Safety and Progress Proofs of a Reactive Autonomous Racing Algorithm

Abolfazl Karimi, Manish Goyal, Parasara Sridhar Duggirala

Department of Computer Science

University of North Carolina, Chapel Hill, NC

Email: {ak,manishg,psd}@cs.unc.edu

Abstract—In this paper, we perform a safety and performance analysis of an autonomous vehicle utilizing a *reactive planner and controller* to navigate a race lap. Unlike traditional planning algorithms that use a map of the environment, the reactive planner generates the plan based solely on current sensor inputs. Our reactive planner selects a waypoint on the local Voronoi diagram, and we use a pure-pursuit controller to navigate towards this waypoint.

Our analysis consists of two parts. The first part demonstrates that the reactive planner computes a plan locally consistent with the Voronoi plan derived from a full map. The second part models the vehicle's navigation along the Voronoi diagram as a hybrid automaton. To prove the safety and performance specifications, we compute the reachable set of this hybrid automaton and apply enhancements to simplify this computation.

We show that an autonomous vehicle using our reactive planner and controller is safe and successfully completes a lap on five different circuits. Additionally, we have implemented our planner and controller in a simulation environment and on a scaled-down autonomous vehicle, demonstrating that our approach works well across a variety of circuits.

I. INTRODUCTION

Autonomous vehicles (AVs) are expected to yield several societal benefits by improving the efficiency, lowering resource utilization, and improved passenger safety. However, these benefits would be in jeopardy if AV implementations do not meet the required safety standards. To provide rigorous end-to-end formal guarantees of AV navigation, one needs to provide not only individual formal guarantees for each component in AV, but also compose the safety certificates. A typical workflow of an autonomous vehicle involves three main steps: perception, planning, and control. The estimation of the current state of the vehicle and its surroundings is performed by perception. The sequence of states to be visited for safely navigating the environment is determined by planning. And finally, the plan is realized through a control algorithm. Perception is often performed using neural networks and localization algorithms; planning is often performed by sampling based motion planners; and control is performed using geometric or optimization techniques. Given the complexity of each individual steps and their composition, rigorous safety certificates for end-to-end AV navigation is beyond the scope of current state-of-the-art verification techniques.

For providing end-to-end guarantees on AV navigation, we consider a slightly different implementation of AV navigation — *reactive planning and control*. Unlike traditional perception techniques, reactive planning and control do not have a map of the environment. Instead, reactive planners navigate the environment based on the current data that it receives from the sensors. Hence, the plan and the control inputs are generated *on-the-fly*. For example, one of the simplest algorithms for navigating a narrow corridor by an autonomous vehicle is *follow-the-wall*. In follow the wall strategy, the perception algorithm of the vehicle estimates the distance between the AV and the walls of the corridor. The planner would dynamically decide on a way-point that is *safe* and is at a pre-determined distance ahead from the vehicle's current position along the wall. To reach that way-point, the controller deploys a standard PI/PID controller or a model predictive controller. Such algorithms are standard examples in undergraduate robotics courses for navigation.

While not particularly sophisticated, such reactive planning and control algorithms are simple and can be deployed in instances with limited computational resources. Additionally, reactive planning and control based approaches are useful in instances where the environment is highly dynamic or mapping of the environment has not yet been performed or is computationally very expensive. Finally, these algorithms can be deployed in a wide range of scenarios as it doesn't require constructing a high-fidelity map of the environment.

A major drawback of reactive planning and control is that it is very challenging to provide safety or performance guarantees. Since the map of the environment is not known, the waypoints generated during the motion planning are always relative to the vehicle. Hence, as the vehicle navigates through the environment, the waypoint in the next instance also evolves. *This dynamic nature of the state of the vehicle and the waypoints makes the safety analysis very challenging.* Furthermore, as the vehicle navigates through the environment, the new sensor readings could cause changes in the planned path making safety analysis a very challenging task.

In this paper, we perform safety and performance analysis of a reactive planning and control algorithm deployed on an autonomous vehicle that is navigating a race lap. Our planner involves computing a Voronoi diagram of the walls visible to the perception and our control algorithm implements the pure-pursuit algorithm. Our safety and performance analysis has two parts. In the first part, we demonstrate that the waypoint computed by the reactive planner is consistent with the planner that has access to the full map. In the second part, we model

the co-evolution of the state of the vehicle and the waypoint as a hybrid automata and compute an artifact called *reachable set*. The reachable set contains all the configurations visited by the autonomous vehicle while realizing the plan using the pure-pursuit control algorithm. We show that the reachable set is safe (no overlap with the boundaries of the race track) and achieves a fixed point after the vehicle completes a full lap. This proves that the vehicle satisfies the safety specification while guaranteeing that it will eventually complete the lap. We also employ abstraction techniques from hybrid systems literature [51, 42] to improve the efficiency of the reachable set computation algorithm.

The primary contribution of this paper is to establish the safety and performance specification of reactive planning and control algorithm used for navigation of autonomous vehicles. While several works provide individual guarantees for perception or planning or control algorithms, this paper investigates end-to-end guarantees for robot navigation. In addition, we also show the effectiveness of our reactive planning and control algorithm in two ways. First, we develop a simulation environment of a race lap using Unreal engine and deploy the vehicle in various types of race tracks. Second, we implement the algorithm on an open source hardware platform of F1Tenth, a scaled down version of autonomous vehicle built on a Traxxas RC car.

II. RELATED WORK

Planning and control of autonomous vehicles as well as their safety verification has received a lot of attention in the recent past. A comprehensive survey of many of these techniques are provided by [37, 39]. In this section, we briefly discuss the various planning and control techniques and some of the verification techniques presented in the literature and contrast them with the techniques presented in this paper.

Planners for autonomous vehicles are often hierarchical in nature [31, 52]. The task planner selects the high level task to be performed by the vehicle and the motion planner implements the task decided by the task planner. In this paper, since we are concerned with an autonomous racing vehicle, the task of the vehicle is to complete the lap. We hence focus our attention to the motion planning aspect of the vehicle.

Motion planning of autonomous vehicles is primarily divided into two methods. First are the geometric based planning methods where the sequence of waypoints for the vehicle are decided on the geometry of the configuration space [50, 32]. Planning based on Voronoi diagrams is one of the popular geometric techniques for planning [48, 46]. Second are the sampling based motion planning techniques. In particular, Rapidly-exploring Random Trees (RRTs) [27, 22] and Probabilistic roadmaps (PRMs) [24] are two of the most influential techniques. In the literature, several variants of RRTs and PRMs specific to the domain of autonomous vehicles have been proposed [55, 6, 28, 40, 21]. All of these techniques assume that a partial map of the environment is provided and the location of the vehicle in the map is known. In the case of reactive planning and control, we do not assume that a partial map of the system is known.

Reactive planning was proposed as an alternative to offline planning when all the information for completing a task are not available to the robot [19], or when the environment is highly dynamic [8] such as robot soccer [9] or human collaborative environment [15]. Such reactive plans have been successfully deployed in robots that are resource constrained such as small-scale helicopters [44], micro air vehicles [45], and autonomous sailboats [41]. Additionally, reactive planners have also been used to modify an existing plan due to the presence of dynamic obstacles [33, 34]. Finally, when autonomous vehicles have to satisfy service requests along liveness specification given according to a temporal logic formula, a reactive sampling based motion planning algorithm has been used [53, 54].

Control of autonomous vehicles is also divided into two categories, geometric and model based. In geometric control techniques, the underlying geometric properties of the bicycle model are used in order to make the vehicle reach its destination. Two popular geometric control techniques are pure-pursuit [38, 47] and Stanley [20]. These techniques are intuitive and easy to implement. Model based control (MPC) techniques assume a given model of the vehicle and generate control inputs depending on the model [12, 35]. Variants of MPC such as, linearization with path tracking [17], online linearization [18], and successive linearizations [43] have been proposed in the literature. In fact, MPC based control methods have been used in other autonomous racing vehicles [49].

In the safety analysis of mobile robots, the synthesis of a safe plan based on temporal logic specification has received a lot of attention [26, 16, 25]. For proving the safety of the control algorithms, various reachable set computation methods have been proposed and evaluated for behavior of an autonomous vehicle in different scenarios [2, 30, 29, 36]. Given a map of the environment, accurate sensors, noise free localization, and accurate model of the vehicle dynamics, it is possible to provide a sequence of waypoints for moving along the track and prove that the vehicle finishes the lap without colliding with any obstacles using standard reachability based techniques. However, in this paper, we consider reactive planning and control, where the waypoint dynamically changes along with the vehicle position and orientation.

The works that are closest to the current work are [11] and [13]. In both these works, the plan of the vehicle is based on computing the Voronoi diagram. In [11], the authors do not precisely model the interaction between the waypoint decided by the planner and the control algorithm and hence do not provide any safety guarantees even when the track is known apriori. In [13], in contrast to our work, the authors assume that a partial map of the environment is known. Furthermore, they do not provide any safety guarantees assuming an uncertainty in the initial conditions of the vehicle.

Fig. 1. Bicycle model [47]

Fig. 2. Example of a vehicle calculating the local Voronoi diagram and selecting the way-point on the Voronoi diagram.
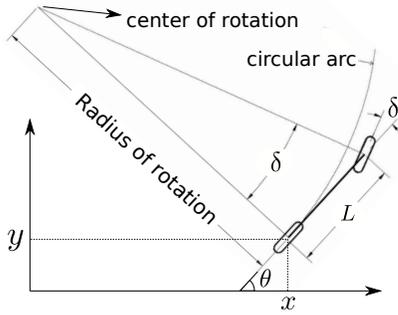
III. PRELIMINARIES

A. Problem definition

Consider that an autonomous vehicle is tasked to complete a circuit track without hitting the borders of the track. Completing the track is a *progress* requirement, and avoiding the track boundaries is a *safety* requirement. Furthermore, the planning and control algorithms are required to be *reactive*: at each time step, steering and speed controls are calculated from the current sensor data only. Assuming that there is no lateral wheel slippage, we can isolate the problem of finding the steering control from finding the velocity control. This is because if a wheel does not slip laterally, it will move along its direction and so the shape (i.e. footprint) of the trajectory only depends on the steering angle. In other words, the shape of the trajectory is not influenced by the speed, as long as the speed is nonzero. In this paper, we present a simple reactive planner and controller and formally analyze its safety and progress properties.

B. Bicycle model of a car

Our formal analysis is based on the bicycle model of a car, where we imagine that there is one rear wheel at the center of the rear axle and one front wheel at the center of the front axle. We assume no wheel slippage, and only the front wheel can steer. This model defines a dynamical system. The *state* of the system at time t is described by the triple $(x(t), y(t), \theta(t))$ where (x, y) are the coordinates of the rear wheel in some inertial frame, say the racing track, and θ is the angle of the heading direction of the bicycle measured from the x-axis counter-clockwise. If v is the speed (magnitude of velocity) of the rear wheel, L is the *wheelbase* (distance between the rear and front wheels), and δ is the steering angle, then

$$\begin{cases} \dot{x} = v \cdot cos(\theta) \\ \dot{y} = v \cdot sin(\theta) \\ \dot{\theta} = \frac{v}{L} tan(\delta) \end{cases}$$

Illustration of the bicycle model of the vehicle is provided in Figure 1.

IV. PLANNING AND CONTROL

In this section we describe our planning and control algorithms which are a variation of Voronoi-based planning
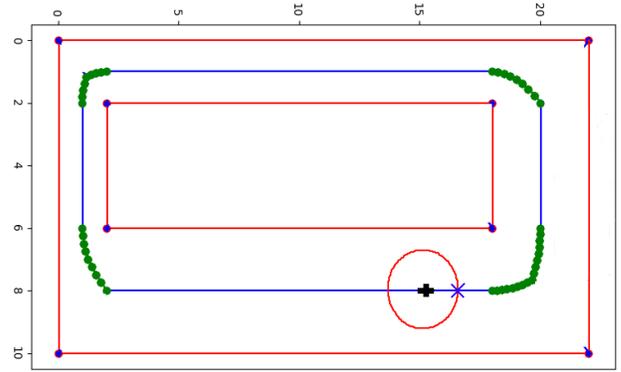
[11] and geometric control [47]. The safety and progress properties of our algorithms will be formally analyzed and experimentally validated in subsequent sections.

A. The Planner

Our planner is *reactive* in the sense that it does not remember its past inputs or outputs. The input is simply a 2D point-cloud from lidar, and the output is a 2D *waypoint* passed to the controller. We assume that the environment is polygonal. The planner calculates a Voronoi diagram corresponding to the point-cloud, then it chooses a point on the Voronoi diagram as the waypoint.

The first step is to calculate a Voronoi diagram. Since the environment is polygonal, the planner converts the 2D point cloud to a set of line segments. Representing a set of points by a line segment simplifies the computation and representation of the Voronoi diagram. Since the input to the Voronoi computation is a set of line segments, the Voronoi edges are either linear or parabolic arcs. After computing the Voronoi diagram, we approximate each parabolic edge by a polyline using a *deviation threshold*. The *deviation threshold* determines the maximum distance of the points on a parabolic arc from the approximating polyline. This linear approximation simplifies the planner and its formal analysis. A Voronoi diagram calculated by the planner is called a *local* Voronoi diagram since it is computed for the point-cloud visible from lidar. This is in contrast to the *global* Voronoi diagram where the diagram is computed with respect to the whole polygonal environment.

The next step is to choose a waypoint based on the local Voronoi diagram. We choose a point *on* the Voronoi diagram to try to stay as far as possible from the track walls (i.e. to be as safe as possible). If the waypoint is too close or too far from the vehicle, the controller may make the car steer too sharply or slowly. We choose among the points at a fixed distance from the center of the rear axle. This distance is called the *lookahead distance* and the corresponding circle is called the *lookahead circle*. The lookahead circle may intersect the Voronoi diagram in more than one point, so we need to choose among them. Since the goal is to make the car progress
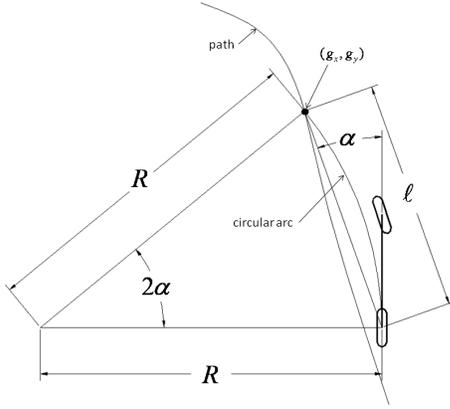
Fig. 3. Purepursuit controller [47].

towards finishing the track, the intersection point further along the heading direction of the car is selected as the waypoint.

This is illustrated in the Figure 2. Here, an autonomous vehicle is navigating a lap where the inner and outer walls are aligned rectangles. The Voronoi diagram — the set of points that are equidistant from the walls is shown in the blue straight segments and the curved green segments. The vehicle (depicted as a black cross) is currently situated on one of the edges of the Voronoi diagram. The circle with the radius of lookahead distance intersects with the Voronoi diagram and the appropriate intersection is labeled as the waypoint in the figure. While the figure shows the global Voronoi diagram, the vehicle does not have access to the global Voronoi diagram but can only observe part of the map.

B. The Controller

The pure-pursuit controller [5] as formulated in [47] is used to determine the steering angle. This simple controller was "the most stable and accurate tracker" of the three methods tested in [5]. Furthermore, pure-pursuit performed "fairly well and is quite robust to large errors and discontinuous paths" in comparison to a few more complicated (geometric, kinematic or dynamic) controllers [47].

The input to the pure-pursuit controller is a waypoint described in the car's *rear-axle coordinates*. The origin of the rear axle coordinates is the center of the rear axle, the x-axis is the heading of the car, and the heading of the y-axis is 90 degrees counterclockwise from the x-axis. The output is the steering angle δ for the bicycle model of a car. Let (g_x, g_y) be the coordinates of the waypoint in the rear-axle frame. Note that the lookahead distance ℓ is $\sqrt{g_x^2 + g_y^2}$. If L is the *wheelbase* of the car (i.e. distance between rear and front axles), then the pure-pursuit steering angle δ is[1]

$$\delta = tan^{-1}(\frac{2Lsin(\alpha)}{\ell})$$
$$= tan^{-1}(\frac{2Lg_y}{\ell^2}) \tag{1}$$

[1]See [47] for the derivation of the formula.

Note that this formula is valid even when the waypoint is on the right of the car, where α, g_y and δ are all negative. This formula is valid only when $g_x > 0$ i.e. when the waypoint is in front of the rear axle. In this case, we have $\frac{-\pi}{2} < \delta < \frac{\pi}{2}$. In practice, $-\delta_{max} \leq \delta \leq \delta_{max}$ where $\delta_{max} < \frac{\pi}{2}$ is the maximum possible angle that the car can steer. For example, the maximum steering angle for the Traxxas car in the open source hardware platform of F1Tenth vehicle is about 34 degrees.

V. RIGOROUS ANALYSIS OF PLANNING AND CONTROL ALGORITHMS FOR SAFETY AND PROGRESS PROPERTIES

In this section we perform rigorous analysis of the Voronoi planning along with the pure-pursuit controller. We prove that the vehicle would not hit any of the walls and will successfully complete a lap when its starting position belongs to a defined set of initial states. Our analysis has two parts: the first part is about the properties of the planner and the second part is about the closed loop behavior of the vehicle.

In the first part, we prove that the *local Voronoi diagram* computed by the vehicle from its current environment will be consistent with the *global Voronoi diagram* under certain conditions. This consistency ensures that the reactive path planning performed by the robot is equivalent to the path planning performed while having access to the map. The second part of analysis requires computing an artifact called *reachable set* of the closed loop vehicle dynamics. We model the evolution of the vehicle along the edges of the Voronoi diagram as a hybrid automaton. As shown in Figure 2, the vehicle tries to move along the edges of the Voronoi diagram. We assign one mode for modeling the evolution of the dynamics on each straight edge and use new overapproximation techniques for modeling the vehicle navigation through the curved segments. Given a set of initial positions for the vehicle on the lap, we compute the reachable set that includes all the set of states visited by the trajectories starting from the set. We demonstrate that the reachable set does not overlap with any of the walls and achieves a fixed point after finishing the lap, thus proving the safety and progress properties.

A. Consistency of local Voronoi diagrams

In this section, we prove that the Voronoi diagram computed by the vehicle with a given lidar scan is consistent with the global Voronoi diagram computed from the map. Intuitively, this proof formalizes the notion that a lidar with a sufficiently long range can detect all the edges for computing the Voronoi diagram for a small neighborhood. The proof will formalize the requirements on the track and the range of the lidar. At each point in time, only a subset of the walls are *visible* to the lidar, i.e. all the walls in its range that are not occluded. A *local Voronoi diagram* is the Voronoi diagram of the visible walls. The *global Voronoi diagram* is the Voronoi diagram of all walls. The planner chooses the waypoint from the intersection of the local Voronoi diagram and the lookahead circle. Recall that the *lookahead circle* is a circle of fixed radius centered at the rear axle of the car. We also assume that the lidar is placed

at the middle of the front axle of the car. We give sufficient conditions such that within the lookahead circle the local and global Voronoi diagrams coincide.

Consider the visible subset of the walls, the local Voronoi diagram, and the lookahead circle at some arbitrary time. Pick a point p on the global Voronoi diagram inside the lookahead circle. We give sufficient conditions such that the closest wall points to p in the global Voronoi diagram are visible to the lidar. Thus, p is also on the local Voronoi diagram. The sufficient conditions rule out the two possible cases for invisibility of the closest wall points: being out of range, or occluded by a visible point.

Let R be the range of lidar, L the distance from lidar to the rear axle, ℓ the lookahead radius, m the minimum width of the track, M the maximum width of the track, and D the minimum distance between lidar and the walls at any point in time. Then we have the following guarantee:

Theorem 1. *The local and global Voronoi diagrams coincide within the lookahead circle if*

$$R > M, \qquad (2)$$

$$R > L + \ell + \frac{M}{2}, \qquad (3)$$

$$\text{and, } D^2 \geq (L + \ell)^2 - \frac{m^2}{4}. \qquad (4)$$

Before proving Theorem 1, we will prove a lemma about the relationship between lookahead circle, walls of the circuit, and distance between a point on the Voronoi diagram and the walls.

Lemma 2. *1) Any point on (local or global) Voronoi diagrams is at most $\frac{M}{2}$ away from its closest walls.*
 2) The lookahead disk is contained in the circle $C_{L+\ell}$ of radius $L + \ell$ centered at lidar.
 3) For any point p on the global Voronoi diagram inside $C_{L+\ell}$, p's closest walls are in lidar's range.

Proof. 1) Lidar can see its closest walls since $R > M$. Furthermore, a point on the Voronoi diagram is equidistant to its closest walls.
 2) The distance from lidar to rear axle is L, and lookahead circle is the circle of radius ℓ centered at the rear axle.
 3) The closest wall points to p are in the circle of radius $L + \ell + \frac{M}{2}$ centered at lidar. By Equation 3, p's closest wall points are in lidar's range. □

Proof of Theorem 1. We need to show that for any point p on the global Voronoi diagram, if p is inside the lookahead circle then p's closest walls are visible from lidar, so that p is also on the local Voronoi diagram. By the lemma, the circle $C_{L+\ell}$ of radius $L+\ell$ centered at lidar contains the lookahead circle, so it is sufficient to assume that p is in $C_{L+\ell}$. Consider an arbitrary visible wall point w occluding a wall point u. That is, both w and u are on the same ray from lidar but u is further

Fig. 4. Consistency of local and global Voronoi diagrams. The blue curve is the global Voronoi diagram. The green dot is lidar, and the pink dot is the waypoint. The gray area is $H \setminus C_{u,\frac{m}{2}}$.

away than w. We show that p is closer to w than u. Let d be the distance of lidar to w and c be the distance of w to u (so the distance of lidar to u is $d + c$). First note that there is no Voronoi point inside the circle $C_{u,\frac{m}{2}}$ of radius $\frac{m}{2}$ centered at u, since m is the minimum width of the track. Also, the points that are closer to u than w constitute a half-plane H with distance $d + \frac{c}{2}$ from lidar and its boundary being the bisector of u and w. Hence Voronoi points that are closer to u than w are in $H \setminus C_{u,\frac{m}{2}}$ i.e. in H and outside of $C_{u,\frac{m}{2}}$. See Fig. 4. If $(d + \frac{c}{2})^2 + \frac{m^2}{4} - \frac{c^2}{4} > (L+\ell)^2$ or equivalently $d^2 + dc > (L+\ell)^2 - \frac{m^2}{4}$ then $C_{L+\ell}$ does not intersect $H \setminus C_{u,\frac{m}{2}}$. The stronger condition

$$d^2 \geq (L+\ell)^2 - \frac{m^2}{4} \qquad (5)$$

ensures that a global Voronoi point inside $C_{L+\ell}$ is closer to w than any wall point in w's shadow. This inequality is guaranteed by Inequality 4, since D is the minimum distance of lidar to walls so $d^2 \geq D^2$. □

Observe that Theorem 1 gives us some bounds on the size of the lookahead circle. The lookahead circle cannot be too large, else, the closest walls to a points in the lookahead circle might be occluded from the lidar.

B. Coupled model of control and plan

As described in Section IV-B, the pure-pursuit controller determines the steering angle δ according to the waypoint (g_x, g_y) by the planner (given in Equation 1). This waypoint lies on the local Voronoi diagram and always at a distance ℓ from the rear-axel. When the conditions given in Theorem 1 are satisfied, the waypoint also lies on the global Voronoi diagram. Given that the Voronoi diagram is a conjunction of several line segments, without loss of generality, we assume that the waypoint lies on one such line segment.

Observe that as the vehicle moves towards the waypoint, the position and orientation of the vehicle changes. This indeed changes the intersection of the lookahead circle with the Voronoi diagram and as a result, changes the waypoint. Thus, the evolution of the state of the vehicle (position and

orientation) and the waypoint are tightly coupled. In this section, we model the joint behavior of the waypoint along with the vehicle dynamics.

Observe that for the pure-pursuit controller, the waypoint (g_x, g_y) is described in the vehicle's coordinate system (i.e., the origin being the center of the rear-axel and the orientation of the vehicle as the x-axis). We transform this waypoint along the coordinate axis where the Voronoi line segment is the x-axis with the new origin is such that the initial position of the center of the rear-axel has no displacement on the x-axis. In the trajectory, the rear wheel has coordinates (x, y) and the orientation of the vehicle θ is w.r.t the line segment on the Voronoi diagram. After the coordinate transformation, the orientation δ in this coordinate system will be

$$tan(\delta) = \frac{2L}{\ell^2}(-\sqrt{\ell^2 - y^2} \cdot sin(\theta) - y \cdot cos(\theta)).$$

Therefore,

$$\dot{\theta} = \frac{v}{L}\frac{2L}{\ell^2}(-\sqrt{\ell^2 - y^2} \cdot sin(\theta) - y \cdot cos(\theta))$$
$$= \frac{2v}{\ell^2}(-\sqrt{\ell^2 - y^2} \cdot sin(\theta) - y \cdot cos(\theta)) \quad (6)$$

The closed loop behavior of the vehicle dynamics is therefore given as:

$$\dot{x} = v \cdot cos(\theta)$$
$$\dot{y} = v \cdot sin(\theta)$$
$$\dot{\theta} = \frac{2v}{\ell^2}(-\sqrt{\ell^2 - y^2} \cdot sin(\theta) - y \cdot cos(\theta))$$

Notice that this closed loop behavior is accurate as long as the waypoint lies on the same line segment. When the vehicle makes progress, the lookahead circle would intersect with a different line segment on the voronoi diagram. As a result, the coordinate system for modeling the evolution of vehicle state and the waypoint changes. We model this change as a hybrid automata.

Definition 3. *A hybrid automata is defined as a tuple* $\langle Mod, X, E, Flows \rangle$ *where*

- $Modes$ *is the set of discrete modes,*
- X *is the state space,*
- E *is the set of discrete transitions among modes, and*
- $Flows$ *describes the evolution of the system in each mode.*

Often, $Flows$ are described as a collection of nonlinear differential equations, one for each mode. Each discrete transition $e = (mode, mode') \in E$ among the modes $mode, mode' \in Modes$ of the hybrid automata has an associated $guard(e)$ condition. The hybrid automata can take the transition only when the state of the trajectory satisfies the guard condition. Additionally, after taking the discrete transition e, the state of the system changes from its current state x to a new state x' defined according to a reset function $x' = reset(e, x)$.

In the case of the vehicle following the waypoint on Voronoi diagram, when a new line segment is encountered by the lookahead circle, the change of basis variables (alignment
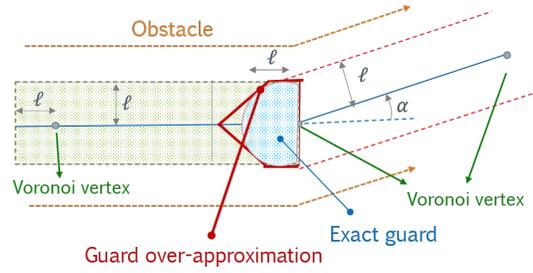
Fig. 5. Over-approximation of guard. l is the look ahead distance of the car. The original exact guard is circular and we linearize it as a convex polytope that is defined using linear constraints.

of the x-axis along the new segment) is the reset function. Notice that the set of states that take this transition lies on a circle of radius ℓ from an end point of the new line segment. This set is a non-convex set. Performing analysis on such a hybrid automata with non-convex guard conditions is very challenging. We therefore construct an abstract hybrid automata that allows more behaviors than the original hybrid automata, but is easier to analyze. More specifically, we allow the hybrid automata to non-deterministically take a transition whenever the vehicle enters the convex overapproximation of the original guard condition. Given that we strictly increase the set of possible states that can take the discrete transitions, it is easy to observe that the abstract hybrid automata includes all the behaviors of the original hybrid automata. Illustration of the abstraction of the guard condition for a hybrid automata modeling the behavior of the vehicle is given in Figure 5.

Given a circuit, we construct the hybrid automata model of the vehicle moving in the lap using our Voronoi planning and pure pursuit control algorithm. In the next section, we present the details of the techniques employed to prove the safety and progress properties of the vehicle.

C. Reachable Set Computation

Definition 4. *Given a hybrid system H modeled as an hybrid automaton and an initial set of states Θ, an execution of H is a sequence of trajectories and transitions $\xi_0 e_1 \xi_1 e_2 \ldots$ such that (i) the first state of ξ_0 denoted as q_0 is in the initial set, i.e., $q_0 = (mode_0, x_0) \in \Theta$, (ii) each trajectory ξ_i is the solution of the differential equation $Flows_i$ of the corresponding mode $mode_i$, (iii) the state of the trajectory before each discrete transition $e_i = (mode_i, mode_{i+1})$ satisfies $guard(e_i)$, (iv) and the state of the trajectory after taking the transition e_i changes to $q_{i+1} = (mode_{i+1}, x_{i+1})$ where $x_{i+1} = reset(e_i, x_i)$.*

The set of states encountered by all executions that conform to the above semantics is called the *reachable set* and is denoted as $Reach_{[H,\Theta]}$. Bounded-time variant of these executions and the reachable set defined over the time bound T is denoted as $Reach_{[H,\Theta]}^T$. We drop H and/or Θ from $Reach_{[H,\Theta]}^T$ whenever it is clear from the context, and abuse the term *trajectory* to denote the hybrid system execution as well as the solution of the differential equation of a mode.

We say the reachable set computation has a *fixed point at time t* when there exists $t \leq T$ such that $Reach^t = Reach^{t+1}$. Since it is computationally hard to compute the exact reachable set for most system classes including hybrid systems [4, 7], a verification engine is typically used to compute an overapproximation of the reachable set denoted as $\tilde{Reach}^T$. Given the sequence of reachable set computed at discrete time instances, the ith element in the is denoted as $Reach^T[i]$.

Definition 5. *A hybrid system H with initial set Θ, time bound T, and unsafe set U is said to be safe with respect to its executions if all trajectories starting from Θ for bounded time T are safe i.e., $Reach^T \cap U = \emptyset$.*

As the reachable set overapproximation $\tilde{Reach}^T$ includes more behaviors than the exact reachable set, its safety w.r.t. U proves the safety of the exact reachable set w.r.t. U. However, the safety result is inconclusive when $\tilde{Reach}^T \cap U \neq \emptyset$. We next discuss the fixed point based computation of the reachable set for a given initial set Θ.

- **Computing a Fixed Point :** Recall that the vehicle motion for a given circuit is modeled as the hybrid automata where each mode is associated with an edge from the global Voronoi diagram. We denote the lap as one full pass of the given circuit which is the sequence of edges from the global Voronoi diagram such that an end point of the last segment coincides with an initial point of the first segment. Informally, the number of laps represents the number of passes of the given circuit performed by the vehicle.
  As the vehicle completes one lap while following Voronoi edges by switching modes of the hybrid automaton, it may not arrive back at the exact system state it initially started from. As a consequence, the set of states reachable in a mode during next lap is not necessarily same as the set of states reachable in this mode during previous lap(s). Whereas, a fixed point at some time t is an evidence that the set of reachable states beyond time t is time-invariant. Computing a fixed point for the reachable set serves three important purposes - (i) arriving at the same mode(s) after completing a lap underscores *progress*, (ii) safety of the fixed point at t ensures *safety* of the reachable set at all times beyond t, (iii) it makes the analysis *efficient* because one can save a significant amount of computational resources by not requiring to compute the reachable set after t for a much larger initial set as explained next.
- **Computing additional reachable sets :** The error bound of reachable set overapproximation is proportional to the size of the initial set Θ, i.e., the larger the initial set, the higher is the overapproximation error. Whereas, computing the reachable set for each state in an infinite state system such as ours is practically impossible. The standard approach for handling this trade-off is to refine the large initial set into smaller subsets and perform a reachable set computation on each subset.
  A fixed point can assist in accelerating the reachable set computation for a large set Θ. For any two given sets

$\theta, \theta' \subset \Theta$, it may not be necessary to compute the fixed point of $Reach_{\theta'}$ once the fixed point $Reach_\theta^t$ at time t is obtained. While computing $Reach_{\theta'}$, we iteratively check whether $Reach_{\theta'}^i[i] \subseteq Reach_\theta^t[i], i \leq t$, and halt the computation as soon as the containment check returns true. Generalizing this approach to more than two sets would mean computing the fixed point of one set and performing the containment check for the rest w.r.t. the fixed point.

- **Refinement techniques :** One way to computing the reachable set for a large set Θ is to *manually* obtain the fixed point candidate $\theta \in \Theta$, and find the partitions $\theta' \in \Theta, \theta \cup \theta' = \Theta$ such that each partition has its reachable set at some $i \leq t$ contained in the fixed point. An efficient strategy is to *automatically* obtain both the fixed point candidate and other partitions. If the overapproximation error is too high for the given set Θ, it is automatically refined into multiple sets. One of these subsets is picked for the fixed point candidacy. If the overapproximation error for the candidate set is still high, it is further divided into parts until a candidate with its fixed point at some t is found. The rest of Θ is automatically partitioned depending on whether their reachable sets are contained in the fixed point at $i \leq t$ or not.

VI. EVALUATION

In this section, we evaluate our fixed-point based reachable set computation on five different tracks with different characteristics. For each track given as the sequence of global Voronoi diagram edges, a new hybrid automaton model for the evolution of vehicle along this track is generated. Typically, a hybrid automata has as many discrete modes as the number of Voronoi edges. The dynamics in each mode of the automaton is the closed loop behavior of the vehicle dynamics and the way point as described in Section V-B. Then, we employ a non-linear hybrid systems verification tool to compute the fixed point of the reachable set for each track. The computation of a fixed point in each track underscores both safety and progress of the vehicle. In addition, we tested the planning and control algorithm both in simulation and on the open source F1Tenth platform.[2]

We experiment with multiple safety verification platforms such as CORA [3], Flow* [10], and C2E2 [14] for non-linear hybrid systems. Different platforms use different symbolic representations for the reachable set. For instance, Flow* uses Taylor models, C2E2 uses Jacobian matrix and discrepancy functions, and CORA primarily makes use of zonotopes for representing the reachable sets. As their utility is application specific, we observed that overapproximation error in both C2E2 and Flow* is very high for our case studies. One possible reason could be that Taylor model approximation in Flow* and discrepancy function computed in C2E2 are too conservative. We illustrate the reachable set accuracy across Flow* and CORA on a small system in Section IX-A in the Appendix.

In the safety verification of our dynamical model, the obstacles or walls constitute the unsafe set i.e., the objective is

---

[2]https://tinyurl.com/24zfjvmn

(a) for small initial set   (b) for a large initial set
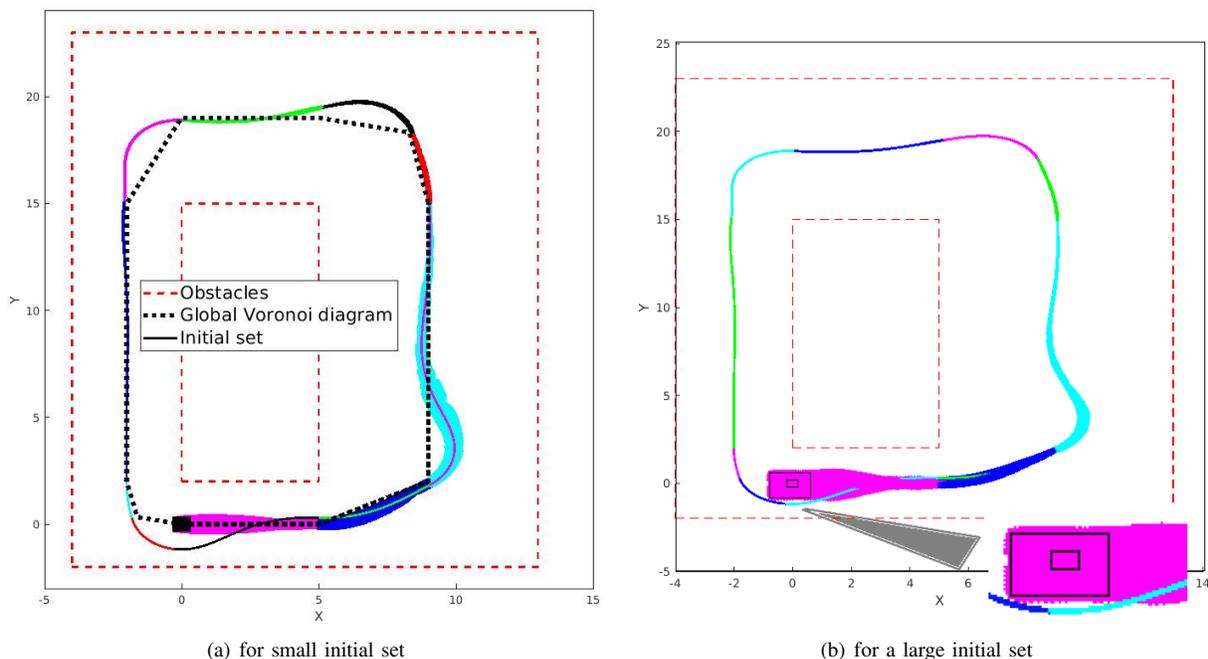
Fig. 6. **Reachable set computation for Track-1**. In Fig. 6(a), the fixed point is obtained in mode 3 in the second lap. The filled black rectangle depicts the initial set. In Fig. 6(b), the smaller rectangle corresponds to the fixed point partition and larger one is the originally given initial set.

to verify that the reachable set does not overlap with the walls. The initial set considered during reachable set computation in track-1 is $x \in [-0.3, 0.3]$, $y \in [-0.3, 0.3]$ and $\theta \in [-0.2, 0.2]$. The Voronoi diagram and the corresponding reachable set for track-1 are shown in Fig. 6(a). Based on the track width, the turns in this track encompass 4 different transitions - wide to narrow, wide to wide, narrow to narrow, and narrow to wide. Also observe that the fixed point of the reachable set computation is obtained in *mode 3* in lap 2. The values of state variables in *mode 3* across lap-1 and lap-2 are depicted in Fig. 10 that establishes our fixed point claim. The reachable set computation results for other tracks are illustrated in Fig. 11 in the Appendix. CORA takes roughly a minute to compute the reachable set for each track.

An observation from the safety verification results is that if the set of initial conditions is large (larger uncertainty of vehicle's initial orientation and position with respect to the map), CORA fails to compute the reachable set over-approximation. As suggested earlier, this can potentially be mitigated by fixed-point based partitioning of the large initial set and verifying the safety for each partition.

We have implemented the automatic partitioning algorithm and evaluated the approach on track-1 for the initial set $x \in [-0.8, 0.6]$, $y \in [-0.8, 0.6]$ and $\theta \in [-0.2, 0.2]$. CORA fails to compute the reachable set for this initial set. Our technique keeps partitioning the set until it finds a partition $x \in [-0.2, 0.2]$, $y \in [-0.2, 0.2]$ and $\theta \in [-0.2, 0.2]$ having a fixed point. (We do not partition θ in this case). Once the fixed point is found, the rest of the initial set is partitioned into intervals of the given width and reachable set is computed for each partition. In addition, at each step, reachable set contain-

ment w.r.t. the fixed point is conducted to avoid redundant computation. Fig. 6(b) demonstrates the safe reachable set successfully computed by our approach for the given set.

We also performed reachable set computation using CORA over tracks where the vehicle had to take a steep turn (turns of more than 90 degrees). For such instances, CORA failed to compute an overapproximation of the reachable set that is sufficient to establish the safety even for small initial sets. This shows that there is room for improvement in the current reachable set computation methods.

VII. EVALUATION OF PLANNING AND CONTROL IN SIMULATION AND NOISY ENVIRONMENTS

A. Computer simulations

We have built a virtual environment including various racing circuits in Unreal Engine for testing the reactive planning and control algorithm. Some of these tracks are provided in Figure 7. The tracks in the left column are similar to the tracks on which F1Tenth competitions [1] were conducted. Other tracks include a triangular track, and a straight track with static obstacles. We have also tested the simulation on some polygonal tracks for which we will provide formal analysis in the subsequent sections. Videos of these simulations are available on the web.[3] Our algorithm was able to successfully complete a lap while not colliding with any of the race track boundaries or obstacles in nine different circuits.

The vehicle used in the simulations is the vehicle from Unreal Engine 4.21's 'Vehicle Advanced' template project. The default tire friction constants are increased to avoid

[3]https://tinyurl.com/24zfjvmn
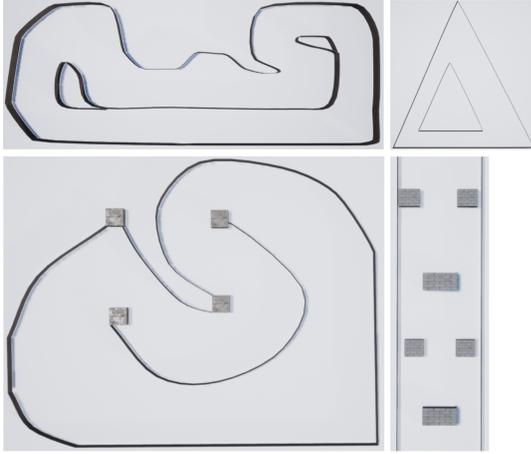
Fig. 7. Racing tracks for the simulations.

Fig. 8. Physical validations.

(a) Flow*

(b) CORA

Fig. 9. Reachable sets computed in Flow* and CORA with time step 0.02 sec and time bound 15 sec for a set of initial states. The vehicle follows a vertical path downwards before making a left turn.

wheel slippage. We have implemented a model of Hokuyo UST-10LX LiDAR in Unreal Engine. The measurements are taken using Unreal Engine's line tracing. We do not model measurement errors and uncertainty.

B. Physical validation

We also have tested the planner and controller on the open source hardware platform of F1Tenth vehicle. This is to show the practicality of our simple reactive planning and control in the presence of noise, unstructured environment, and limited computational resources. For example, Hokuyo UST-10LX LiDAR provides a new point-cloud every 25 milliseconds, but our planning and control is an order of magnitude faster on the Nvidia Jetson TX2's CPU so we can process every point-cloud.

We tested the car on two tracks. The simpler one is shown in Fig. 8 where the track boundaries are mostly structured using corrugated cardboard rolls. The harder track (provided in the videos) is an office with chairs and tables around the office walls and in the middle of the office. Our algorithm manages to successfully complete a lap while avoiding obstacles on both these tracks.

VIII. CONCLUSION AND FUTURE WORK

We have illustrated the reachability based verification results of a reactive planning and control autonomous vehicle that uses Voronoi diagrams for planning and pure-pursuit controller

for navigation. To the best of authors' knowledge, this is the first work that considers all three aspects of planning, control, and the dynamically changing waypoint for prove the safety specification. We also demonstrated reactive planning and control technique in various tracks in simulation, and on two physical tracks using a scaled down version of autonomous vehicle. We believe that the coupled effect of planning and control requires further investigation. In future, we intend to extend this to prove and demonstrate the safety of autonomous vehicles in presence of dynamic obstacles.

IX. EVALUATION DETAILS

A. Reachable set accuracy

Suppose the track is defined in a stationary (x, y) frame, θ is the vehicle's orientation and ℓ is the look ahead distance. As a part of experimentation, we perform the reachable set computation in Flow* and CORA by modeling a single turn on the track as the hybrid automata. The initial set is given as intervals over state variables i.e., $x \in [0.0, 0.5]$, $y \in [0.9, 1.1]$ and $\theta \in [-0.5, -0.5]$. The reachable sets computed in both
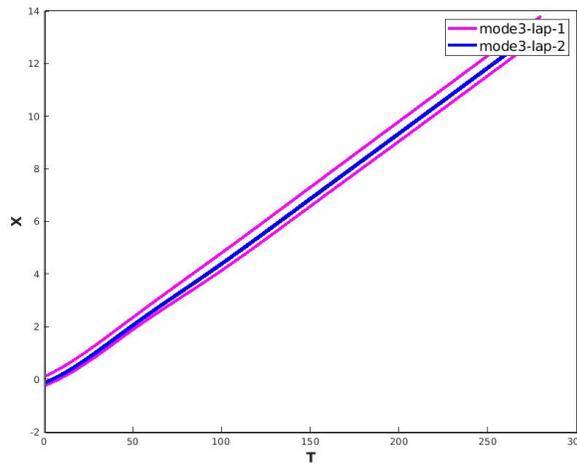
(a) Track 1-X
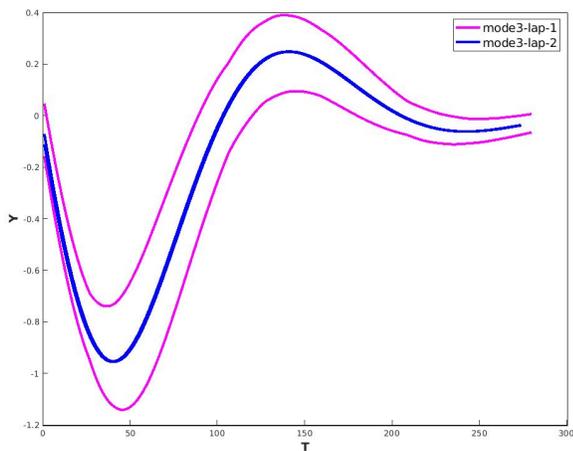
(b) Track 1-Y

Fig. 10. **Fixed point illustration using state variables in Track-1**

(a) Track 4

(b) Track 5

Fig. 11. Reachable set of our reactive planner and controller on two different tracks.

tools are shown in figures 9(a) and 9(b). The divergent behavior of the reachable set in Flow* is seemingly due to error compounded over time because of coarse approximation. The figures also illustrate that the vehicle while turning swings to some extent before merging back on to the track.

B. Fixed Point of Reachable Sets

The proof that our reactive planning and control algorithms completes a lap requires the reachable set to reach a fixed point. We illustrate that the reachable set after completing a lap is contained within the initial set or reachable set of the previous lap. The projection of reachable set on x and y axis is given in Figure 10

C. Reachable Sets on Various Tracks

The reachable sets of our reactive planner and controller on two of the tracks are given in Figure 11. The reachable sets for more laps are available in [23].

REFERENCES

[1] F1tenth website. URL https://f1tenth.org/.

[2] M. Althoff and J. M. Dolan. Online verification of automated road vehicles using reachability analysis. *IEEE Transactions on Robotics*, 2014.

[3] M. Althoff, D. Grebenyuk, and N. Kochdumper. Implementation of taylor models in cora 2018. In *Proc. of the 5th International Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 145–173, 2018.

[4] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995. Hybrid Systems.

[5] Omead Amidi and Chuck E Thorpe. Integrated mobile robot control. In *Mobile Robots V*, volume 1388, pages 504–523. International Society for Optics and Photonics,

1991.

[6] Georges S Aoude, Brandon D Luders, Daniel S Levine, and Jonathan P How. Threat-aware path planning in uncertain urban environments. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 6058–6063. IEEE, 2010.

[7] Eugene Asarin, Olivier Bournez, Thao Dang, and Oded Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In Nancy Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, pages 20–31, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-46430-3.

[8] Fethi Belkhouche. Reactive path planning in a dynamic environment. *IEEE Transactions on Robotics*, 25(4):902–911, 2009.

[9] James Bruce and Manuela M Veloso. Real-time randomized path planning for robot navigation. In *Robot Soccer World Cup*, pages 288–295. Springer, 2002.

[10] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, pages 258–263, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-39799-8.

[11] A Das, P Mukherjee, C Wang, G Salas, S Bedi, and S Waslander. Graph based path planning in unknown environments using voronoi diagrams. In *Canadian Congress of Applied Mechanics, Proceedings of the 2011*. Vancouver, BC, 2011.

[12] Alessandro De Luca, Giuseppe Oriolo, and Claude Samson. Feedback control of a nonholonomic car-like robot. In *Robot motion planning and control*, pages 171–253. Springer, 1998.

[13] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, 29(5):485–501, 2010.

[14] Parasara Sridhar Duggirala, Sayan Mitra, Mahesh Viswanathan, and Matthew Potok. C2e2: A verification tool for stateflow models. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 68–82. Springer, 2015.

[15] Gautier Dumonteil, Guido Manfredi, Michel Devy, Ambroise Confetti, and Daniel Sidobre. Reactive planning on a collaborative robot for industrial applications. In *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, volume 2, pages 450–457. IEEE, 2015.

[16] Georgios E Fainekos, Antoine Girard, Hadas Kress-Gazit, and George J Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343–352, 2009.

[17] Paolo Falcone, Francesco Borrelli, Jahan Asgari, Hongtei Eric Tseng, and Davor Hrovat. Predictive active steering control for autonomous vehicle systems. *IEEE*

Transactions on control systems technology, 15(3):566–580, 2007.

[18] Paolo Falcone, Francesco Borrelli, H Eric Tseng, Jahan Asgari, and Davor Hrovat. Linear time-varying model predictive control and its application to active steering systems: Stability analysis and experimental validation. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, 18(8):862–875, 2008.

[19] Michael P Georgeff and Amy L Lansky. Reactive reasoning and planning.

[20] Gabriel M Hoffmann, Claire J Tomlin, Michael Montemerlo, and Sebastian Thrun. Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing. In *2007 American Control Conference*, pages 2296–2301. IEEE, 2007.

[21] Kamal Kant and Steven W Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *The international journal of robotics research*, 5(3):72–89, 1986.

[22] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[23] Abolfazl Karimi, Manish Goyal, and Parasara Sridhar Duggirala. Safety and progress proofs for a reactive planner and controller for autonomous driving. *CoRR*, abs/2107.05815, 2021.

[24] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[25] M. Kloetzer and C. Belta. Automatic deployment of distributed teams of robots from temporal logic motion specifications. *IEEE Transactions on Robotics*, 2010.

[26] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE transactions on robotics*, 25(6):1370–1381, 2009.

[27] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[28] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 163–168. IEEE, 2011.

[29] Sarah M. Loos, André Platzer, and Ligia Nistor. Adaptive cruise control: Hybrid, distributed, and now formally verified. In Michael Butler and Wolfram Schulte, editors, *FM 2011: Formal Methods*, 2011.

[30] J. Lygeros, D. N. Godbole, and S. Sastry. Verified hybrid controllers for automated vehicles. *IEEE Transactions on Automatic Control*, 1998.

[31] Isaac Miller, Mark Campbell, Dan Huttenlocher, Frank-Robert Kline, Aaron Nathan, Sergei Lupashin, Jason Catlin, Brian Schimpf, Pete Moran, Noah Zych, et al. Team cornell's skynet: Robust perception and planning

in an urban environment. *Journal of Field Robotics*, 25 (8):493–527, 2008.

[32] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, et al. Junior: The stanford entry in the urban challenge. *Journal of field Robotics*, 25(9):569–597, 2008.

[33] Julien Moreau, Pierre Melchior, Stéphane Victor, Louis Cassany, Mathieu Moze, François Aioun, and Franck Guillemard. Reactive path planning in intersection for autonomous vehicle. *IFAC-PapersOnLine*, 52(5):109–114, 2019.

[34] Julien Moreau, Pierre Melchior, Stéphane Victor, Mathieu Moze, François Aioun, and Franck Guillemard. Reactive path planning for autonomous vehicle using bézier curve optimization. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1048–1053. IEEE, 2019.

[35] Richard M Murray and Sosale Shankara Sastry. Nonholonomic motion planning: Steering using sinusoids. *IEEE transactions on Automatic Control*, 38(5):700–716, 1993.

[36] P. Nilsson, O. Hussien, A. Balkan, Y. Chen, A. D. Ames, J. W. Grizzle, N. Ozay, H. Peng, and P. Tabuada. Correct-by-construction adaptive cruise control: Two approaches. *IEEE Transactions on Control Systems Technology*, 2016.

[37] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1 (1):33–55, 2016.

[38] Myung-Wook Park, Sang-Woo Lee, and Woo-Yong Han. Development of lateral control system for autonomous vehicle based on adaptive pure pursuit algorithm. In *2014 14th International Conference on Control, Automation and Systems (ICCAS)*, pages 1443–1447. IEEE, 2014.

[39] Scott Drew Pendleton, Hans Andersen, Xinxin Du, Xiaotong Shen, Malika Meghjani, You Hong Eng, Daniela Rus, and Marcelo H. Ang. Perception, planning, control, and coordination for autonomous vehicles. *Machines*, 5(1), 2017. ISSN 2075-1702. doi: 10.3390/machines5010006.

[40] Alejandro Perez, Robert Platt, George Konidaris, Leslie Kaelbling, and Tomas Lozano-Perez. Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *2012 IEEE International Conference on Robotics and Automation*, pages 2537–2542. IEEE, 2012.

[41] Clément Pêtrès, Miguel-Angel Romero-Ramirez, and Frédéric Plumet. Reactive path planning for autonomous sailboat. In *2011 15th International Conference on Advanced Robotics (ICAR)*, pages 112–117. IEEE, 2011.

[42] Pavithra Prabhakar, Parasara Sridhar Duggirala, Sayan Mitra, and Mahesh Viswanathan. Hybrid automata-based cegar for rectangular hybrid systems. *Formal Methods in System Design*, 46(2):105–134, 2015.

[43] Guilherme V Raffo, Guilherme K Gomes, Julio E Normey-Rico, Christian R Kelber, and Leandro B Becker. A predictive controller for autonomous vehicle path tracking. *IEEE transactions on intelligent transportation systems*, 10(1):92–102, 2009.

[44] Joshua Redding, Jayesh Amin, Jovan Boskovic, Yeonsik Kang, Karl Hedrick, Jason Howlett, and Scott Poll. A real-time obstacle detection and reactive path planning system for autonomous small-scale helicopters. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, page 6413, 2007.

[45] Rajnikant Sharma, Jeffery B Saunders, and Randal W Beard. Reactive path planning for micro air vehicles using bearing-only measurements. *Journal of Intelligent & Robotic Systems*, 65(1):409–416, 2012.

[46] Alexander Shkolnik, Matthew Walter, and Russ Tedrake. Reachability-guided sampling for planning under differential constraints. In *2009 IEEE International Conference on Robotics and Automation*, pages 2859–2865. IEEE, 2009.

[47] Jarrod M Snider et al. Automatic steering methods for autonomous automobile path tracking. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08*, 2009.

[48] Osamu Takahashi and Robert J Schilling. Motion planning in a plane using generalized voronoi diagrams. *IEEE Transactions on robotics and automation*, 5(2):143–150, 1989.

[49] Kirstin LR Talvala, Krisada Kritayakirana, and J Christian Gerdes. Pushing the limits: From lanekeeping to autonomous racing. *Annual Reviews in Control*, 35(1): 137–148, 2011.

[50] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692, 2006.

[51] Ashish Tiwari and Gaurav Khanna. Series of abstractions for hybrid automata. In *International Workshop on Hybrid Systems: Computation and Control*, pages 465–478. Springer, 2002.

[52] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[53] Cristian Ioan Vasile and Calin Belta. Reactive sampling-based temporal logic path planning. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4310–4315. IEEE, 2014.

[54] Cristian Ioan Vasile, Xiao Li, and Calin Belta. Reactive sampling-based path planning with temporal logic specifications. *The International Journal of Robotics Research*, 39(8):1002–1028, 2020.

[55] Wenda Xu, Junqing Wei, John M Dolan, Huijing Zhao, and Hongbin Zha. A real-time motion planner with trajectory optimization for autonomous vehicles. In *2012 IEEE International Conference on Robotics and Automation*, pages 2061–2067. IEEE, 2012.