MITSUBISHI ELECTRIC RESEARCH LABORATORIES
https://www.merl.com

Autonomous Vehicle Decision-Making and Monitoring based on Signal Temporal Logic and Mixed-Integer Programming

Sahin, Yunus Emre; Quirynen, Rien; Di Cairano, Stefano

TR2020-095 July 03, 2020

Abstract

We propose a decision-making system for automated driving with formal guarantees, synthesized from Signal Temporal Logic (STL) specifications. STL formulae specifying overall and intermediate driving goals and the traffic rules are encoded as mixed-integer inequalities and combined with a simplified vehicle motion model, resulting in a mixed-integer optimization problem. The specification satisfaction for the actual vehicle motion is guaranteed by imposing constraints on the quantitative semantics of STL. For reducing the computational burden, we propose an STL encoding that results in a block-sparse structure. The same STL formulae are used for monitoring faults due to imperfect prediction on the vehicle and environment. We demonstrate our method on an urban scenario with intersections, obstacles, and no-pass zones.

American Control Conference (ACC) 2020

Mitsubishi Electric Research Laboratories, Inc.
201 Broadway, Cambridge, Massachusetts 02139

Autonomous Vehicle Decision-Making and Monitoring based on Signal Temporal Logic and Mixed-Integer Programming

Yunus Emre Sahin[1], Rien Quirynen[2], and Stefano Di Cairano[2]

***Abstract*— We propose a decision-making system for automated driving with formal guarantees, synthesized from Signal Temporal Logic (STL) specifications. STL formulae specifying overall and intermediate driving goals and the traffic rules are encoded as mixed-integer inequalities and combined with a simplified vehicle motion model, resulting in a mixed-integer optimization problem. The specification satisfaction for the actual vehicle motion is guaranteed by imposing constraints on the quantitative semantics of STL. For reducing the computational burden, we propose an STL encoding that results in a block-sparse structure. The same STL formulae are used for monitoring faults due to imperfect prediction on the vehicle and environment. We demonstrate our method on an urban scenario with intersections, obstacles, and no-pass zones.**

I. Introduction

The complexity of autonomous driving and its real-time requirements in resource-limited automotive platforms [1] often impose a decomposition of the guidance and control systems with guarantees that the overall system still satisfies the driving specifications [2], [3], see, e.g., Fig. 1.

The design of a decision-making module that generates intermediate goals for the motion planner according to the vehicle and traffic conditions is challenging since it requires making discrete and continuous decisions for a dynamic system, which operates in a changing environment. Some of the recent works in this area are based on machine learning [4] and on automata combined with set reachability [3]. While the former suffers from the lack of guarantees, the construction of the automaton from natural language specifications of driving goals and traffic rules makes the latter challenging.

In this paper, we take a similar approach to that of [3], but rather than designing an automaton from natural language specifications, we formulate those as Signal Temporal Logic (STL) formulae. These formulae are then converted into a set of mixed-integer inequalities for real-time decision-making and fault monitoring. Solving the resulting mixed-integer programs (MIP) provides a sequence of intermediate goals that satisfy the specifications and are used as waypoints by the motion planner. To reconcile the differences between the simplified model used in decision-making, and the more precise model used in motion planning, we exploit the quantitative semantics of STL and impose a robustness margin on the formulae satisfaction. This robustness is used to ensure that the generated waypoints are feasible, and the motion planner can compute a trajectory to achieve the sequence of goals computed by the decision-maker.

Solving general MIPs is NP-complete, and thus, challenging to do in real-time. However, recent work [5] indicated that, by exploiting the particular structure of mixed-integer optimal control problems, real-time solvers might achieve performance comparable to commercial desktop solvers, such as Gurobi [6]. Therefore, we propose an encoding for the STL formulation that results in a block-sparse mixed-integer problem, for which computing times may be significantly reduced.

Finally, the same STL formulae used for decision-making can be used to monitor whether, due to deviations of the vehicle and the environment from their nominal behavior, the specifications are still met or to trigger possibly asynchronous execution of fault-recovery mechanisms.

The organization of this paper is as follows. In Section II, we introduce modeling, STL, and problem definition. In Section III, we propose an STL encoding that results in a block-sparse MIP. In Section IV, we discuss the implementation of STL-based decision-making and monitoring and the test case, for which simulation results are shown in Section V. Our conclusions are in Section VI.

II. Preliminaries and Problem Definition

In this paper, we consider automated driving, where an autonomous vehicle must reach a desired destination while obeying the traffic rules. This task requires the vehicle to adjust its velocity to obey the speed limits, to avoid collisions, to follow and to change lanes, and to cross intersections following the appropriate right of way rules. The vehicle is equipped with sensors to detect static and dynamic obstacles within a given range and to locate itself in the environment. Furthermore, the vehicle is equipped with a module that provides conservative predictions of the future trajectory of each dynamic obstacle, such that the actual position of the obstacle is always contained in this prediction in the future.

According to the general principles outlined for instance in [3], [7], the guidance and control architecture of the autonomous driving system is divided into several modules, which can be seen from Fig. 1. In this paper, we focus on the following problem formulation.

Problem 1 (Decision Making): Given the navigation information, i.e., the sequence of road segments, the current vehicle state, and the current and predicted location of the obstacles, the decision-making module computes a coarse trajectory, i.e., a sequence of waypoints/goals over a horizon

[1]Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor, MI, USA; ysahin@umich.edu. This work was done while he was an intern with MERL.

[2] Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA, USA; (quirynen, dicairano)@merl.com

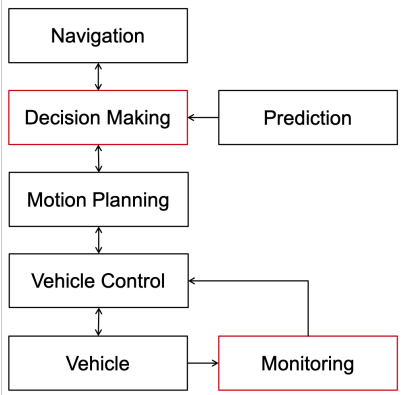

Fig. 1: Control architecture for autonomous driving (red: this paper focus).

of several seconds to be used by the motion planning module, that defines what maneuver the vehicle should perform. □

The sequence of goals provided by the decision-making module is used by the motion planning module to compute a more accurate trajectory that achieves the next goals, over shorter horizons of a few seconds. The trajectory of the motion planning module is then tracked by the vehicle controller. The architecture we envision also includes a monitoring module that verifies at relatively high frequency whether the executed motion is achieving the desired goals. In case of a negative answer, appropriate measures are taken, such as the immediate re-computing of the goals or transitioning to a fault-recovery mode.

We model the ego vehicle motion dynamics by the discrete-time linear (affine) system

$$x_{t+1} = A_t x_t + B_t u_t + a_t, \tag{1}$$

where $x_t \in \mathcal{X}$ is the state vector, $u_t \in \mathcal{U}$ is the control input and a_t is the affine measured disturbance at time t, and the matrices A_t, B_t are possibly time varying yet known ahead of time. The vehicle model (1) is an approximation of more precise models, see, e.g., [8], which are usually nonlinear. However, since we consider only driving in normal conditions, several of the vehicle nonlinearities, such as those in tire force curve, are not excited, while others can be neglected because the decision-making operates over long horizons with a fairly coarse sampling period.

A. Signal Temporal Logic

To describe traffic rules, we use temporal logics, which provide a powerful framework to define system requirements and the dynamic environment in which the system lives. Using temporal logics, natural language specifications can be formulated precisely, without leaving room for different interpretations. This unambiguity enables one to synthesize controllers from high-level specifications using algorithmic techniques. One advantage of such a formal approach is that the solutions are correct-by-construction [9], that is, the trajectories of the closed-loop system are guaranteed to satisfy the temporal logics specifications.

Let $\Pi = \{\pi^{\mu_1}, \pi^{\mu_2}, \ldots\}$ be a given set of predicates, where each $\mu_i : \mathbb{X} \to \mathbb{R}$ maps system states to the reals. An STL formula over Π is defined recursively as follows:

$$\phi := \pi^{\mu} \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \, \mathcal{U}_I \, \phi_2. \tag{2}$$

where symbols $\neg$, $\wedge$, and $\mathcal{U}$ correspond to the logical operators *negation*, *conjunction*, and the temporal operator *until*, respectively. Temporal operator $\mathcal{U}$ (*until*) is equipped with an interval I, indicating the time window in which the STL formula is evaluated [10].

Given a trajectory $\xi = \{x_0, x_1, \ldots\}$, where x_t denotes the state of the system at time step t, satisfaction of an STL formula ϕ can be checked as in [11]. In addition to checking if the formula is satisfied or not, a *robustness function* ρ is used such that ϕ is satisfied by ξ at time t if and only if $\rho(\phi, \xi, t) > 0$. The sign of the robustness score indicates if the specifications are satisfied or violated, whereas the absolute value indicates how strongly did this happen.

While, due to modeling errors and disturbances, perfect tracking of a solution is not possible in real-life scenarios, it is often possible to find bounds on such tracking errors [2]. By ensuring a minimum robustness score, one can prevent such errors to result in a violation of the specifications.

B. Problem Definition

Next, we formally define Problem 1 based on the STL formulae introduced in the previous sections.

Problem 2 (STL-based Decision Making): Given the current vehicle state $x(t)$, the vehicle dynamics (1) and an STL specification ϕ according to (2), find a trajectory $\xi = \{x_0, x_1, \ldots\}$ over a future horizon of N steps that satisfies the vehicle dynamics and enforces ϕ with a robustness score greater or equal than c, while optimizing J, a function of performance metrics $\mathcal{P}$ and a robustness metric $\mathcal{R}$,

$$\min_{u_0,\ldots,u_{N-1}} \quad J(\mathcal{P}, \mathcal{R}) \tag{3a}$$

$$\text{s.t.} \quad x_{t+1} = A_t x_t + B_t u_t + a_t \tag{3b}$$

$$\rho(\phi, \xi, 0) \geq c \tag{3c}$$

$$x_0 = x(t). \tag{3d}$$

The N-step trajectory $\xi^* = \{x_0^*, \ldots, x_N^*\}$, obtained by solving (3), is the sequence of goals, i.e., waypoints, provided to the motion planning according to Fig. 1.

The obstacle predictions are embedded in the STL specification ϕ in (3c). The robustness margin in (3c) for the STL formula satisfaction is particularly useful for at least two reasons. First, it compensates for the approximations in model (1) when the sequence of states, i.e., goals, determined by the decision-making is provided to a motion planner that computes a trajectory over a shorter horizon but using a more precise motion model. Second, it introduces some robustness to the environment, such as the behavior of moving obstacles, which allows for the STL specification to remain satisfied even if the environment prediction is not perfect.

Next, we show how the problem in (3) can be formulated effectively for the numerical solution by MIP solvers.

III. Block-Sparse MIP Formulation of STL Specifications

In this section, we explain how to capture STL specifications as MIP constraints. This framework was first introduced in [12] for LTL specifications and later adapted to STL specifications in [13]. More efficient encodings exist if STL formulae are given in negation normal form [14]. That is, the negation connectives appear immediately before predicates, which are defined by union of polyhedra, i.e., $\mu_j(x_t) > 0 \iff x_t \in \bigcup_{i=1}^{s}\{x : H_i^j x \leq K_i^j\}$. This assumption is not restrictive as any STL formula can be written in this form. However, these methods result in an optimization problem that is generally not block-sparse,

$$\begin{aligned} \min_{Z} \quad & Z^\top \mathbf{H} Z + \mathbf{C}^\top Z \\ \text{s.t.} \quad & \mathbf{G}_i Z \leq K_i \\ & \mathbf{F}_b Z \in \{0,1\}, \end{aligned} \tag{4}$$

due to the STL specifications coupling the decision variables across many, if not all, future prediction steps.

Here, we propose a different encoding that results in a block-sparse structure, which can be exploited by solvers, with the objective of reducing the computational burden that is the most limiting factor of MIPs. In fact, in [5], we have proposed a solver that, despite being entirely custom-made without using any special libraries, was capable of achieving a performance similar to that of commercial high-end solvers, by exploiting the structure of the optimal control problem

$$\begin{aligned} \min_{X,U} \quad & \sum_{t=0}^{N-1} (x_t^\top Q_t x_t + u_t^\top R_t u_t + c_t x_t) + x_N^\top Q_N x_N \\ \text{s.t.} \quad & x_{t+1} = A_t x_t + B_t u_t + a_t \\ & x_0 = x(t) \\ & l_t^c \leq C_t x_t + D_t u_t \leq u_t^c \\ & F_t u_t \in \{0,1\}, \end{aligned} \tag{5}$$

where the optimization variables are the state trajectory $X = [x_0^\top, \dots, x_N^\top]$ and control inputs $U = [u_0^\top, \dots, u_N^\top]$. We refer the interested reader to [5] for more details. Besides being exploitable by custom solvers, even general-purpose commercial solvers can typically take advantage of a block-sparse structure, especially in the pre-solve step that reduces the number of integer variables and their admissible combinations, thus speeding up computations.

Without loss of generality, we further assume that STL specifications are given in negation normal form. Given an STL formula ϕ, we define decision variables u_t^ϕ and x_t^ϕ such that $x_{t+1}^\phi = u_t^\phi$. We ensure through a set of mixed-integer linear constraints that $u_t^\phi = 1$ holds if STL robustness score of ϕ at time t is greater than some predefined threshold, i.e., $\rho(\phi, \xi, t) > c$. The auxiliary states x_t^ϕ are introduced to keep the block-sparse structure in (5).

For each predicate π^μ, u_t^μ is defined as binary, i.e., $u_t^\mu \in \{0,1\}$, and satisfies the constraint $\mu(x_t) + M(1 - u_t^\mu) > c$, where M is a sufficiently large positive number. When $u_t^\mu = 1$, $\rho(\phi, \xi, t) > c$ as expected. If there is a negation operator before a predicate μ, we define a binary variable $u_t^{\neg\mu} \in \{0,1\}$ such that $\mu(x_t) - M(1 - u_t^{\neg\mu}) \leq -c$, so that $u_t^{\neg\mu} = 1$ implies $\rho(\phi, \xi, t) \leq -c$.

Boolean operators are encoded similar to [14]. We define auxiliary inputs $u_t \in [0,1]$ as continuous variables and enforce $\varphi = \bigwedge_i \phi_i \implies u_t^\varphi \leq u_t^{\phi_i}$ for all i and $\varphi = \bigvee_i \phi_i \implies u_t^\varphi \leq \sum_i u_t^{\phi_i}$. For the remainder of this paper, we write $u_t^\varphi = \bigwedge_i u_t^{\phi_i}$ and $u_t^\varphi = \bigvee_i u_t^{\phi_i}$ instead of the corresponding inequalities.

Boolean operators couple decision variables that belong to the same time step. On the other hand, temporal operators couple variables with the same superscript across multiple time steps. For example, the "eventually" operator can be encoded as $\varphi = \Diamond_I \phi \implies u_t^\varphi = \bigvee_{k \in I} u_k^\phi$ as in [13], [14].

To avoid coupling variables across time steps, and to retain the block-sparse structure, we propose a different encoding of the temporal operators as follows. For the "eventually" operator, first we explicitly state $\rho(\Diamond_{[0,0]}\phi, \xi, t) = \rho(\phi, \xi, t)$. Then, we rewrite its quantitative semantics by recursion.

$$\begin{aligned} &\rho(\Diamond_{[a,b]}\phi, \xi, t) = \rho(\Diamond_{[a-1,b-1]}\phi, \xi, t+1), \quad \text{if } a > 0, \\ &\rho(\Diamond_{[a,b]}\phi, \xi, t) = \max\big(\rho(\phi, \xi, t), \\ &\qquad\qquad \rho(\Diamond_{[0,b-1]}\phi, \xi, t+1)\big), \quad \text{if } a = 0. \end{aligned} \tag{6}$$

Equation (6) couples decision variables only if they are at consecutive time steps, instead of over entire time intervals. Exploiting auxiliary states, we can further limit the coupling to variables of the same time step. Let $\varphi = \Diamond_{[a,b]}\phi$, we set

$$\begin{aligned} &x_{t+1}^\varphi = u_{t+1}^{\tilde{\varphi}}, \ \tilde{\varphi} = \Diamond_{[a-1,b-1]}\phi, \quad \text{if } a > 0, \\ &x_{t+1}^\varphi = x_{t+1}^\phi \vee u_{t+1}^{\tilde{\varphi}}, \ \tilde{\varphi} = \Diamond_{[0,b-1]}\phi, \quad \text{if } a = 0, \\ &u_t^\varphi = u_t^\phi, \quad \text{if } a = b = 0, \end{aligned} \tag{7}$$

where continuity conditions $x_{t+1}^\varphi = u_t^\varphi$ and $x_{t+1}^\phi = u_t^\phi$ have been used, and the resulting constraints in (7) can be written as mixed-integer inequalities in the form of (5), allowing us to exploit the block-sparse problem structure.

The encodings for "always" and "until" operators can be derived in a similar way. Let $\varphi = \Box_{[a,b]}\phi$, we set

$$\begin{aligned} &x_{t+1}^\varphi = u_{t+1}^{\tilde{\varphi}}, \ \tilde{\varphi} = \Box_{[a-1,b-1]}\phi, \ \text{if } a > 0, \\ &x_{t+1}^\varphi = x_{t+1}^\phi \wedge u_{t+1}^{\tilde{\varphi}}, \ \tilde{\varphi} = \Box_{[0,b-1]}\phi, \ \text{if } a = 0, \\ &u_t^\varphi = u_t^\phi, \ \text{if } a = b = 0. \end{aligned} \tag{8}$$

Let $\varphi = \phi_1 \mathcal{U}_{[a,b]} \phi_2$, we set

$$\begin{aligned} &x_{t+1}^\varphi = x_{t+1}^{\phi_1} \wedge u_{t+1}^{\tilde{\varphi}}, \ \tilde{\varphi} = \phi_1 \mathcal{U}_{[a-1,b-1]}\phi_2, \quad \text{if } a > 0, \\ &x_{t+1}^\varphi = x_{t+1}^{\phi_2} \vee (x_{t+1}^{\phi_1} \wedge u_{t+1}^{\tilde{\varphi}}), \ \tilde{\varphi} = \phi_1 \mathcal{U}_{[0,b-1]}\phi_2, \ \text{if } a = 0, \\ &u_t^\varphi = u_t^{\phi_2}, \quad \text{if } a = b = 0. \end{aligned} \tag{9}$$

In summary, Eqs. (7)–(9) encode the temporal operators in a way that results in the block-sparse structure (5).

IV. STL-based Decision Making and Monitoring

Next, we describe our STL-based implementation of the decision-making and monitoring systems. The navigation module provides a route that reaches the destination, e.g.,

a sequence of roads and intersections. Based on the route and the prediction of the obstacles, the decision-making module solves an instance of Problem 2 in a receding horizon manner. The decision-making module finds a sequence of waypoints trajectory from one intersection to the next one, while avoiding collisions and obeying traffic rules. Specifically, the decision-making module: (i) represents the driving requirements as the STL specification ϕ; (ii) generates the corresponding mixed-integer constraints based on Section III; (iii) adds the vehicle motion model and a cost function that maximizes the progress to the next waypoint and possibly other soft objectives such as tracking a desired velocity and discouraging large longitudinal accelerations; (iv) solves the MIP problem to obtain a waypoint trajectory part of which is provided to the motion planner.

The vehicle motion is modeled in curvilinear coordinates

$$p_x(t+1) = p_x(t) + v_x(t)\Delta t \tag{10a}$$
$$p_y(t+1) = p_y(t) + v_y(t)\Delta t \tag{10b}$$
$$v_x(t+1) = v_x(t) + a_x(t)\Delta t \tag{10c}$$
$$v_y(t) \leq \alpha\, |v_x(t)|, \tag{10d}$$

where p_x is the longitudinal distance from the reference point, p_y is the lateral deviation from the rightmost lane, v_x is the longitudinal velocity, and the control inputs are the lateral velocity v_y and the longitudinal acceleration a_x. Constraint (10d) is imposed to restrict the range of the vehicle motion, i.e., avoid side movements, so that (10) more closely approximate the real vehicle motion.

We consider the urban driving scenario shown in Fig. 2, where the blue cross is the initial position and the blue rectangle is the destination. The speed of the other vehicles is considerably lower than the ego vehicle to induce overtaking and protracted queuing, the ego vehicle sensor range for detecting position and velocity of other vehicles is 150m.

The traffic rules for the scenario are: *1)* The other vehicles drive according to traffic rules with varying speed, and collisions must be avoided. *2)* The speed limit is 30m/s, reduced to 15m/s in the curved segment. *3)* Lane changing is not allowed within 40m from an intersection and in the upper right curved segment. *4)* The intersections are regulated by a *first-in-first-out* rule. Thus, their STL formulation is

$$\phi = \bigwedge_i \Box(\neg O_t^i) \wedge \tag{11a}$$
$$\bigwedge_i \Box\big(S_i \implies (v_x \in [v_i^{min}, v_i^{max}])\big) \wedge \tag{11b}$$
$$\bigvee_i \Box(\neg L_i^c) \wedge \tag{11c}$$
$$(\neg S_{int}\, \mathcal{U}\, IntersectionClear), \tag{11d}$$

where O_t^i represents a region occupied by the i^{th} obstacle at time t so that Eq (11a) enforces collision avoidance. Eq (11b) enforces speed limits, where the road is partitioned into regions $\{S_i\}$, and the vehicle speed must be in the range $\{[v_i^{min}, v_i^{max}]\}$ in each of those. Eq. (11c) forbids lane changes in certain regions, i.e., if lane changes are not allowed in a region $L = \bigcup_i L_i$, where each L_i represents a different lane, the vehicle must choose one of the lanes and never visit the others. Eq. (11d) ensures that the vehicle comes to a full stop before each intersection and follows the right-of-way rules, where the variable $IntersectionClear$ becomes $true$ only after the vehicle stops and has the right to enter the intersection, and is $false$ otherwise.

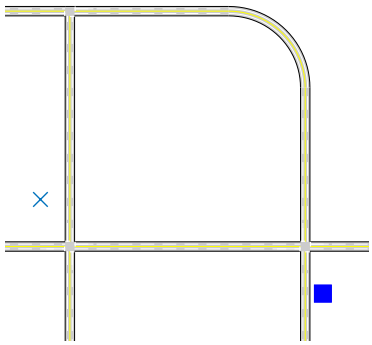

Fig. 2: Setting for the urban driving scenario.

For solving the MIP problems, we use both the commercial desktop solver Gurobi and our structure exploiting branch-and-bound method for mixed-integer model predictive control [5], called BB-PRESAS, where the convex relaxations are solved by the method in [15]. While commercial desktop solvers often achieve higher performance, BB-PRESAS is a self-contained compact solver tailored to execution on embedded platforms such as those for on automotive-grade applications, that have considerably less computational resources and memory than desktop computers. BB-PRESAS can achieve a fast solution by a thoughtful selection of pre-solve techniques, such as bound strengthening and dual fixings, branching strategies, such as reliability branching with pseudo-costs, and tailored warm-starting strategy, such as tree propagation, and structure exploitation.

While our approach guarantees safety by design when the assumptions in the motion models are met, the specifications may be violated due to unexpected conditions leading to wrong predictions of the vehicle or traffic motion. Thus, the architecture in Fig. 1 includes a run-time monitoring based on the same STL encodings that analyzes the trajectory up to the current time, and verifies the satisfaction/violation of the specifications and whether the next waypoint is reachable. Once the trajectory is fixed, for the encodings in Section III, this only requires checking linear inequalities, which is computationally light. Thus, monitoring can operate at high frequency to immediately recognize potential problems.

V. Simulation Results

We simulated the decision-making system in the scenario in Fig. 2, where the path planning is perfectly following the waypoints to better highlight the decision-making behavior. Key frames from the simulated scenario are shown in Fig. 3, where the sampling period for the decision-making is $\Delta t = 2$, the ego vehicle is shown in blue and all the other vehicles are shown in red,. The time histories with respect to the sampling instant t of relevant signals are shown in Fig. 5.

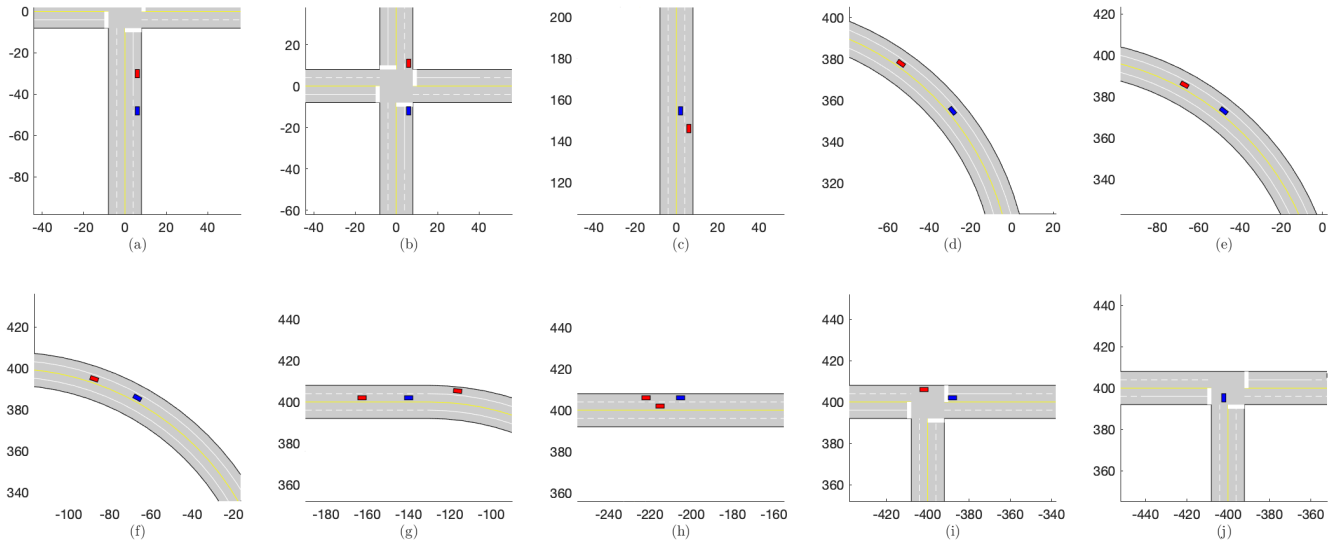

Fig. 3: Important time frames from the simulation results for the decision-making module in the urban driving scenario.

In Fig. 3a, the ego vehicle is behind another vehicle, approaching an intersection, and it keeps a safe distance with the lead vehicle and comes to a full stop before the intersection (Fig. 3b). The ego vehicle reaches to a full stop (0 velocity at $t = 10$), and remains stationary until $IntersectionClear$ signal is $false$, and, once it becomes $true$ (at $t = 12$), it accelerates to cross the intersection, Fig. 5. In Fig. 3c, the ego vehicle is in an area where lane changing is allowed (see, `NoPass` signal in Fig. 5, around step $t = 20$) and it overtakes the leading traffic vehicle, as shown by its lateral position on the road in Fig. 5. The ego vehicle enters the curvy road on the left lane (Fig. 3d) and slows down to meet the 15m/s speed limit (Fig. 5 at around $t = 22$). Around $t = 25$, the ego vehicle approaches a slow lead vehicle (Fig. 3e-3f) and keeps a safe minimum distance by reducing velocity (Fig. 5, where the velocity of the slow lead vehicle is also shown), as lane changing is not allowed in this segment. Once back passing is allowed, the ego vehicle first waits for another vehicle occupying the right lane, before overtaking the leading vehicle (Fig. 3g-3h). The ego vehicle comes to a full stop on the left lane before the intersection (Fig. 3i) and then clears the intersection with a left turn, before reaching the goal (Fig. 3j). As the simulation shows, fairly complex rules can be satisfied by an appropriate formulation as STL specifications, and the MIP approach is effective for synthesizing the decision-making.

As for the computational aspects, Fig. 4 reports solve times for BB-PRESAS and Gurobi for the dense MIQP formulation in (4), i.e., the *Regular* encoding, in a laptop with 2.5 GHz Intel Core i7 and 16 GB RAM where MATLAB is used to interface with the MIP solver. For $N = 5$, the computation time is always well below the decision-making module sampling period of 2s. For $N = 7$, the solver can still compute in less than 2s, but $N > 7$ the computation time increases to above 2s for some steps, especially when the vehicle is near obstacles. Fig. 4 also shows the number

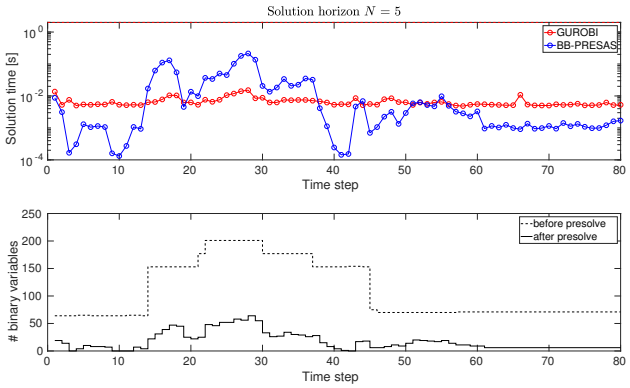

Fig. 4: Top: solve times of BB-PRESAS and Gurobi for the dense MIQP formulation with $N = 5$. Bottom: number of binary variables at each time step before and after the pre-solve procedure for BB-PRESAS.

of binary decision variables before and after BB-PRESAS pre-solve, where the latter number gives a more correct assessment of the actual problem size. Gurobi seems to be less affected by the increase in problem size, possibly due to its more extensive pre-solve heuristics for generating cutting planes. However, there is still a large room for improving the BB-PRESAS for the STL-based decision-making problems, as previously done for optimal control problems [5].

Next, we compare a *Regular* encoding from [14] that solves (4) with our proposed *Block-Sparse* encoding from Section III, which results in solving (5). Solution times using Gurobi as MIQP solver for both formulations and varying horizon N are reported in Fig. 6. For all three values of the horizon N, the *Block-Sparse* encodings have faster solution times, sometimes by more than 1 order of magnitude. As the horizon increases, the solution time increases, but such an increase appears to be larger for the *Regular* encoding. Thus, the *Block-Sparse* encoding seems to be faster and to scale better with the horizon. This is due to the structure of the problem leading to a better reduction of the integer

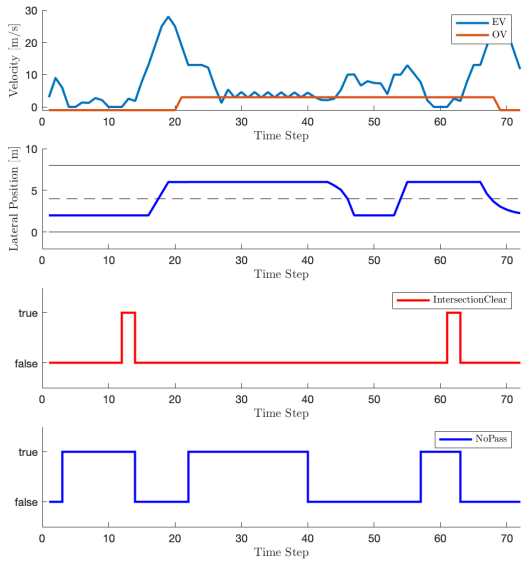

Fig. 5: Time history of relevant signals during the simulation of the urban driving scenario. Top: ego vehicle (EV) velocity, and lead vehicle (OV) velocity (set to -1 when out of range). Mid-high: EV lateral position with respect to the rightmost lane. Mid-low: $IntersectionClear$ signal, $true$ if EV can cross. Bottom: $NoPass$ signal, inhibiting EV lane changing.

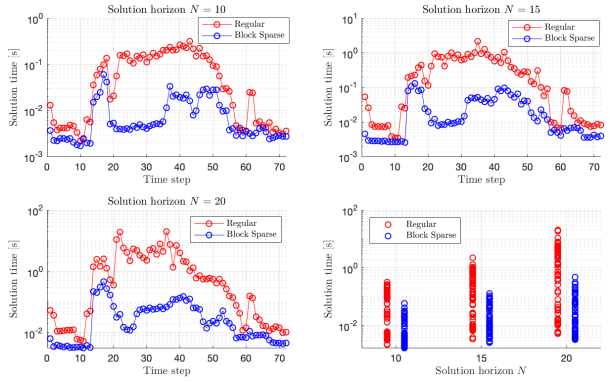

Fig. 6: Timing results for different horizon lengths ($N = 10$, 15 and 20), using Gurobi for the *Regular* and *Block-Sparse* STL encoding formulations.

variables. In fact, Gurobi's pre-solve information show that many more variables are eliminated in the *Block-Sparse* STL encoding of Section III, than in the *Regular* STL encoding.

The monitoring module is evaluated for the scenario in Fig. 2h, where an obstacle behaves unexpectedly. The ego vehicle changes lane assuming the lead vehicle in the target lane to travel at a constant speed. However, from $t = 46$ the lead vehicle suddenly and aggressively brakes, see Fig. 7, which results in the violation of the minimum distance specification. Such fault is detected immediately by the monitoring module which operates at higher frequency than decision-making due to the low computational burden, and emergency braking is initiated timely, avoiding a collision.

VI. Conclusions

We developed decision-making and monitoring for automated driving based on STL specifications, providing waypoints to motion planning and verifying correct execution, respectively. STL specifications, with robustness constraints to reconcile the different models used for decision-making and

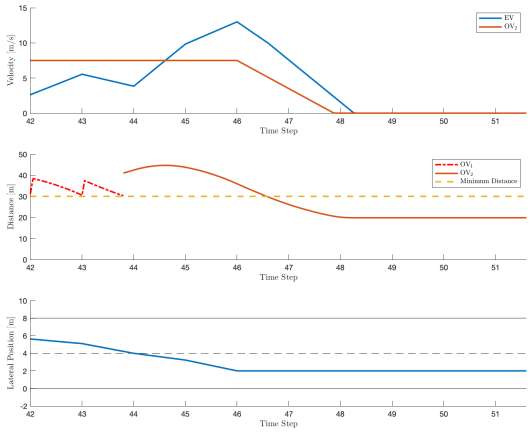

Fig. 7: Simulation of monitoring module detecting minimum distance violation between $t = 46$ and $t = 47$: time history of relevant signals during the simulation of the urban driving scenario.

motion planning, are encoded via mixed-integer constraints. The optimal control problem structure can be retained by a block-sparse STL, which reduces computing time. We demonstrated our approach for both desktop-oriented and embedded platform-oriented MIP solvers.

References

[1] S. Di Cairano and I. V. Kolmanovsky, "Real-time optimization and model predictive control for aerospace and automotive applications," in *American Control Conf.*, pp. 2392–2409, 2018.

[2] S. Di Cairano, U. Kalabić, and K. Berntorp, "Vehicle tracking control on piecewise-clothoidal trajectories by mpc with guaranteed error bounds," in *55th IEEE Conf. Decision and Control*, pp. 709–714, 2016.

[3] H. Ahn, K. Berntorp, and S. Di Cairano, "Reachability-based decision making for city driving," in *Amer. Control Conf.*, pp. 3203–3208, 2018.

[4] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and decision-making for autonomous vehicles," *Annual Review of Control, Robotics, and Autonomous Systems*, 2018.

[5] P. Hespanhol, R. Quirynen, and S. Di Cairano, "A structure exploiting branch-and-bound algorithm for mixed-integer model predictive control," in *European Control Conference (ECC)*, pp. 2763–2768, 2019.

[6] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2018.

[7] K. Berntorp, T. Hoang, R. Quirynen, and S. Di Cairano, "Control architecture design for autonomous vehicles," in *IEEE Conf. Control Technology and Applications*, pp. 404–411, 2018.

[8] R. Quirynen, K. Berntorp, and S. Di Cairano, "Embedded optimization algorithms for steering in autonomous vehicles based on nonlinear model predictive control," in *Amer. Control Conf.*, pp. 3251–3256, 2018.

[9] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Trans. Automatic Control*, vol. 57, no. 11, pp. 2817–2830, 2012.

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

[11] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *International Conference on Formal Modeling and Analysis of Timed Systems*, pp. 92–106, Springer, 2010.

[12] S. Karaman, R. G. Sanfelice, and E. Frazzoli, "Optimal control of mixed logical dynamical systems with linear temporal logic specifications," in *47th IEEE Conf. Dec. and Control*, pp. 2117–2122, 2008.

[13] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *53rd IEEE Conf. Dec. and Control*, pp. 81–87, 2014.

[14] C. Belta and S. Sadraddini, "Formal methods for control synthesis: An optimization perspective," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, pp. 115–140, 2019.

[15] R. Quirynen and S. D. Cairano, "PRESAS: Block-structured preconditioning of iterative solvers within a primal active-set method for fast MPC," *arXiv preprint arXiv:1912.02122*, 2019.