Symposium*, 2021, pp. 37–52.

[23] P. Chaudhari, T. Wongpiromsarny, and E. Frazzoli, "Incremental minimum-violation control synthesis for robots interacting with external agents," in *Proc. of the American Control Conf.*, 2014, pp. 1761–1768.

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

[25] J. Karlsson, S. van Waveren, C. Pek, I. Torre, I. Leite, and J. Tůmová, "Encoding human driving styles in motion planning for autonomous vehicles," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2021, pp. 1050–1056.

[26] W. Xiao, N. Mehdipour, A. Collin, A. Bin-Nun, E. Frazzoli, R. D. Tebbens, and C. Belta, "Rule-based optimal control for autonomous driving," in *Proc. of the ACM/IEEE 12th Int. Conf. on Cyber-Physical Systems*, 2021, pp. 143–154.

[27] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, and S. Sankaranarayanan, "Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications," in *Lectures on Runtime Verification: Introductory and Advanced Topics*. Springer, 2018, pp. 135–175.

[28] J. G. Rosenstein, *Linear orderings*. Academic press, 1982.

[29] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and multidisciplinary optimization*, vol. 26, no. 6, pp. 369–395, 2004.

[30] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2017, p. 719–726.

[31] C. Pek and M. Althoff, "Fail-safe motion planning for online verification of autonomous vehicles using convex optimization," *IEEE Transactions on Robotics*, vol. 37, no. 3, pp. 798–814, 2021.

[32] R. Rajamani, *Vehicle Dynamics and Control*. Springer US, 2012.

[33] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[34] Y. Meng, Y. Wu, Q. Gu, and L. Liu, "A decoupled trajectory planning framework based on the integration of lattice searching and convex optimization," *IEEE Access*, vol. 7, pp. 130 530–130 551, 2019.

[35] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, third edition*. The MIT Press, 2009.

[36] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974.

[37] C. Pek, V. Rusinov, S. Manzinger, M. C. Üste, and M. Althoff, "CommonRoad drivability checker: Simplifying the development and validation of motion planning algorithms," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2020, pp. 1013–1020.