Model Checking Collision Avoidance of Nonlinear Autonomous Vehicles

Rong Gu, Cristina Seceleanu, Eduard Enoiu, Kristina Lundqvist

Mälardalen University, Sweden
firstname.lastname@mdh.se

Abstract. Autonomous vehicles are expected to be able to avoid static and dynamic obstacles automatically, along their way. However, most of the collision-avoidance functionality is not formally verified, which hinders ensuring such systems' safety. In this paper, we introduce formal definitions of the vehicle's movement and trajectory, based on hybrid transition systems. Since formally verifying hybrid systems algorithmically is undecidable, we reduce the verification of nonlinear vehicle behavior to verifying discrete-time vehicle behavior overapproximations. Using this result, we propose a generic approach to formally verify autonomous vehicles with nonlinear behavior against reach-avoid requirements. The approach provides a Uppaal timed-automata model of vehicle behavior, and uses Uppaal STRATEGO for verifying the model with user-programmed libraries of collision-avoidance algorithms. Our experiments show the approach's effectiveness in discovering bugs in a state-of-the-art version of a selected collision-avoidance algorithm, as well as in proving the absence of bugs in the algorithm's improved version.

1 Introduction

Autonomous vehicles (AV), such as driverless cars and robots, are becoming increasingly promising, hence prompting a wide interest in industry and academia. Safety of vehicle operations is the most important concern, requiring these systems to move and act without colliding with static or dynamic objects (obstacles) in the environment, such as big rocks, humans, and other mobile machines. Algorithms like A* [21], Rapidly-exploring Random Tree (RRT) [17], and Theta* [5] are able to navigate the AV towards reaching their destinations, while avoiding static obstacles along the way. However, when encountering dynamic obstacles that could appear and move arbitrarily in the environment, these algorithms are not enough for collision avoidance, and have to be complemented by algorithms such as those based on dipole flow fields [23] or dynamic window approach [10], which are capable of circumventing dynamic obstacles.

Although many collision-avoidance algorithms are being proposed in recent years, few of them have been formally verified, despite the fact that formal verification is a very important tool for discovering problems in the early stage of algorithm design. In this paper, we consider two main challenges that can turn formal verification of AV models and their algorithms into a daunting task: (i)

nonlinearity of the vehicle kinematics, and (ii) complexity and uncertainty of the environment where AV move. On the one hand, ordinary differential equations are used to describe the continuous dynamics and kinematics of the often nonlinear vehicles. The trajectories formed by these vehicle models are consequently nonlinear, which is the nonlinearity that we consider throughout the paper. On the other hand, discrete decisions made by the vehicles' control systems influence the movement of vehicles. In the model-checking world, verification of these so-called *nonlinear hybrid systems* that combine nonlinear continuous kinematics and discrete control is undecidable [13, 15]. In addition, AV that aim at tracking initially planned paths are inevitably diverted by their tracking errors caused by the inaccuracy of their sensors and actuators, and the disturbance from the complex environment. Dynamic obstacles are unpredictable before AV sense them. All these reasons render exhaustive model checking of models of nonlinear vehicles that move in an environment containing static obstacles and uncertain dynamic obstacles an unsolved problem.

In this paper, we solve this problem by addressing challenges (i) and (ii). First, we introduce *safe zones* of the trajectories formed by *controllable* nonlinear AV models, which overcomes challenge (i), as follows. If an AV's tracking error has a Lyapunov function, it is called *controllable* in this paper, and its deviation from the reference path is bounded [8]. The boundaries of tracking errors form the safe zone of the AV, assuming the reference path as the axis. As long as the dynamic obstacles do not intrude into these zones, the vehicles are guaranteed to be safe. Based on this observation, we reduce the verification of *controllable* AV's nonlinear trajectories to the verification of its piece-wise-continuous (PWC) reference trajectories, and further to the verification of discrete-time models of trajectories. The various vehicle dynamics and kinematics, together with the uncertain tracking errors are all subsumed by the safe zones, so the undecidable verification problem is simplified to a decidable one, without losing completeness.

Next, we solve challenge (ii) by leveraging the nondeterminism of timed automata in Uppaal STRATEGO [6]. The initialization and movement of dynamic obstacles are modeled as timed automata, in which their positions etc. are nondeterministically initialized and updated. In this way, the vehicle model satisfies the *liveness* property only when it is able to reach the destination, and the *invariance* property if there is no collision happening under any circumstance. When multiple dynamic obstacles are involved, the state space of the model becomes large and the verification becomes computationally expensive or even unsolvable. Consequently, we also propose a way of reducing the state space by splitting the verification into multiple tractable phases.

Note that, our approach is orthogonal to the methods of controller synthesis (e.g., [7, 9]). The latter targets the construction of motion plans that avoid static and dynamic obstacles, whereas our method can be used to verify the correctness of these methods, regardless of the path-planning and collision-avoidance algorithms considered. To summarize, our main contributions are:

1. A proven transformation of the verification of nonlinear vehicle trajectories to the verification of PWC trajectories and discrete-time trajectories.

2. A generic verification approach for model checking reach-avoid requirements of AV equipped with different collision-avoidance algorithms (Section 4).
3. An implementation of the approach in UPPAAL STRATEGO, and a demonstration showing the ability of the approach to discover bugs in a state-of-the-art collision-avoidance algorithm, and to prove the absence of bugs in an improved version of the same algorithm (Section 5).

Preliminaries. In this paper, we denote a vector x by $\vec{x}$, the module of $\vec{x}$ by $\|\vec{x}\|$, and multiplications between two scalars, and between a vector and a scalar by "×". *Timed Automata* is a widely-used formalism for modeling real-time systems [4]. The UPPAAL model checker [16] uses an extension of the timed-automata language with a number of features such as constants, data variables, arithmetic operations, arrays, broadcast channels, urgent and committed locations. Properties that can be checked by UPPAAL are formalized in a simplified *timed computation tree logic* (TCTL) [3], which basically contains a decidable subset of *computation tree logic* (CTL) plus clock constraints. A branch of UPPAAL, named UPPAAL STRATEGO [6], supports calling external C-code functions written in libraries. This new feature enables us to treat the user-designed collision-avoidance algorithm as a black box in our model.

The remainder of the paper is organized as follows. In Section 2, we introduce the systems to be verified. In Section 3, we concretely define the movement and trajectories of AV and prove two theorems of transforming the verification of nonlinear vehicle trajectories to the verification of PWC trajectories and discrete-time trajectories. A detailed description of the verification approach and tool support is presented in Section 4, followed by experiments in Section 5. We compare our study to related work in Section 6, and conclude the paper in Section 7.

2 Problem Description

Vehicles that are capable to calculate paths to their destinations, which avoid collision with any obstacles in the environment, and follow them without human intervention, are called *autonomous vehicles* (AV). As depicted in Fig. 1, when the environment contains only static obstacles whose positions are already known by the AV, paths are calculated by the path planner inside the controller of the AV. Path planners are usually equipped with path-planning algorithms, e.g.,

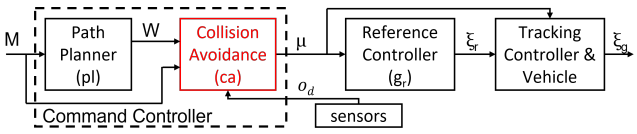


Fig. 1. The architecture of the controller of autonomous vehicles. The collision-avoidance module does not exist if the environment only contains static obstacles.

Theta* [5] or RRT [17], which explore the map (M) to find a path that avoids the static obstacles and reaches the destination. The reference controller (g_r) uses the output of the path planner and generates a trajectory of the state variables

of the system, e.g., position and linear velocity of the vehicle, as a reference (ξ_r) for the tracking controller to follow. The tracking controller aims to produce an input to the vehicle to drive it to track the reference trajectory. The real trajectory (ξ_g) follows the reference path (ξ_r) with some tracking errors.

Since the dynamics and kinematics of a real AV are nonlinear, and tracking errors between the actual trajectory and reference trajectory exist inevitably, path planners do not guarantee the safety of AV driving. Moreover, formally verifying if the actual trajectories ever hit the static obstacles is an undecidable problem, due to the model-checking of nonlinear hybrid systems being undecidable [15]. Overapproximation is a method of linearizing the vehicle model, to facilitate verification. Fan et al. [8] propose a method that proves that, as long as the dynamics of tracking errors has a Lyapunov function, the tracking errors are bounded by a piece-wise constant value, which depends on the initial tracking error and the number of segments of the reference trajectory. Fig. 2 shows an example of a reference trajectory and the boundary of tracking errors. Consequently, as long as the safe regions of AV (green color) do not overlap with the grey areas, the actual trajectory is guaranteed to be safe.

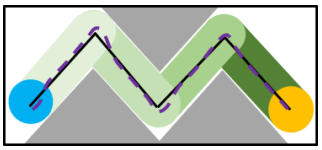


Fig. 2. The reference trajectory is solid black lines, and the actual trajectory is violet dotted lines. The initial area is blue and the goal area is yellow. The boundaries of tracking errors are green. Static obstacles are grey [8].

Due to this result, one can reduce the problem of verifying whether the actual trajectory (ξ_g) ever overlaps with obstacles, to a simplified problem of verifying whether the distance between the reference trajectory (ξ_r) and the obstacles is larger than the respective boundary of tracking error on each segment of ξ_r. In other words, the verification of nonlinear vehicle trajectories is reduced to the verification of their piece-wise-continuous reference trajectories. Although much simplified, the problem is still undecidable as long as the piece-wise-continuous trajectories are non-linear [8]. Moreover, when dynamic obstacles appear, the verification becomes intractable, because dynamic obstacles cannot be known completely before the AV encounters them. The controller must be additionally equipped with a collision-avoidance module that perceives the environment periodically, via sensors. Fig. 1 shows such a controller. The path planner still calculates a path that avoids known static obstacles and goes to the destination. The path serves as input to the collision-avoidance module as a sequence of waypoints (positions of turning directions, denoted as W), as well as the information of the map (M) and dynamic obstacles (o_d). The command controller should meet the following two requirements, which are the focus of verification in this paper:

- Collision avoidance (invariance property): always circumventing the static and dynamic obstacles;
- Destination reaching (liveness property): always eventually reaching the goal area.

3 Definitions and Verification Reduction Theorems

In this section, we introduce the definitions of the important concepts used in this paper and the collision-avoidance verification theorems that eventually reduce the nonlinear trajectory verification to discrete-time trajectory verification. We denote AV and dynamic obstacles collectively by the term *agents*.

First, let us establish an overall view of the different types of models that are used in this section. So far, we have stated that model-checking *liveness* properties (e.g., destination reaching) and *invariance* properties (e.g., collision avoidance) of nonlinear hybrid systems is undecidable. Note that hybrid systems are described by syntactic models with an underlying semantics defined as hybrid transition systems (HTS), used in the following definitions. As depicted in Fig. 3, the continuous trajectories of agents are modeled as HTS. By incorporating the tracking errors of agents, the continuous trajectories are simplified into piece-wise-continuous (PWC) trajectories. However, the verification of PWC trajectories is still undecidable, so we transform the PWC trajectories into discrete-time trajectories, whose verification is decidable. Furthermore, the two-step transformation from continuous trajectories to discrete-time trajectories is proved to preserve the *liveness* and *invariance* properties that we want to verify (Theorem 1 and Theorem 2).

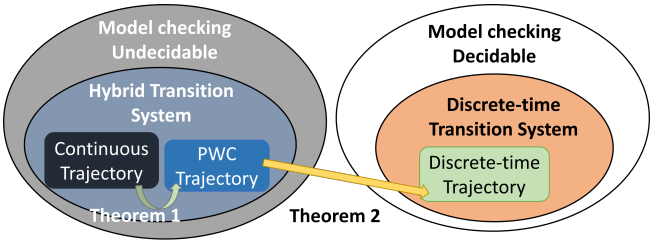


Fig. 3. Overall description of models and their decidability

3.1 Definitions of Maps, Agent States, and Trajectories

In this section, we first define the agent states and the map where agents move. Next, we define the command controllers and agent-state trajectories.

Definition 1 (Map). *A map is a 4-tuple* $\mathcal{M} =< \mathcal{X}, \boldsymbol{O}_u, \mathcal{I}, \mathcal{G} >$*, where (i)* $\mathcal{X} \in \mathbb{R}^d$ *is the moving space, with* $d \in \{2,3\}$ *being the dimension of the map, (ii)* $\boldsymbol{O}_u \subseteq \mathcal{X}$ *is the unsafe area, (iii)* $\mathcal{I} \subseteq \mathcal{X}$ *is the initial area of AV, and (iv)* $\mathcal{G} \subseteq \mathcal{X}$ *is the goal area where the AV aims to go.*

An example of a map is illustrated in Fig. 2.

Definition 2 (Agent State). *Given a map* $\mathcal{M} =< \mathcal{X}, \boldsymbol{O}_u, \mathcal{I}, \mathcal{G} >$*, an agent state is a 5-tuple* $\mathcal{S} =< \vec{p}, \vec{v}, \vec{a}, \theta, \omega >$*, where (i)* $\vec{p} \in \mathcal{X}$ *is the position vector, (ii)* $\vec{v}$ *is the linear velocity vector,* $\|\vec{v}\| \in [0, V_{max}] \subset \mathbb{R}_{\geq 0}$*, (iii)* $\vec{a}$ *is the acceleration vector,* $\|\vec{a}\| \in [A_{min}, A_{max}] \subset \mathbb{R}$*, (iv)* $\theta \in [-\pi, \pi] \subset \mathbb{R}$ *is the heading, and (v)* $\omega \in [\Omega_{min}, \Omega_{max}] \subset \mathbb{R}$ *is the rotational velocity.*

The agent states are states of AV and dynamic obstacles. Some elements in the tuple of agent states $\mathcal{S}$ evolve continuously and some are assumed to change instantaneously. We define the trajectories of the evolution of the agent states in Definition 4. Before that, we first define the controller of AV, where dynamic obstacles ($\mathbf{O}_d$) are instances of agent states $\mathcal{S}$, as follows:

Definition 3 (Controller). *Given a map $\mathcal{M}$, and a set of dynamic obstacles $\boldsymbol{O}_d$, we define a command controller of AV as a 3-tuple $\mathcal{C} =< pl, ca, \Lambda >$, where (i) $pl : \mathcal{M} \longrightarrow \mathcal{W}$ is a path-planning function, $\mathcal{W} \subseteq \mathcal{X}$ is a set of waypoints, (ii) $ca : \mathcal{M} \times \mathcal{W} \times \boldsymbol{O}_d \longrightarrow \Lambda$ is a collision-avoidance function, and (iii) $\Lambda = \{ACC, BRK, TR^+, TR^-, STR\}$ is a set of commands.*

The commands are signals sent from the controllers to the actuators of the AV: ACC means acceleration, BRK means brake, TR^+ and TR^- mean turning counter-clockwise and clockwise, respectively, and STR means moving straightly at a constant speed. An example of the AV's controller architecture is shown in Fig. 1. When an AV starts to move, the transitions of its agent states form a trajectory, in which its position, linear velocity, and heading evolve continuously according to corresponding dynamic functions, whereas its acceleration and rotational velocity change discretely based on the commands.

Definition 4 (Continuous Trajectory). *Given an AV, whose command controller is $\mathcal{C} =< pl, ca, \Lambda >$, we define its movement by a hybrid transition system $< S, s_0, \Sigma, X, \rightarrow >$, where S is a set of states, s_0 is the initial state, $\Sigma \subseteq \Lambda$ is the alphabet, $X = X_d \cup X_c$ is a set of variables combining discrete variables in X_d and continuous variables in X_c, and $\rightarrow$ is a set of transitions defined by the following rules, with kinematic functions of the AV denoted by f:*

- *Delayed transitions: $< \vec{p}, \vec{v}, \vec{a}, \theta, \omega > \xrightarrow{\Delta t} < \vec{p}', \vec{v}', \vec{a}', \theta', \omega' >$, where $t \in X_c$, $\vec{p}' = \vec{p} + \int_l^u \vec{v} dt$, $\vec{v}' = \vec{v} + \int_l^u \vec{a} dt$, $\vec{a}' = \vec{a}$, $\theta' = \theta + \int_l^u \omega dt$, $\omega' = \omega$, $l \in \mathbb{R}_{\geq 0}$ and $u \in \mathbb{R}_{>0}$ are the upper and lower time bounds, respectively, and $\Delta t = u - l$;*
- *Instantaneous transitions: $< \vec{p}, \vec{v}, \vec{a}, \theta, \omega > \xrightarrow{cmd} < \vec{p}', \vec{v}', \vec{a}', \theta', \omega' >$, where $\vec{p}' = \vec{p}$, $\vec{v}' = \vec{v}$, $\vec{a}' = ca(\vec{a}, cmd)$, $\theta' = \theta$, $\omega' = ca(\omega, cmd)$, $cmd \in \Sigma$.*

A run of the transition system defined above over a duration U is a *trajectory* of agent states, also described by the function $\xi : [0, U] \rightarrow \mathcal{S}$. Henceforth, we name the agent-state trajectory as *trajectory* for brevity, and denote $\xi(t)$ as a point of ξ at time t, the projection of ξ on a dimension of an agent-state as $\xi{\downarrow}dimension$, e.g., positions on a trajectory are $\xi{\downarrow}\vec{p}$. The continuous variables of actual trajectories of agents are generated by their nonlinear kinematic functions, yet these variables are piece-wise-continuous (PWC) in reference trajectories (see Figure 2). More specific, a reference trajectory ξ_r is a sequence of concatenated trajectory segments $\xi_{r,1} \frown ... \frown \xi_{r,k}$. The concatenating points $\{\vec{p}_i\}_{i=0}^k$ are the waypoints calculated by path-planners, where the discontinuity of the vehicle's heading θ happens. Therefore, the definition of agent movement on a reference trajectory changes as follows:

Definition 5 (Reference Trajectory). *Let us assume an AV, whose command controller is $\mathcal{C} =< pl, ca, \Lambda >$, and a PWC trajectory ξ_r of the AV, which*

is a sequence of trajectories $\xi_{r,1} \frown ... \frown \xi_{r,k}$ *concatenated by a set of waypoints* $\{\vec{P}_i\}_{i=0}^{k}$*. Then, the AV's movement along the reference trajectory is a hybrid transition system similar to that of Definition 4, and its transitions are defined by the following rules:*

- *Delayed transitions on* $\xi_r{\downarrow}\vec{p} \notin \{\vec{P}_i\}_{i=0}^{k}$*:* $< \vec{p}, \vec{v}, \vec{a}, \theta, \omega > \xrightarrow{\Delta t} < \vec{p}', \vec{v}', \vec{a}', \theta', \omega' >$*, where* $\vec{p}' = \vec{p} + (\vec{v} + \frac{\vec{a} \times \Delta t}{2}) \times \Delta t$*,* $\vec{v}' = \vec{v} + \vec{a} \times \Delta t$*,* $\vec{a}' = \vec{a}$*,* $\theta' = \theta$*,* $\omega' = 0$*;*
- *Instantaneous transitions:* $< \vec{p}, \vec{v}, \vec{a}, \theta, \omega > \xrightarrow{cmd} < \vec{p}', \vec{v}', \vec{a}', \theta', \omega' >$*, where* $\vec{p}' = \vec{p}$*,* $\vec{v}' = \vec{v}$*,* $\vec{a}' = ca(\vec{a}, cmd)$*,* $\theta' = \begin{cases} arctangent(\vec{P}_i, \vec{P}_{i+1}), & \text{if } \vec{p} \in \{\vec{P}_i\}_{i=0}^{k-1} \\ \theta, & \text{if } \vec{p} \notin \{\vec{P}_i\}_{i=0}^{k-1} \end{cases}$*,* $\omega' = 0$

Intuitively, when an agent is moving along its reference trajectory (ξ_r), its heading ($\xi_r{\downarrow}\theta$) remains unchanged before it arrives at a waypoint, which means the rotational velocity ($\xi_r{\downarrow}\omega$) is irrelevant and remains 0. Therefore, the reference trajectory is infeasible to be tracked exactly by the agents. Although the integration of $\xi_r{\downarrow}\vec{p}$ and $\xi_r{\downarrow}\vec{v}$ on delayed transitions is simplified to polynomial functions, the nonlinearity of $\xi_r{\downarrow}\vec{p}$ still renders undecidability. The trigonometric function in the definition also causes a computational difficulty when running verification. In practice, we use linear speed vector ($\vec{v}$) to describe both the linear speed and the orientation of the agent. The acceleration ($\xi_r{\downarrow}\vec{a}$) changes instantaneously based on the commands from the command controller. Last but not least, the trajectories of dynamic obstacles are similar to Definition 4, but without a well-defined controller. On their instantaneous transitions, accelerations and rotational velocities are changed arbitrarily within the valid ranges.

3.2 Collision-Avoidance Verification Reduction

We use ξ_r and ξ_g to denote the reference and actual trajectory of AV, respectively, and ξ_o for the actual trajectories of dynamic obstacles.

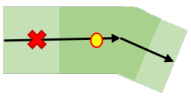


Fig. 4. A dynamic obstacle is at the red cross, while the current position of AV on the reference path is the yellow dot. The safety-critical area is dark green.

Let $d(var_1, var_2)$ denote the distance between var_1 and var_2, e.g., $d(\vec{p}_i, \xi_j{\downarrow}\vec{p})$ is the distance from position $\vec{p}_i$ to trajectory $\xi_j{\downarrow}\vec{p}$, and $d(\xi_i{\downarrow}\vec{p}, \mathbf{O}_u)$ is the distance from trajectory $\xi_i{\downarrow}\vec{p}$ to static obstacles. For brevity, we omit the projection when using this notation, i.e., $d(\vec{p}_i, \xi_j{\downarrow}\vec{p}) = d(\vec{p}_i, \xi_j)$. Let $\xi(t_1, t_2)$ denote a segment of trajectory ξ between time points t_1 and t_2. The problem of verifying if AV hit static obstacles $\mathbf{O}_u$ is relatively simple, as $\mathbf{O}_u$ does not change. However, checking if AV hit moving obstacles is different and much harder, because both trajectories are formed dynamically while the agents are moving. Dynamic obstacles might meet an AV's reference trajectory, yet far enough from its current position(see Fig. 4). Therefore, we introduce the concept of *safety-critical segments*:

Definition 6 (Safety-Critical Segment). *Let* C *be the current time. Given a trajectory* ξ*, a time span of length* $T \in \mathbb{R}_{>0}$*, we define a safety-critical segment* $sc(\xi)$ *of* ξ*, as* $\xi(C - T, C + T)$[1]*.*

[1] When $C < T$, $sc(\xi) = \xi(0, C + T)$.

The length of time-span T, so that the safety-critical area covers the actual current position of AV, can be delivered by design engineers with knowledge of vehicle dynamics, so this is not within the scope of this paper. Now, instead of checking if any part of the AV's entire trajectory (ξ_g) overlaps with a moving obstacle's trajectory (ξ_o), we check if the safety-critical segments of these two trajectories ($sc(\xi_g)$ and $sc(\xi_o)$) overlap.

Definition 7 (Collision-Avoidance Verification). *Given a map $\mathcal{M} =< \mathcal{X}, \boldsymbol{O}_u, \mathcal{I}, \mathcal{G} >$, a nonlinear AV, whose actual continuous trajectory is ξ_g, and a set of dynamic obstacles whose trajectories are in set Ξ_o, we say that the collision-avoidance verification of the AV's actual trajectory equates with verifying that condition $\xi_g{\downarrow}\vec{p} \cap \mathcal{G} \neq \varnothing \wedge \xi_g{\downarrow}\vec{p} \cap \boldsymbol{O}_u = \varnothing \wedge sc(\xi_g{\downarrow}\vec{p}) \cap sc(\xi_o{\downarrow}\vec{p}) = \varnothing$ holds, where $\xi_o \in \Xi_o$.*

Since model-checking ξ_g is undecidable, we prove next that its verification can be reduced to one over the PWC trajectory ξ_r that ξ_g tracks.

Theorem 1 (Non-linearity to PWC). *Assume the collision-avoidance verification condition of Definition 7, a position $\vec{p}_g \in \mathcal{G}$ whose distance to the closest boundary of $\mathcal{G}$ is B, and that the tracking errors of the AV have a Lyapunov function. Then, it follows that if the condition $\xi_r{\downarrow}\vec{p} \cap \{\vec{p}_g\} \neq \varnothing \wedge d(\xi_r, \boldsymbol{O}_u) > L \wedge d(sc(\xi_r), sc(\xi_o)) > L$, with $L \in \mathbb{R}_{>0}$ and $L \leq B$ holds, then the collision-avoidance condition of Definition 7 holds too.*

Proof. Based on Lemmas 2 and 3 proven by Fan et al. [8], if the tracking errors of the AV have a Lyapunov function, its ξ_g is bounded within a certain distance to its ξ_r. Let the distance be L, then $d(\xi_g, \xi_r) < L \leq B$. Hence, if $\xi_r{\downarrow}\vec{p} \cap \{\vec{p}_g\} \neq \varnothing$, then $\xi_g{\downarrow}\vec{p} \cap \mathcal{G} \neq \varnothing$. Since $d(\xi_r, \mathbf{O}_u) > L > d(\xi_g, \xi_r)$ and $d(sc(\xi_r), sc(\xi_o)) > L > d(\xi_g, \xi_r)$, then $\xi_g{\downarrow}\vec{p} \cap \mathbf{O}_u = \varnothing \wedge sc(\xi_g{\downarrow}\vec{p}) \cap sc(\xi_o{\downarrow}\vec{p}) = \varnothing$. □

Note that these two problems are not equivalent. When the actual trajectory is not colliding with any obstacles, the distance from the reference trajectory to the obstacles could be less than L. The method of calculating L is not the concern of this paper. We refer the reader to literature [8] for details.

3.3 Discretization of Trajectories

Although the verification of nonlinear trajectories is simplified by Theorem 1, model-checking PWC trajectories is still difficult. PWC trajectories are described by hybrid systems, in which variables, e.g., $\vec{p}$ and $\vec{v}$, change continuously (specifically, $\vec{p}$ is nonlinear), whereas variables, e.g., θ, $\vec{a}$ and ω, change instantaneously (Definition 5). Unfortunately, the algorithmic verification of such model is undecidable [20]. To make the problem tractable, we discretize PWC trajectories into a discrete-time model, where the movement of agents (including AV and dynamic obstacles) is sampled synchronously:

Definition 8 (Discrete-Time Trajectory). *Given a PWC trajectory named ξ_r, whose concatenating points (waypoints) are $\{\vec{P}_i\}_{i=0}^{k}$, a discretized trajectory ξ_{rd} of ξ_r is a run of a corresponding discrete-time transition system <*

$D, d_0, \Pi, \rightarrow>$, where D is the set of states, d_0 is the initial state, $\Pi \subseteq \Lambda \cup \{sync\}$ is the set of labels consisting of controller commands and a label for synchronization with other discretized trajectories, and $\rightarrow$ is a transition relation, in which the instantaneous transitions of $\theta, \vec{a}$ and ω remain the same as defined in Definition 5, and the delayed transitions are sampled at the time points when $\Delta t = \varepsilon$, where $\varepsilon \in \mathbb{R}_{>0}$ is the granularity of sampling:

- *if $\Delta t < \varepsilon$, $< \vec{p}, \vec{v}, \vec{a}, \theta, \omega >$ does not change,*
- *if $\Delta t = \varepsilon$, $< \vec{p}, \vec{v}, \vec{a}, \theta, \omega > \xrightarrow{\Delta t, sync} < \vec{p}', \vec{v}', \vec{a}', \theta', \omega' >$, where $\theta' = \theta, \omega' = \omega, \vec{a}' = \vec{a}, \vec{v}' = \begin{cases} \vec{v} + \vec{a} \times \varepsilon, & \text{if } \|\vec{v} + \vec{a} \times \varepsilon\| < V_{max}, \\ \frac{\vec{v}}{\|\vec{v}\|} \times V_{max}, & \text{if } \|\vec{v} + \vec{a} \times \varepsilon\| \geq V_{max} \end{cases}, \vec{p}' = \begin{cases} \vec{P}_i, & \text{if } \vec{p} + (\vec{v} + \frac{\vec{a} \times \varepsilon}{2}) \times \varepsilon \succ \vec{P}_i, \\ \vec{p} + (\vec{v} + \frac{\vec{a} \times \varepsilon}{2}) \times \varepsilon, & \\ \quad \text{if } \vec{p} + (\vec{v} + \frac{\vec{a} \times \varepsilon}{2}) \times \varepsilon \preccurlyeq \vec{P}_i \end{cases}$*

To denote if the position passes (resp., does not pass) the next waypoint, we use the syntactic sugar $\succ$ (resp., $\preccurlyeq$). The algorithm of judging this is given in literature [12]. Intuitively, when the time interval Δt is less than a small period ε, the environment is not observed, so the trajectories of the agents are not sampled; when Δt reaches ε, the agent states are observed and sampled. When an agent reaches or passes its target waypoint in the current period ε, it stops at the waypoint until the next period comes when the new waypoint and heading are updated by the instantaneous transitions.

Dynamic obstacles do not have pre-computed waypoints but appear and move arbitrarily in the map. However, a reasonable obstacle would not change its direction too frequently, e.g., every sampling period. We design dynamic obstacles such that, initially, they choose their starting agent-states arbitrarily. Then, they keep moving for N sampling periods before choosing a new agent-state as a target. The straight path between the current and target positions is a reference trajectory that the dynamic obstacle tracks in the next N periods, and the tracking errors are also bounded.

The agents' accelerations and rotational velocities are assumed to be changing discretely in these definitions. If the assumption is violated in some applications, one can discretize these two variables in the same way as in the discretization of position and linear velocity. Next, we prove a theorem that reduces the verification of PWC reference trajectories to the one of discrete-time trajectories.

Theorem 2. *(PWC to discrete-time trajectories). Assume a map $\mathcal{M} =< \mathcal{X}, \boldsymbol{O}_u, \mathcal{I}, \mathcal{G} >$, a set of trajectories Ξ_o formed by dynamic obstacles, with the maximum linear velocity V, a reference trajectory ξ_r of an AV with concatenating points $\{\vec{P}_i\}_{i=0}^k$, whose safety-critical segment is $sc(\xi_r)$, and synchronized and discretized trajectories ξ_{rd} of ξ_r, and ξ_{od} of $\xi_o \in \Xi_o$ with a granularity of sampling $\varepsilon \leq \frac{L}{\|V\|}$; here, $L = L_a + L_o$, where L_a is the tracking-error boundary of the AV, and L_o is the smallest tracking-error boundary among dynamic obstacles[2]. Then, if $\vec{p}_g \in \mathcal{G}$, and $\xi_{rd} \downarrow \vec{p} \cap \{\vec{p}_g\} \neq \varnothing \wedge d(\xi_{rd}, \boldsymbol{O}_u) > L \wedge d(sc(\xi_{od}), sc(\xi_r)) > L$, it follows that $\xi_r \downarrow \vec{p} \cap \{\vec{p}_g\} \neq \varnothing \wedge d(\xi_r, \boldsymbol{O}_u) > L \wedge d(sc(\xi_o), sc(\xi_r)) > L$.*

[2] When no dynamic obstacle is detected, L_o is zero.

Proof. By substituting Δt in the delay transitions of Definition 5 with ε, we can see that $\xi_{rd}(\varepsilon)$ is a sampling of the reference trajectory $\xi_r(t)$ at the time points when $\Delta t = \varepsilon$. Hence, $\xi_{rd}{\downarrow}\vec{p} \subseteq \xi_r{\downarrow}\vec{p}$. Therefore, if $\xi_{rd}{\downarrow}\vec{p} \cap \{\vec{p}_g\} \neq \varnothing$, which means ξ_{rd} can reach $\vec{p}_g$, then $\xi_r{\downarrow}\vec{p} \cap \{\vec{p}_g\} \neq \varnothing$ as well.

Based on Definition 8, waypoints $\{\vec{P}_i\}_{i=0}^{k} \subseteq \xi_{rd}{\downarrow}\vec{p}$, where turning occurs. Therefore, if t_i and t_{i+1} are two consecutive sampling points of ξ_{rd}, the line segment connecting t_i and t_{i+1} must be on ξ_r, denoted by $\xi_{rd}(t_i, t_{i+1})$. Therefore, if $d(\mathbf{O}_u, \xi_{rd}(t_i, t_{i+1})) > L$[3], then the concatenation of $\{\xi_{rd}(t_i, t_{i+1})\}_{i=0}^{n-1}$, which is ξ_r, satisfies $d(\mathbf{O}_u, \xi_r) > L$.

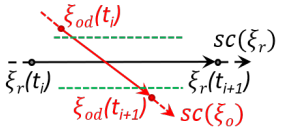


Fig. 5. The trajectory of a dynamic obstacle is red. The reference trajectory of AV is black. Dotted greens lines are the boundaries of tracking errors.

For $\xi_o \in \Xi_o$, similarly, t_i and t_{i+1} are two consecutive sampling points. As depicted in Fig. 5, $\xi_o(t_i, t_{i+1})$ and $\xi_r(t_i, t_{i+1})$ are the segments of $sc(\xi_o)$ and $sc(\xi_r)$, respectively. Assume $d(sc(\xi_{od}), sc(\xi_r)) > L$, but $d(sc(\xi_o), sc(\xi_r)) \leq L$, which means $d(\xi_{od}(t_i), \xi_r(t_i, t_{i+1})) > L$ and $d(\xi_{od}(t_{i+1}), \xi_r(t_i, t_{i+1})) > L$, but $d(\xi_o(t_i, t_{i+1}), \xi_r(t_i, t_{i+1})) \leq L$, then $\xi_o(t_i, t_{i+1})$ and $\xi_r(t_i, t_{i+1})$ must be intersecting, and thus $d(\xi_o(t_i), \xi_o(t_{i+1})) > L$ (see Fig. 5). Based on Definition 8, $d(\xi_o(t_i), \xi_o(t_{i+1})) = \|(\vec{v} + \frac{\vec{a} \times \varepsilon}{2}) \times \varepsilon\| \leq \|V\| \times \varepsilon$. Therefore, $\|V\| \times \varepsilon > L$, which contradicts the assumption $\varepsilon \leq \frac{L}{\|V\|}$. Hence, if $d(sc(\xi_{od}), sc(\xi_r)) > L$, then $d(sc(\xi_o), sc(\xi_r)) > L$. □

Based on Theorems 1 and 2, the reach-avoid verification of discretized trajectories is sufficient to entail that of nonlinear trajectories. The reach-avoid verification of discrete-time transition systems is decidable [13]. Therefore, the undecidable problem of model-checking nonlinear trajectories of agents is successfully simplified to a decidable one over discrete-time trajectories. In the next section, we introduce our approach of verifying the discrete-time models.

4 Verification Approach and Tool Support

In our verification approach, we employ Uppaal Timed Automata (UTA) [16] to build the discrete-time model of the agents, and Uppaal STRATEGO as the model checker to execute the verification. The latest version of Uppaal STRATEGO provides a function of calling external libraries. This function enables us to design a model for verification without knowing the implementation details of algorithms, hence modeling them as black boxes. Although Uppaal STRATEGO is mainly designed for strategy synthesis of stochastic timed games, our approach only leverages its function of exhaustive model checking. The semantics of UTA is timed transition systems. When discretizing time in timed transition systems, one gets discrete-time transition systems, which can be used to model the discrete-time trajectory of agents (Definition 8). Our UTA templates are designed to act only at the end of each sampling period simultaneously, so within

[3] Computation of $d(\mathbf{O}_u, \xi_{rd}(t_i, t_{i+1}))$ is in a more detailed version of this paper [12].

the sampling periods, nothing happens but only time elapses. Therefore, the semantics of our UTA templates is shown to be conservatively abstracted by the discrete-time transition semantics, with the discretizing step being equal to the sampling period of the discrete-time trajectories.

4.1 General Description of the Approach

Fig. 6 shows the workflow of the verification approach. The input of the approach is the parameters of the agents (i.e., AV and dynamic obstacles) and their boundary of the tracking errors, as well as the environment (e.g., static obstacles). In Step 1, users provide their nonlinear vehicle models, which are for

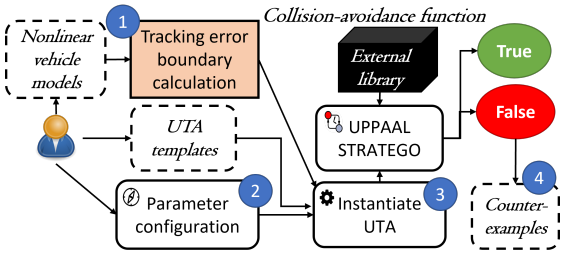


Fig. 6. The workflow of the verification approach

calculating the boundary of tracking errors. This module is the approach provided by Fan et al. [8], which is not the focus of this paper. We simply use the output of this approach in our models for verification. In Step 2, users configure the parameters of the approach, which are used for instantiating the UTA models. Parameters regulate the minimum and maximum values of the elements of agent states, e.g., linear velocity. The detailed specification of the parameters is in literature [12]. In Step 3, UTA templates of the discrete-time models are instantiated into UTA models based on the configured parameters. Note that the user-programmed collision-avoidance algorithm is embedded in the models as executable libraries, e.g., Dynamic-Link Libraries (DLL) in Windows, or Shared Object (SO) in Linux. After the instantiation of UTA, the model checker verifies the model by traversing its state space, calling the external libraries when necessary, and checking if the vehicle model avoids all obstacles and reaches the destination under all circumstances. If the verification result is "true", the algorithm is guaranteed to be correct under the current parameter configuration; otherwise, counter-examples are returned by the model checker for the users to debug their algorithm or change the configuration of the parameters (Step 4).

4.2 Design of the UTA Templates and CTL Properties

There are four UTA templates that are well designed to be reusable. The figures and the detailed description of the templates are in our technical report [12]. First, we overview the UTA templates:

– **AV Parameter Template**. Based on Definition 8, after being initialized, the AV parameters (e.g., position, speed) either stay unchanged or update

their values at the end of the sampling periods, simultaneously. Therefore, we define this template for updating the AV parameters periodically. Instances of this template are parameters of AV, hence, users can add their parameters of AV simply by instantiating this template. The update of AV parameters are synchronized by the controller template.
- **AV Controller Template**. The AV controller template mainly accomplishes three jobs: initializing the AV parameters; invoking the UTA of AV parameters periodically; making decisions, such as turning at waypoints, or calling the external function of collision avoidance when seeing an obstacle.
- **Obstacle Initialization Template**. As depicted by its name, this template is responsible for initializing moving obstacles. For each parameter of the obstacle (e.g., position, speed), the template traverses the range of its value and nondeterministically chooses one to be the initial value of the parameter. Therefore, when running the exhaustive model checking in UPPAAL STRATEGO, all the values are enumerated and verified.
- **Obstacle Movement Template**. This template is for updating the obstacle's parameters periodically. At every end of the sampling period, the AV controller UTA invokes the AV parameter UTA as well as the obstacle movement UTA. In this way, sampling the AV and dynamic obstacles is synchronized at the same moments. Note that this template updates the acceleration and heading of the obstacle every N periods, $N > 1$. As aforementioned, reasonable obstacles do not change their direction and acceleration too frequently.

The CTL properties that formalize the reach-avoid requirement are as following:

- **Obstacle avoiding**: `A[]!collision`, where `collision` is a Boolean variable that is updated every sampling period. When the distances from the safety-critical segment of AV to any of the obstacles in the map is less than the boundaries of tracking errors, `collision` is turned to true, and remains *false* elsewhere. Therefore, this query asks: for *all* execution paths, is collision *always* avoided?
- **Destination reaching**: `A<>controller.STOP`, where `STOP` is a location in UTA of AV's controller. When `controller` goes to location *STOP*, it means that the AV has reached the destination. Therefore, this query asks: for *all* the execution paths of the model, does AV *eventually* reach the destination?

4.3 Reduction of the State Space of the UTA Model

To explore all the possible behaviors of dynamic obstacles, in the worst-case scenario, we would have to explore the entire map, and enumerate all possible values of linear speeds, rotational speeds, and headings of dynamic obstacles. This generates a huge state space of the model that can be infeasible to check. In this section, we introduce how to reduce the state space of the UTA model without damaging the completeness of the verification.

Reduction of Initial Values of Parameters. Even though the dynamic obstacles can appear at any positions in the map, some positions are too far away

from the AV to be relevant at the current period, and some are too close to the AV to be possible to be avoided. Hence, we categorize positions into three classes, namely *safety-critical* area, *closest* area, and *valid* area. Fig. 7 depicts these three kinds of areas. The safety-critical area is defined in Definition 6.

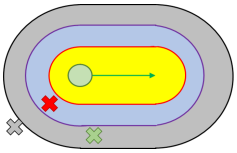


Fig. 7. The green arrow is the reference path. The green circle is the AV. The crosses are the dynamic obstacles, where red and grey ones are invalid positions, and the green cross is valid.

Positions from which the distance to the safety-critical segment of the reference path is shorter than or equal to $V \times n \times \varepsilon$ is called *closest* area, where V is the velocity of the dynamic obstacle, ε is the sampling period, and $n \in \mathbb{N}$ is a coefficient whose value depends on the physical limitations of the AV. Obstacles appearing within the closest area are impossible to be avoided, so they should be excluded from the valid initial positions. Similarly, positions from which the distance to the safety-critical segment is greater than $V \times n \times \varepsilon$ and less than or equal to $V \times m \times \varepsilon$ are called *valid* area, where $m \in \mathbb{N}$ is a coefficient for calculating the detection period of sensors. Obstacles outside this area cannot enter the safety-critical area within the current detection period, so they should be excluded from the verification in this period.

Collision-avoidance algorithms can turn the AV to any angle, so any heading of the dynamic obstacles can be dangerous. Hence, the initial value of heading is within π to $-\pi$ and cannot be reduced, and same for the linear velocity.

Phased Verification. Another way of handling large state spaces is to split the verification into several phases, and in each phase, the state space is constrained under a solvable level. For example, when the traveling time of AV is long, the entire journey can be split into multiple sections. As long as the concatenating states between consecutive phases are unchanged, the logic conjunction of verification results of each phase implies the result throughout the entire verification.

5 Experimental Evaluation

The experiments are conducted on a server with Ubuntu 18.04, 48 CPU, and 256 GB memory. The verification is executed in Uppaal 4.1.20-stratego-7[4] [6].

5.1 The Collision-Avoidance Algorithm to be Verified

In the following experiments, we employ a state-of-the-art algorithm to demonstrate the ability of our verification approach. The algorithm is based on dipole flow fields [23], and calculates static flow fields for all objects in the map, and dynamic dipole fields for moving objects. When the AV starts to move, the static flow fields generate attractive forces along the reference path to draw the AV to move towards the closest waypoint. When it encounters a dynamic obstacle, dipole fields are generated dynamically and centered by these two moving objects. Magnetic moments are thus calculated in these dipole fields, which push

[4] The models and external library: https://github.com/rgu01/FM2021.

the moving objects away from each other. Therefore, the AV could possibly deviate from its planned path when meeting dynamic obstacles, and thus, it might encounter some static obstacles that are not taken into account by the reference path. Static flow fields now generate repulsive forces surrounding these static obstacles and push the AV away from them. Formulas for calculating these fields and forces can be found in the literature [23]. This algorithm has not been comprehensively verified considering all possible scenarios of dynamic obstacles.

5.2 Verification Results

In this study, we verify the model containing a C-code library that implements this algorithm, by using our approach. We demonstrate how to find the potential problems of this newly-designed algorithm by using counter-examples returned from the approach, followed by verifying iteratively the improved version.

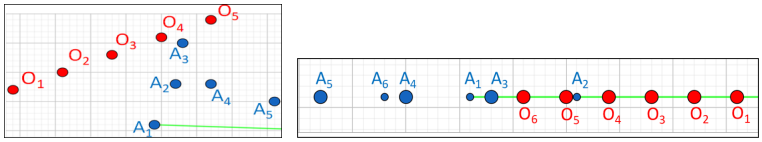
Experiment Design. We report in Table 1 several statistics relevant to the obtained results. For each scenario S, we vary the following aspects relevant in real scenarios: (i) WP representing the number of waypoints, (ii) TT that stands for the travelling time of AV, (iii) DO, the number of dynamic obstacles, and (iv) VA, the number of allowed velocities of dynamic obstacles. In scenarios S1

Table 1. Verification results of the improved version of the algorithm.

S	Environment		Obstacles		Avoiding Obstacles			Reaching Destination		
	WP	TT	DO	VA	NOS	CT	Result	NOS	CT	Result
S1	2	25	1	1	547,617	2.7 s	true	545,505	5.5 s	true
S2	6	25	1	1	411,747	1.8 s	true	411,168	3.6 s	true
S3	2	85	1	1	3,222,290	15.3 s	true	3,217,767	31.8	true
S3.1	1	30	1	1	1,532,082	7.4 s	true	1,527,811	15.7 s	true
S3.2	1	30	1	1	1,183,792	5.5 s	true	1,185,550	11.4 s	true
S3.3	1	25	1	1	506,416	2.4 s	true	504,406	4.7 s	true
S4	2	15	1	3	12,317,809	1.0 mins	true	12,498,924	2.1 mins	true
S5	2	15	2	1	1,398,011	7.6 s	false	226,896,902	43.2 mins	true

and S2, we use one phase of verification and one allowed velocity, which means that the dynamic obstacle can appear at any moment, always moving at the highest speed throughout the verification. S3 is similar to S1 but it prolongs the travelling time of the AV, and thus, the verification is split into three phases (S3.1 - S3.3). In S4, the dynamic obstacle has three possible velocities, which means its velocity has three initial values and changes arbitrarily during the verification. S5 increases the number of dynamic obstacles to 2, which means there could be at most 2 dynamic obstacles in the map at the same time. For each scenario S, we report the number of states (NOS) and the computation time (CT) needed to verify two requirements, namely obstacle avoiding and destination reaching (see Section 4.2 for details). These two values are useful indicators of our approach's performance dealing with various scenarios. All the dynamic obstacles are detected only when they get close to the AV, i.e., they are not foreknown by the AV.

Problems Discovered by Counter-examples. Initially, the proposed collision-avoidance algorithm could not pass the reach-avoid verification in any of these



(a) Problematic scenario 1 (b) Problematic scenario 2

Fig. 8. Problematic scenarios discovered by counter-examples. AV's discretized trajectory is blue dots. The dynamic obstacle's discretized trajectory is red dots. AV's reference path is the green line. For differentiation, positions that are too close but belong to different time points are represented by small and large dots in scenario 2. A_n and O_m indicate the AV and obstacle, respectively, n and m are time points.

scenarios, and we have discovered several problematic scenarios by analyzing the counter-examples returned from our approach:

Problematic scenario 1. When there is only one dynamic obstacle whose maximum velocity is less than the maximum velocity of AV, the dipole flow fields generated by the algorithm sometimes draw the AV to the obstacle instead of pushing the obstacle away from it, until their distance is too short (see Fig. 8(a)). This happens because the magnetic moments could push or draw the moving objects. Here, we improve the algorithm by simply turning the direction of the magnetic moments before the AV and the dynamic obstacle get too close.

Problematic scenario 2. When the dynamic obstacle and AV move directly towards each other, the dipole fields can only generate magnetic moments on the line of their moving directions, which drive the AV to its opposite direction but on the same line. When the dynamic obstacle keeps moving towards the same direction, the AV can only move backwards until its distance is longer than a certain value and turns 180° towards its next waypoint, which soon lets the AV get close to the dynamic obstacle again and turn backward (see Fig. 8(b)). According to the counter-examples, this scenario keeps happening iteratively until the AV stops at the boundary of the map, and is hit by the dynamic obstacle eventually. This is the so-called "livelock" scenario that was also discovered by Gu et al. [11]. To overcome this, we force the AV to turn slightly when its heading is opposite to a dynamic obstacle's heading.

Experimental Results. Although the improved algorithm passes the verification in S1-S4, our results suggest that it still cannot satisfy the obstacle-avoiding requirement in the last scenarios (S5) that contain more than one dynamic obstacle (see Table 1). Note that the destination-reaching property is still satisfied in S5, because the vehicle models are not designed to stop when a collision happens. The rationale of this design is that collisions do not necessarily stop a car from continuing moving. We want to see if the dipole-flow field algorithm can draw the vehicle to its destination anyway when it deviates from the planned paths. Counter-examples are found relatively fast in S5, even though it is more complicated than other scenarios. We leave the further improvement of the algorithm to deal with multiple agents as a future work. The experiments have demonstrated the approach's ability of discovering problems in the early stage of designing collision-avoidance algorithms, and proving the absence of errors in some scenarios for the improved version of the algorithm.

6 Related Work

Mitsch et al. [18] propose a method to verify safety properties of robots. Their method is based on hybrid system models and differential dynamic logic for theorem proving in KeYMaera. Abhishek et al. [1, 2] also use KeYMaera for collision-avoidance verification. Their models consider the realistic geometrical shapes of vehicles, as well as the combination of maneuvers and braking. Heß et al. [14] propose a method to verify an autonomous robotic system during its operation, in order to cope with changing environments. Our work differs from the above studies in the following aspects: we prove that the reach-avoid verification of nonlinear vehicle models can be simplified to a decidable problem of verifying discrete-time models. In addition, our approach provides counter-examples that are useful to improve the algorithms.

Shokri-Manninen et al. [22] have proposed maritime games as a special case of Stochastic Priced Timed Games and modelled the autonomous navigation using Uppaal STRATEGO. Their models do not consider the nonlinear kinematics of the vessels, and the options of maneuvers for collision-avoidance are limited. O'Kelly et al. [19] have developed a verification tool, called APEX, and have investigated the combined action of a behavioral planner and state lattice-based motion planner to guarantee a safe vehicle trajectory. In contrast, our approach provides users a generic interface to verify their specific vehicle models equipped with their own collision-avoidance functions. This feature is beneficial to finding bugs in the early stage of designing new algorithms, or employing modified ones.

Although our work relies on the theorems proposed by Fan et al. [8], our work is orthogonal to theirs, that is, their work can be used for the initial construction of reference paths that avoid static obstacles, and our method can be used to verify the dynamic collision-avoidance function of moving obstacles.

7 Conclusion and Future Work

In this paper, we propose a verification approach to formally verify reach-avoid requirements of autonomous vehicles, assuming nonlinear trajectories of movement. We overcome the difficulty of verifying nonlinear hybrid vehicle trajectories by transforming the latter into discrete-time trajectories whose verification we prove sufficient to guarantee meeting the requirements of the original nonlinear ones. Moreover, we engage tool support (i.e., Uppaal STRATEGO) that provides users a generic interface to configure and verify their own vehicle models equipped with different collision-avoidance algorithms. We show the abilities of our verification method by model checking a state-of-the-art collision-avoidance algorithm based on dipole flow fields, which discovers bugs not detectable by simulation or testing.

Some interesting directions of future work include: (i) exploring ways of handling complex vehicle models that represent more detailed kinematic features, and (ii) statistical verification of the cases where the distances between dynamic obstacles and AV are smaller than the tracking-error boundaries but collisions do not necessarily occur.

Acknowledgement We acknowledge the support of the Swedish Knowledge Foundation via the profile DPAC - Dependable Platform for Autonomous Systems and Control, grant nr: 20150022, and via the synergy ACICS – Assured Cloud Platforms for Industrial Cyber-Physical Systems, grant nr. 20190038.

References

1. Abhishek, A., Sood, H., Jeannin, J.B.: Formal verification of braking while swerving in automobiles. In: Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control. pp. 1–11 (2020)
2. Abhishek, A., Sood, H., Jeannin, J.B.: Formal verification of swerving maneuvers for car collision avoidance. In: 2020 American Control Conference (ACC). pp. 4729–4736. IEEE (2020)
3. Alur, R., Courcoubetis, C., Dill, D.: Model-Checking in Dense Real-Time. Information and computation **104**(1), 2–34 (1993)
4. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theoretical Computer Science **126**, 183–235 (1994)
5. Daniel, K., Nash, A., Koenig, S., Felner, A.: Theta*: Any-angle path planning on grids. Journal of Artificial Intelligence Research **39**, 533–579 (2010)
6. David, A., Jensen, P.G., Larsen, K.G., Mikučionis, M., Taankvist, J.H.: Uppaal stratego. In: TACAS. Springer (2015)
7. DeCastro, J.A., Alonso-Mora, J., Raman, V., Rus, D., Kress-Gazit, H.: Collision-free reactive mission and motion planning for multi-robot systems. In: Robotics research, pp. 459–476. Springer (2018)
8. Fan, C., Miller, K., Mitra, S.: Fast and guaranteed safe controller synthesis for nonlinear vehicle models. In: International Conference on Computer Aided Verification. pp. 629–652. Springer (2020)
9. Fan, C., Qin, Z., Mathur, U., Ning, Q., Mitra, S., Viswanathan, M.: Controller synthesis for linear system with reach-avoid specifications. IEEE Transactions on Automatic Control (2021)
10. Fox, D., Burgard, W., Thrun, S.: The dynamic window approach to collision avoidance. IEEE Robotics Automation Magazine **4**(1), 23–33 (1997)
11. Gu, R., Marinescu, R., Seceleanu, C., Lundqvist, K.: Formal verification of an autonomous wheel loader by model checking. In: Proceedings of the 6th Conference on Formal Methods in Software Engineering. pp. 74–83. ACM (2018)
12. Gu, R., Seceleanu, C., Enoiu, E.P., Lundqvist, K.: Formal verification of collision avoidance for nonlinear autonomous vehicle models. Tech. rep., Mälardalen University (April 2021), http://www.es.mdh.se/publications/6205-
13. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? Journal of computer and system sciences **57**(1), 94–124 (1998)
14. Heß, D., Althoff, M., Sattel, T.: Formal verification of maneuver automata for parameterized motion primitives. In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 1474–1481. IEEE (2014)
15. Lafferriere, G., Pappas, G.J., Yovine, S.: A new class of decidable hybrid systems. In: International Workshop on Hybrid Systems: Computation and Control. pp. 137–151. Springer (1999)
16. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. In: International journal on software tools for technology transfer. pp. 134–152. Springer (1997)

17. LaValle, S.M.: Rapidly-exploring random trees: A new tool for path planning. Tech. rep., Computer Science Dept., Iowa State University (10 1998)
18. Mitsch, S., Ghorbal, K., Vogelbacher, D., Platzer, A.: Formal verification of obstacle avoidance and navigation of ground robots. The International Journal of Robotics Research **36**(12), 1312–1340 (2017)
19. O'Kelly, M., Abbas, H., Gao, S., Shiraishi, S., Kato, S., Mangharam, R.: Apex: Autonomous vehicle plan verification and execution. SAE World Congress (2016)
20. Platzer, A.: Differential-algebraic dynamic logic for differential-algebraic programs. Journal of Logic and Computation **20**(1), 309–352 (2010)
21. Rabin, S.: Game programming gems, chapter a* aesthetic optimizations. Charles River Media (2000)
22. Shokri-Manninen, F., Vain, J., Waldén, M.: Formal verification of colreg-based navigation of maritime autonomous systems. In: International Conference on Software Engineering and Formal Methods. pp. 41–59. Springer (2020)
23. Trinh, L., Ekström, M., Çürüklü, B.: Dipole flow field for dependable path planning of multiple agents. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (September 2017)